The Hotlink Optimizer

Evangelos Kranakis [‡*]
School of Computer Science
Carleton University
Ottawa, Ontario, K1S 5B6, Canada

Danny Krizanc
Department of Mathematics
Wesleyan University
Middletown CT 06459, USA

Miguel Vargas Martin [‡†]
School of Computer Science
Carleton University
Ottawa, Ontario, K1S 5B6, Canada

Abstract

The Hotlink Optimizer (HotOpt) is a new software tool that helps administrators and designers to structure their websites efficiently. We propose an improvement on website access by making the most popular pages more accessible to users. Suppose that there is a page with high number of access hits; we want this page to be closer to the home page so visitors can reach it in less hops; this is done by adding hotlinks to the existing structure of the website. A hotlink is a link that provides a shortcut between two web pages within a website. HotOpt is empowered by a hotlink assignment algorithm that assigns hotlinks to web pages. In this paper we describe the architecture of HotOpt and show the simplicity of its usage and the usefulness of its outputs.

Keywords: website performance, hotlinks, web tools.

1 Introduction

A well designed website will contribute to the improvement of the Internet because well structured websites 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 websites become more attractive to users, who are more motivated to visit those websites offering rapid access to information. Our approach improves web performance by enhancing the design of websites: a well-designed website will avoid some useless traffic, save time to users, and reduce the web server work load. Our idea is conceptually simple: "bring the most popular pages closer to the home page."

We propose an improvement on website access by making the most popular pages more accessible to users. Suppose that there is a page with lots of access hits; we want this page to be closer to the home page so users can reach it in less hops. We do this by adding *hotlinks* to the existing structure of the website. A hotlink is a (unidirectional) link that provides a shortcut between two related web pages within a website.

In this paper we describe the architecture and user interface of the Hotlink Optimizer (HotOpt). HotOpt is a powerful software tool that assists administrators and designers in structuring their websites efficiently, according to user access patterns.

2 Related work

Several researchers have focused their attention on the optimal design of websites. Some of them ([1, 3, 8, 9, 10]) suggest analyzing user access patterns to help design better web pages, websites, and browsers. Perkowitz and Etzioni [8] propose the design of adaptive websites by promoting and demoting pages, highlighting

[*]Research supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) grant.

[‡]Research supported in part by MITACS (Mathematics of Information Technology and Complex Systems) NCE (National Center of Excellence) grant.

[†]Research supported in part by CONACYT (Science and Technology Council of Mexico).

links, adding hyperlinks, and clustering related pages. Perkowitz and Etzioni [9] presented an algorithm that analyses user access logs in order to identify candidate link 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 links to the documents in the cluster.

The algorithm was tested only in a particular website. The performance of the algorithm was 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 found that when the number of clusters is 1, the percentage of visits is around 72%; but when the number of clusters is 10, the percentage of visits is around 20%. They also compared clusters constructed by a human with clusters constructed by their algorithm; they found that the algorithm constructed clusters with at least 15% more visits than the human-authored ones.
Another way to improve website design by looking at user's access patterns was suggested by Srikant and Yang [10]. They propose improving website design by identifying the pages whose locations are different from their expected locations; that is, where visitors expected 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 she or he expected to find it. The algorithm was tested only in a particular website. They found that "many" pages were wrongly placed (according to their criteria).

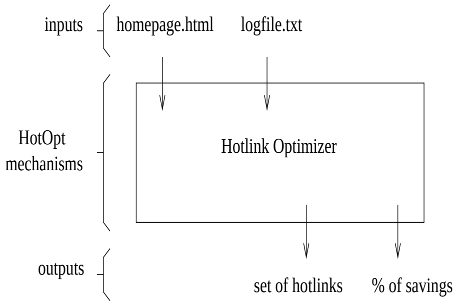

Figure 1: *Inputs and outputs of HotOpt.*

3 HotOpt tool

By analysing the link structure of a website, and looking at the user's patterns, HotOpt is able to suggest a set of hotlinks to be added to the website. This is a semi-automatic process in the sense that the hotlinks are found automatically, but they are not automatically added to the website. We want to assist web designers, not to replace them.
HotOpt is able to suggest a set of hotlinks H and thus save a certain proportion of the access cost of the website. The inputs are the *home page file* and the *access log files*, and the output is the set of hotlinks H and the percentage of savings offered by H, x. See Figure 1.

We can summarize the tasks performed by HotOpt in four basic processes (cf. Figure 2):

1. From 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 links between the web pages. After executing this process we are in *stage a*.

2. Construct a tree $T(V, E')$ performing a breadth first search on the graph G. The root of the tree is the node associated to the home page. After executing this process we will be in *stage b*.

3. Assign probabilities to the leaves of the tree by counting the number of times that each leaf was requested, according to the log files. The execution of this process will take us to *stage c*.

4. Finally, find an assignment of hotlinks that reduce the access cost of T. After execut-

ing this process we will end up in *stage d*. Then, output the set of hotlinks found along with the savings that those hotlinks can achieve.

Based on previous simulation results (see [2] and [6]), we concluded that the best hotlink assignment algorithm known so far is *greedyBFS*. This is the algorithm implemented by HotOpt. HotOpt was built with Java, and contains over $1,500$ lines of code, which compose 7 classes.
As far as we know, before we published our first paper on hotlinks [1], there were no contributions suggesting to add hotlinks to the underlying structure of websites in order to improve their performance.

4 Notation and terminology

A *website* is a collection of web pages administered by the same authority which are linked together to form a unified source of information. There is exactly one unique page in every website, which is called the *home page*. The home page is considered to be the starting point of any website, and it is assumed that any web page belonging to the website can be reached from the home page. Under this distinction, we can say that, within a website, a web page b is a *descendant* of a web page a if there is a path of links leading from a to b; therefore, any web page is a descendant of the home page.
Consider a website as a directed graph $G(V, E)$, where web pages are nodes connected by links. Let us call the node that represents the home page, the root, which we denote by r. The *expected number of steps* to reach a page is also called the *access cost* of the website, and is denoted by $E[G]$. Let p_i be the probability that page i is visited by a user, and let $d(i)$ be the number of edges of the shortest path between r and i, then,

$$E[G] = \sum_{v \in V} d(v) \cdot p_v \qquad (1)$$

A hotlink*, h,* is a directed edge, $(s,t) \notin E$, added to the original graph, such that there is a directed path from s to t in the original graph, where $s, t \in V$. Consider a set $H = \{h_1, \ldots, h_k\}$ of hotlinks assigned to G; the resulting graph is denoted by G^H, and the percentage of *savings* of H is defined by $S(H) = \frac{100(E[G]-E[G^H])}{E[G]}$. A set of hotlinks H is optimal iff $S(H) \geq S(H')$ for any hotlink set H'. The problem of finding an optimal set of hotlinks with a bounded number of hotlinks added per node is referred to as the *optimal hotlink assignment problem*.
The optimal hotlink assignment problem was proven to be NP-hard for general graphs in [1]. In [2], the authors approached the problem for trees, and designed heuristic algorithms that were shown to attain good savings under a variety of circumstances. However, we must be specially careful when we transform a website into a tree, as we have to respect the semantic relationship between the pages, and to keep the shortest distance between the home page and any other page of the website. In section 5.1 we give a breadth-first-search based algorithm that performs this transformation.

5 Components and mechanisms of HotOpt

The tasks performed by HotOpt can be summarized as follows. First, we apply a transformation to the website in order to obtain a web tree. Secondly, we assign an access probability distribution to the leaves of the tree. Finally, we apply the optimization algorithm. Note that the log files are only used to obtain the access probability of the website, and they do not play any role at all in the transformation of a website into a tree.

5.1 Transformation of a website into a web tree

In this section we explain the process of the transformation of a website into a tree. This process corresponds to the task that results in *stage b* in Figure 2.
Consider a website as a directed graph, where web pages are nodes connected by links. It has been proven that the hotlink assignment problem is NP-hard for general graphs [1]. There-

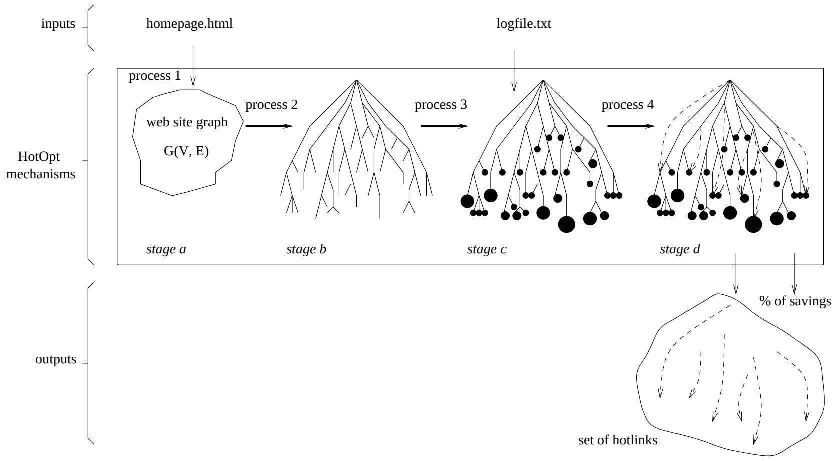

Figure 2: *The tasks performed by HotOpt can be summarized in four basic processes. Process 1 constructs a directed graph, where the nodes are web pages and the arcs are the links that connect the pages. Process 2 builds a web tree in a breadth first search order with the home page as the root node. Process 3 assigns probabilities to the leaves of the tree based on the access log files. Process 4 is crucial since it is responsible for applying the optimization algorithm (*greedyBFS*).*

fore, we approach the problem for arbitrary trees [2]. However, we must be careful not to break the semantic structure, and to keep the shortest path from the home page to all the other pages of the website.

We construct a tree in breadth first search order, whose root will be the node associated to the home page. The process is formally described in Algorithm 1. The algorithm receives as parameters a graph $G(V, E)$ and a distinctive node r associated to the home page of the website, and 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; we will see why in the next sections.

5.2 Assignment of the access probability distribution to the web tree

In this section we describe the process of assigning the probability distribution to the leaves of the tree built in section 5.1.

Consider the access logs file of the website. We want to extract the number of times that a leaf was requested by a user. We will look only at the entries with http code 200 or 304 ("OK", and "Not Modified", see [5] for a complete de-

Algorithm 1 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 link $(u, v) \in E$: v has not been visited*
      i. *enqueue v in q*
      ii. *mark v as visited*
      iii. $E' = E' \cup \{(u, v)\}$
7. *return*($T(V, E'), r$)

scription of the http 1.1 protocol) because they are the only ones that indicate an actual access to the page. We describe the extraction process in Algorithm 2. The algorithm receives as inputs a tree T and a log file. After the execution of the algorithm, each leaf of the tree will be associated to a number that represents the number of times that it was requested (according to the log file).

Algorithm 2 extractrequests(T, *logfile*)

1. *for each leaf l of T*
   (a) *find all the requests to file l in the* logfile

Once the algorithm *extractrequests* has been executed, we will compute the access probability distribution of this website. Let a_l be the number of times that leaf l was requested, and let $A = \sum_l a_l$, where l is a leaf, be the total number of times that all the leaves were requested. Then the access probability of leaf l is defined by $p_l = \frac{a_l}{A}$.
After we have computed the access probability distribution, we need to assign weights to the internal nodes of the tree in a bottom-up fashion, in such a way that the weight of an internal node v, denoted also as p_v is the sum of the weights (or access probabilities in the case of the parents of leaves) of its children. Observe that for the root r, $p_r = 1$.

5.3 Optimization algorithm

The optimization algorithm is the core of HotOpt. Several optimization algorithms have been proposed. See for example [2, 7]. (See [1] for assignment of hotlinks to full binary trees and [4] for assignment in complete trees.) Among those we have tested the best performing algorithm is called *greedyBFS*, and was presented in [2]. The optimization algorithm for assigning at most one hotlink per node is described in Algorithm 3. It takes a tree T as input, and outputs a set of hotlinks H. The function *nextBFS()* returns the nodes of the tree in breadth first search order starting from the root.
We tested *greedyBFS* on the *www.scs.carleton.ca* website. This website contains 798 web pages. The algorithm was able to save over 27.24% of the access cost of the website (cf. Equation 1) by assigning no more than 64 hotlinks. This result was not enough to conclude that *greedyBFS* is a good algorithm. Therefore, we performed simulations, which we describe in Section 5.3.1. With the simulations we found that *greedyBFS* is able to save up to 25% over the access cost of a website.

Algorithm 3 greedyBFS(T)

1. $H = \phi$
2. *while((s = nextBFS()) $\neq \phi$)*
   (a) *$t = v : v$ minimizes equation 1; and v is a descendant of s; and v has not been assigned a (incoming) hotlink; and v is not a descendant of a node x, which is at a higher level than s and already has been assigned a (incoming) hotlink.*[||]
   (b) *if $t \neq \phi$ then* $H = H \bigcup \{(s,t)\}$

5.3.1 Simulations

In this section we present the results obtained with simulation. All our simulations were implemented with Java. We evaluated our algorithms on two different kinds of structures: power-law trees with Zipf's distribution, and real trees with Zipf's distribution. Figures 3 and 4 illustrate the results of the simulations. The simulations were described in more detail in [2]. From Figure 3, note that the percentage of savings oscillates between narrow intervals, which indicates that the size of the tree does not greatly affect the performance. The average number of nodes of the real websites sample is $9,669.2$, however ob-

[||] We say that the levels of the tree are counted in increasing order starting from the root, in such a way that the root is at level 0.

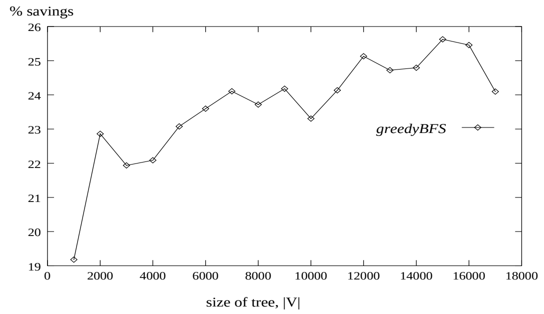

Figure 3: *Gain obtained by applying* greedyBFS, recursive*, and* 2-greedyBFS *to random power-law trees of size* $|V| = n$ *(for* $n = 1000 \ldots 17000$*) with Zipf's probability distribution over the leaves. The savings vary from* 19.18% *to* 25.63%*.*

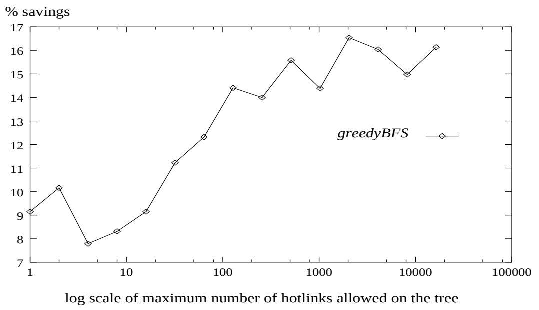

Figure 4: *Average gain obtained by applying* greedyBFS *to real trees with Zipf's probability distribution over the leaves. The average size of the websites was* 9,669.2*. The best saving attained was* 16.04%*, and was attained with a set of around* 2,048 *hotlinks.*

serve in Figure 4 that the algorithm stabilizes after $|H| \geq 2,048 \ll 9,669.2$.

6 Architecture

Figure 5 illustrates the main structure of HotOpt. The *tree constructor* module takes the home page and the access log files as input. This module is responsible for performing the first three processes depicted in Figure 2. The *link structure* sub-module is responsible for transforming the website into a web tree. The transformation process was described in Section 5.1, and includes processes 1 and 2 of Figure 2.

The *access probabilities* sub-module assigns

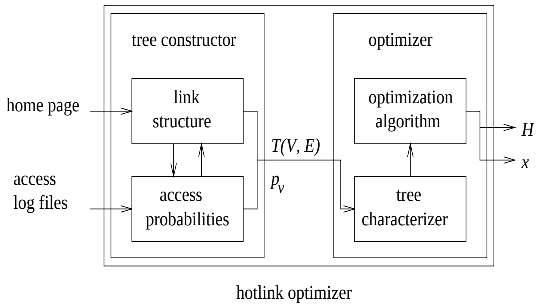

Figure 5: *Main structure of HotOpt.*

probabilities to the nodes of the tree. This process is described in Section 5.2, and corresponds to process 3 of Figure 2. The *tree constructor* module outputs a web tree, $T(V, E)$, with a probability distribution p, over its leaves.

The *optimization algorithm* sub-module applies the best algorithm depending on the characteristics of the tree T. Such characteristics are extracted by the sub-module *tree characterizer*. So far, we apply the algorithm *greedyBFS* to any tree regardless of its characteristics, as we have not been able to find out what characteristics we should take into account; moreover, *greedyBFS* has proven to be the best in all situations we have examined so far (cf. [2]). The *optimization algorithm* sub-module performs the tasks described in Section 5.3 and corresponds to process 4 of Figure 2. The final outputs are the set of hotlinks H and the savings percentage, x, offered by H.

7 The user interface

7.1 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 of his or her system. It is important to remark that HotOpt will not delete or modify any existing file of the collection nor its structure.

7.2 Input

The user interface is quite simple and is self-descriptive. Figure 6 shows the main display window of HotOpt. HotOpt must be fed with

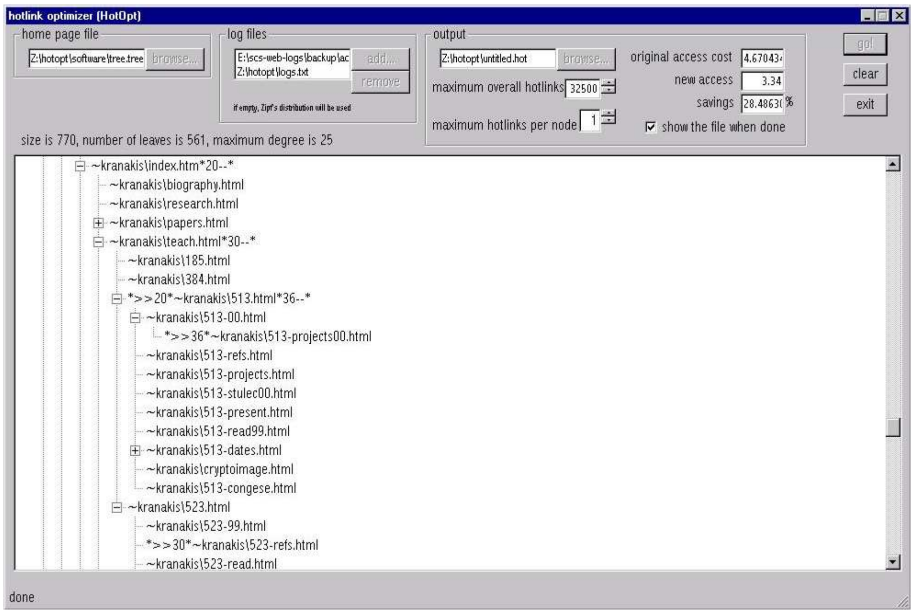

Figure 6: *Main window of HotOpt. The link structure of the website is transformed into a web tree. The set of hotlinks found is* $H = \{h_1, \dots, h_k\}$*. The incident web pages of a hotlink* h_i*, for* $1 \leq i \leq k$*, are marked with the symbols* *i–* *and* *»i* *(the outgoing and incoming page, respectively).*

the home page and the log files. If no log files are specified, HotOpt will assign arbitrarily a Zipf's distribution. The user may specify the output file where the set of hotlinks is to be stored. It is also possible to limit the overall budget of hotlinks to be assigned, as well as the maximum number of hotlinks per page. Once everything has been set up, the optimization process starts by pressing the button "go!". Here, the four processes illustrated in Figure 2 will be executed one by one.

7.3 Output

HotOpt adopts the link structure of the website, then it is transformed into a tree. This tree is shown in the main window, along with the hotlinks, as depicted in Figure 6. The hotlinks can be easily identified by a unique number that is assigned to each of them. HotOpt also shows the proportion of the access cost that can be saved with the set of hotlinks found. It also writes the set of hotlinks in the file specified by the user. If the check box saying "show the file when done" was checked, then, a new window appears, listing the set of hotlinks in the output file.

8 Conclusions and future work

We have presented a friendly and powerful tool capable of suggesting a set of hotlinks that could save up to 25% of the access cost of an arbitrary website. We are still working on the design and testing of other hotlink assignment algorithms. HotOpt will be enhanced if we find a better algorithm.

The savings described above are the theoretical maximum assuming that users regularly take advantage of suggested hotlinks. In order to further validate the usefulness of hotlinks, we plan on experimenting with actual users of a real website to determine how to best display them to achieve the best results.

The construction of a tree may not consider important links between pages; it would be interesting to evaluate new ways of constructing the tree that take into consideration the "importance" of the links. On the other hand, users may be interested on intermediate nodes, not only on leaves; therefore, new optimization algorithms should take this as a premise.

We envision automatically adaptive websites that would work in the following way. Each web page belonging to the website will have within its source html code a special encoding that will be interpreted by HotOpt as the place where the hotlinks should be inserted. The maximum number of hotlinks per web page could also be specified; even more, we could prevent HotOpt from putting hotlinks in certain pages. As a result of this enhancement to the html source code, HotOpt could execute itself periodically in order to offer a better access cost at all times.

References

[1] 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.

[2] 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 Second International Conference on Internet Computing (IC 2001)*, volume 2, pages 793–799, Las Vegas, Nevada, June 25–28 2001. CSREA Press.

[3] 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, NY, September 23–26 1998. ACM Press.

[4] Sven Fuhrmann, Sven Oliver Krumke, and Hans-Christoph Wirth. Multiple hotlink assignment. In Andreas Brandstädt and Van Bang Le, editors, *Proceedings of the Twenty-Seventh International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2001)*, volume 2204 of *Lecture Notes in Computer Science*, pages 189–200, Boltenhagen, Germany, 2001. Springer.

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

[6] Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Miguel Vargas Martin. Evaluation of hotlinks assignment algorithms. In preparation, 2002.

[7] 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.

[8] 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.

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

[10] 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.