Towards Adaptive Web Sites: Conceptual Framework and Case Study

Mike Perkowitz Oren Etzioni
Department of Computer Science and Engineering, Box 352350
University of Washington, Seattle, WA 98195
{map, etzioni}@cs.washington.edu
(206) 616-1845 Fax: (206) 543-2969

Abstract

The creation of a complex web site is a thorny problem in user interface design. In this paper we explore the notion of **adaptive web sites**: sites that semi-automatically improve their organization and presentation by learning from visitor access patterns. It is easy to imagine and implement web sites that offer shortcuts to popular pages. Are more sophisticated adaptive web sites feasible? What degree of automation can we achieve?

To address the questions above, we describe the design space of adaptive web sites and consider a case study: the problem of synthesizing new *index pages* that facilitate navigation of a web site. We present the PageGather algorithm, which automatically identifies candidate link sets to include in index pages based on user access logs. We demonstrate experimentally that PageGather outperforms the Apriori data mining algorithm on this task. In addition, we compare PageGather's link sets to pre-existing, human-authored index pages.

Keywords: adaptive, clustering, data mining

1 Introduction and Motivation

Designing a rich web site so that it readily yields its information can be tricky. Unlike the proverbial oyster that contains a single pearl, a web site often contains myriad facts, images, and hyperlinks. Many different visitors approach a popular web site — each with his or her own goals and concerns. Consider, for example, the web site for a typical computer science department. The site contains an amalgam of research project descriptions, course information, lists of graduating students, pointers to industrial affiliates, and much more. Each nugget of information is of value to someone who would like to access it readily. One might think that a well organized hierarchy would solve this problem, but we've all had the experience of banging our heads against a web site and crying out "it's got to be here *somewhere...*".

The problem of good web design is compounded by several factors. First, different visitors have distinct goals. Second, the same visitor may seek different information at different times. Third, many sites outgrow their original design, accumulating links and pages in unlikely places. Fourth, a site may be designed for a particular kind of use, but be used in many different ways in practice; the designer's *a priori* expectations may be violated. Too often web site designs are fossils cast in HTML, while web navigation is dynamic, time-dependent, and idiosyncratic. In

[13], we challenged the AI community to address this problem by creating **adaptive web sites:** *sites that semi-automatically improve their organization and presentation by learning from visitor access patterns.*

Many web sites can be viewed as user interfaces to complex information stores. However, in contrast to standard user interfaces, where data on user behavior has to be gathered in expensive (and artificial) focus groups and usability labs, web server logs automatically record user behavior at the site. We posit that adaptive web sites could become a valuable method of mining this data with the goal of continually tuning the site to its user population's needs.

While adaptive web sites are a potentially valuable, their feasibility is unclear *a priori*: can non-trivial adaptations be automated? will adaptive web sites run amok, yielding chaos rather than improvement? what is an appropriate division of labor between the automated system and the human webmaster? To investigate these issues empirically, we analyze the problem of *index page synthesis.*[1] We focus on one subproblem (generating candidate link sets to include in index pages) as amenable to automation and describe the PageGather algorithm, which solves it.

The remainder of this paper is organized as follows. We next discuss the design space of adaptive web sites and present previous work in this area. We then present design desiderata which motivate our own approach. In section 2, we define the index page synthesis problem, the focus of our case study. We then present PageGather, analyzing variants of PageGather and both data mining and clustering algorithms as potential solutions. In section 3, we experimentally evaluate variants of PageGather, and compare the performance of PageGather to that of Apriori, the classical data mining algorithm for the discovery of frequent sets [3]. We also compare PageGather's output to pre-existing, human-authored index pages available at our experimental web site. We conclude with a discussion of future work and a summary of our contributions.

1.1 Design Space

Adaptive web sites vary along a number of design axes.

- **Types of adaptations.** New pages may be created. Links may be added or removed, highlighted or rearranged. Text, link labels, or formatting may be altered.
- **Customization vs. Transformation.** Customization is modifying a web site to suit the needs of an individual user; customization necessitates creating a large number of versions of the web site — one for each user. In contrast, transformation involves altering the site to make navigation easier for a large set of users. For example, a university web site may be reorganized to support one "view" for faculty members and a distinct view for students. In addition, certain transformations may seek to improve the site for *all* visitors.
- **Content-based vs. Access-based.** A site that uses content-based adaptation organizes and presents pages based on their content — what the pages say and what they are about. Access-based adaptation uses the way past visitors have interacted with the site to guide how information is structured. Naturally, content-based and access-based adaptations are complementary and may be used together.
- **Degree of automation.** Excite and Yahoo's manually personalized home pages are a simple example of customization; we are interested in more *automatic* adaptation techniques. However, for feasibility, adaptive web sites are likely to be only partially automated.

We now survey previous work on adaptive web site using the vocabulary and distinctions introduced above.

[1] An *index page* is a page consisting of links to a set of pages that cover a particular topic (e.g., electric guitars).

1.2 Previous Work

It is quite common for web sites to allow users to customize the site for themselves. Common manual customizations include lists of favorite links, stock quotes of interest, and local weather reports. Slightly automated customizations include records of previous interactions with the site and references to pages that have changed since the previous visit. Some sites also allow users to describe interests and will present information — news articles, for example — relevant to those interests.

More sophisticated sites attempt *path prediction*: guessing where the user wants to go and taking her there immediately (or at least providing a link). The WebWatcher [5][2] learns to predict what links users will follow on a particular page as a function of their specified interests. A link that WebWatcher believes a particular user is likely to follow will be highlighted graphically and duplicated at the top of the page when it is presented. Visitors to a site are asked, in broad terms, what they are looking for. Before they depart, they are asked if they have found what they wanted. WebWatcher takes an access-based approach, using the paths of people who indicated success as examples of successful navigations. If, for example, many people who were looking for "personal home pages" follow the "people" link, then WebWatcher will tend to highlight that link for future visitors with the same goal. Note that, because WebWatcher groups people based on their stated interests rather than customizing to each individual, it falls on the continuum between pure customization and pure transformation.

A site may also try to customize to a user by trying to guess her general interests dynamically as she browses. The AVANTI Project [7][3] focuses on dynamic customization based on users' needs and tastes. As with the WebWatcher, AVANTI relies partly on users providing information about themselves when they enter the site. Based on what it knows about the user, AVANTI attempts to predict both the user's eventual goal and her likely next step. AVANTI will prominently present links leading directly to pages it thinks a user will want to see. Additionally, AVANTI will highlight links that accord with the user's interests.

Another form of customization is based on *collaborative filtering*. In collaborative filtering, users rate objects (e.g. web pages or movies) based on how much they like them. Users that tend to give similar ratings to similar objects are presumed to have similar tastes; when a user seeks recommendations of new objects, the site suggests those objects that were highly rated by other users with similar tastes. The site recommends objects based solely on other users' ratings or accesses, ignoring the content of the objects themselves. A simple form of collaborative filtering is used by, for example, Amazon.com; the web page for a particular book may have links to other books commonly purchased by people who bought this one. Firefly[4] uses a more individualized form of collaborative filtering in which members may rate hundreds of CDs or movies, building up a very detailed personal profile; Firefly then compares this profile with those of other members to make new recommendations.

Footprints [23] takes an access-based transformation approach. Their motivating metaphor is that of travelers creating footpaths in the grass over time. Visitors to a web site leave their "footprints" behind, in the form of counts of how often each link is traversed; over time, "paths" accumulate in the most heavily traveled areas. New visitors to the site can use these well-worn paths as indicators of the most interesting pages to visit. Footprints are left automatically (and anonymously), and any visitor to the site may see them; visitors need not provide any information about themselves in order to take advantage of the system. Footprints provides essentially *localized*

[2] `http://www.cs.cmu.edu/~webwatcher/`
[3] `http://zeus.gmd.de/projects/avanti.html`
[4] `http://www.firefly.com`

information; the user sees only how often links between adjacent pages are traveled.

A web site's ability to adapt could be enhanced by providing it with *meta-information*: information about its content, structure, and organization. One way to provide meta-information is to represent the site's content in a formal framework with precisely defined semantics, such as a database or a semantic network. The use of meta-information to customize or optimize web sites has been explored in a number of projects (see, for example, XML annotations [9], Apple's Meta-Content Format, and other projects [6, 11]). One example of this approach is the STRUDEL web-site management system [6] which attempts to separate the information available at a web site from its graphical presentation. Instead of manipulating web sites at the level of pages and links, web sites may be specified using STRUDEL's view-definition language. With all of the site's *content* so encoded, its *presentation* may be easily adapted.

A number of projects have explored *client-side customization*, in which a user has her own associated agent who learns about her interests and customizes her web experience accordingly. The AiA project [4, 17] explores the customization of web page information by adding a "presentation agent" who can direct the user's attention to topics of interest. The agent has a model of the individual user's needs, preferences, and interests and uses this model to decide what information to highlight and how to present it. In the AiA model, the presentation agent is on the client side, but similar techniques could be applied to customized presentation by a web server as well. Letizia [10] is a personal agent that learns a model of its user by observing her behavior. Letizia explores the web ahead of the user (investigating links off of the current page) and uses its user model to recommend pages it thinks the user will enjoy. Other projects have investigated performing customization at neither the client nor the server but as part of the network in between, particularly by using transcoding proxies. Transend [8], for example, is a proxy server at the University of California at Berkeley that performs image compression and allows each of thousands of users to customize the degree of compression, the interface for image refinement, and the web pages to which compression is applied.

1.3 Our Approach

Our survey of other work in this area has led us to formulate five desiderata for an adaptive web site.

1. **Avoid creating work for visitors (e.g. filling out questionnaires).** Visitors to a site are turned off by extra work, especially if it has no clear reward, and may opt out rather than participate. Furthermore, if the site cannot improve itself without feedback, it will fail if users do not assist.

2. **Make the web site easier to use for everyone, including first-time users, casual users, etc.** Customization can be genuinely useful for repeat visitors, but does not benefit first-time users. In addition, one user's customizations do not apply to other users; there is no sharing or aggregation of information across multiple users. Transformation has the potential to overcome both limitations.

3. **Use web sites as they are, without presupposing the availability of meta-information not currently available (e.g., XML annotations).** Perhaps web sites of the future will be heavily annotated with both semantic and structural meta-information. However, we would like to make it possible to transform today's existing web sites into adaptive ones, without making assumptions regarding the future of the web.

```
24hrlab-214.sfsu.edu - - [21/Nov/1996:00:01:05 -0800] "GET /home/jones/collectors.html HTTP/1.0" 200 13119
24hrlab-214.sfsu.edu - - [21/Nov/1996:00:01:06 -0800] "GET /home/jones/madewithmac.gif HTTP/1.0" 200 855
24hrlab-214.sfsu.edu - - [21/Nov/1996:00:01:06 -0800] "GET /home/jones/gustop2.gif HTTP/1.0" 200 25460
x67-122.ejack.umn.edu - - [21/Nov/1996:00:01:08 -0800] "GET /home/rich/aircrafts.html HTTP/1.0" 404 617
x67-122.ejack.umn.edu - - [21/Nov/1996:00:01:08 -0800] "GET /general/info.gif HTTP/1.0" 200 331
203.147.0.10 - - [21/Nov/1996:00:01:09 -0800] "GET /home/smith/kitty.html HTTP/1.0" 200 5160
24hrlab-214.sfsu.edu - - [21/Nov/1996:00:01:10 -0800] "GET /home/jones/thumbnails/awing-bo.gif HTTP/1.0" 200 5117
```

Figure 1: Typical user access logs, these from a computer science web site. Each entry corresponds to a single request to the server and includes originating machine, time, and URL requested. Note the series of accesses from each of two users (one from SFSU, one from UMN).

4. **Protect the site's original design from destructive changes.** Web designers put a great deal of effort into designing web sites. While we may be able to assist them, we do not want to replace them or undo their work.

5. **Keep the human webmaster in control.** Clearly, the human webmaster needs to remain in control of the web site in the foreseeable future both to gain her trust in automatic adaptive techniques and to avoid "disasters."

We took the above desiderata as constraints on our approach to creating adaptive web sites. We use transformation rather than customization, both to avoid confronting visitors with questionnaires and to facilitate the sharing of site improvements for a wide range of visitors. We focus on an access-based approach, as automatic understanding of free text is difficult. We do not assume any annotations on Web pages beyond HTML. For safety, we limit ourselves to *nondestructive transformations*: changes to the site that leave existing structure intact. We may add links but not remove them, create pages but not destroy them, add new structures but not scramble existing ones. Finally, we restrict ourselves to generating candidate adaptations and presenting them to the human webmaster — any non-trivial changes to the web site are under webmaster control.

The main source of information we rely on is the site's web server log, which records the pages visited by a user at the site. Our underlying intuition is what we call the **visit-coherence assumption**: the pages a user visits during one interaction with the site tend to be conceptually related. We do not assume that *all* pages in a single visit are related. After all, the information we glean from individual visits is noisy; for example, a visitor may pursue multiple distinct tasks in a single visit. However, if a large number of visitors continue to visit and re-visit the same set of pages, that provides strong evidence that the pages in the set are related. Thus, we accumulate statistics over many visits by numerous visitors and search for overall trends.

It is not difficult to devise a number of simple, non-destructive transformations that could improve a site; we describe several in [14]. Examples include highlighting popular links, *promoting* popular links to the top of a page or to the site's front page, and linking together pages that seem to be related. We have implemented one such transformation: *shortcutting*, in which we attempt to provide links on each page to visitors' eventual goals, thus skipping the in-between pages. As reported in [13], we found a significant number of visitors used these automatic shortcuts.

However, our long-term goal is to demonstrate that more fundamental adaptations are feasible. An example of this is *change in view*, where a site could offer an alternative organization of its contents based on user access patterns. Consider, for example, the **Music Machines** web site,[5]

[5] `http://machines.hyperreal.org`

which has been our primary testbed, as it is maintained by one of the authors, and we have full access to all documents and access logs. Music Machines is devoted to information about various kinds of electronic musical instruments. Most of the data at the site is organized by the manufacturer of the instrument and the particular model number. That is, there is a page for the manufacturer *Roland* and, on that page, links to pages for each instrument Roland produces. However, imagine a visitor to the site who is interested in a comprehensive overview of all the keyboards available from various manufacturers. She would have to first visit the Roland page and look at each of the Roland keyboards, then visit each of the other keyboard manufacturers for its offerings as well. Now, imagine if the site repeatedly observed this kind of behavior and automatically created a new web page containing all the links to all the keyboards. Now our visitor need only visit this new page rather than search for all the keyboards. This page represents a change in view, from the former "manufacturer-centric" organization to one based on type of instrument. If we can discover these user access patterns and create *new* web pages to facilitate them, we should in theory be able to create new views of the site.

2 A Case Study: Index Page Synthesis

Is automatic change in view feasible in practice? As a first step, we investigate the automatic synthesis of new index pages. Index pages are central to site organization. If we are able to generate index pages that are valued and adopted by the human webmaster, we can begin to extend and elaborate the kind of organizational changes suggested. In this section, we define the index page synthesis problem and present an algorithm — PageGather — that solves part of this problem.

2.1 The Index Page Synthesis Problem

Page synthesis is the automatic creation of web pages. An *index page* is a page consisting of links to a set of pages that cover a particular topic (e.g., electric guitars). Given this terminology we define the **index page synthesis problem**: given a web site and a visitor access log, create new index pages containing collections of links to related but currently unlinked pages. An access log is a document containing one entry for each page requested of the web server. Each request lists at least the origin (IP address) of the request, the URL requested, and the time of the request. *Related but unlinked* pages are pages that share a common topic but are not currently linked at the site; two pages are considered linked if there exists a link from one to the other or if there exists a page that links to both of them.

The problem of synthesizing a new index page can be decomposed into several subproblems.

1. **What are the contents (i.e. hyperlinks) of the index page?**
2. How are the hyperlinks on the page ordered?
3. How are the hyperlinks labeled?
4. What is the title of the page? Does it correspond to a coherent concept?
5. Is it appropriate to add the page to the site? If so, where?

In this paper, we focus on the first subproblem — generating the *contents* of the new web page. The remaining subproblems are topics for future work. We note that several subproblems, particularly the last one, are quite difficult and will be solved in collaboration with the site's human webmaster.

Nevertheless, we show that the task of generating candidate index page contents can be automated with some success using the PageGather algorithm described below.

2.2 The PageGather Algorithm

In this section, we introduce *PageGather*. Given a large access log, our task is to find collections of pages that tend to co-occur in visits. Clustering (see [22, 16, 24]) is a natural technique to consider for this task. In clustering, documents are represented in an N-dimensional space (for example, as word vectors). Roughly, a cluster is a collection of documents close to each other and relatively distant from other clusters. Standard clustering algorithms *partition* the documents into a set of mutually exclusive clusters.

Cluster *mining* is a variation on traditional clustering that is well suited to our task. Instead of attempting to partition the entire space of documents, we try to find a small number of high quality clusters. Furthermore, whereas traditional clustering is concerned with placing each document in exactly one cluster, cluster mining may place a single document in multiple overlapping clusters. The relationship between traditional clustering and cluster mining is parallel to that between classification and data mining as described in [20]. Segal contrasts mining "nuggets" — finding high-accuracy rules that capture patterns in the data — with traditional classification — classifying *all* examples as positive or negative — and shows that traditional classification algorithms do not make the best mining algorithms.

The *PageGather algorithm* uses cluster mining to find collections of related pages at a web site, relying on the visit-coherence assumption. In essence, PageGather takes a web server access log as input and maps it into a form ready for clustering; it then applies cluster mining to the data and produces candidate index-page contents as output. The algorithm has five basic steps:

1. Process the access log into visits.
2. Compute the co-occurrence frequencies between pages and create a similarity matrix.
3. Create the graph corresponding to the matrix, and find maximal cliques (or connected components) in the graph.
4. Rank the clusters found, and choose which to output.
5. For each cluster, create a web page consisting of links to the documents in the cluster, and present it to the webmaster for evaluation.

We discuss each step in turn.

1. Process the access log into visits. As defined above, a visit is an ordered sequence of pages accessed by a single user in a single session. An access log, however, is a sequence of page views, or requests made to the web server.[6] Each request typically includes the time of the request, the URL requested, and the machine from which the request originated. For our purposes, however, we need to extract discrete visits from the log. We first assume that each originating machine corresponds to a single visitor.[7] A series of page views in a day's log from one visitor,[8]

[6] A web site is restricted to a collection of HTML documents residing at a single server — we are not yet able to handle dynamically-generated pages or multiple servers.

[7] In fact, this is not necessarily the case. Many Internet service providers channel their users' HTTP requests through a small number of gateway machines, and two users might simultaneously visit the site from the same machine. Fortunately, such coincidences are too uncommon to affect the data significantly; if necessary, however, more accurate logs can be generated using cookies or visitor-tracking software such as WebThreads.

[8] We consider an entire day's page views to be one visit, even if a user made, for example, one morning visit and one evening visit. This simplification does not greatly affect the data; if necessary, however, the series of page views could be divided at significant time gaps.

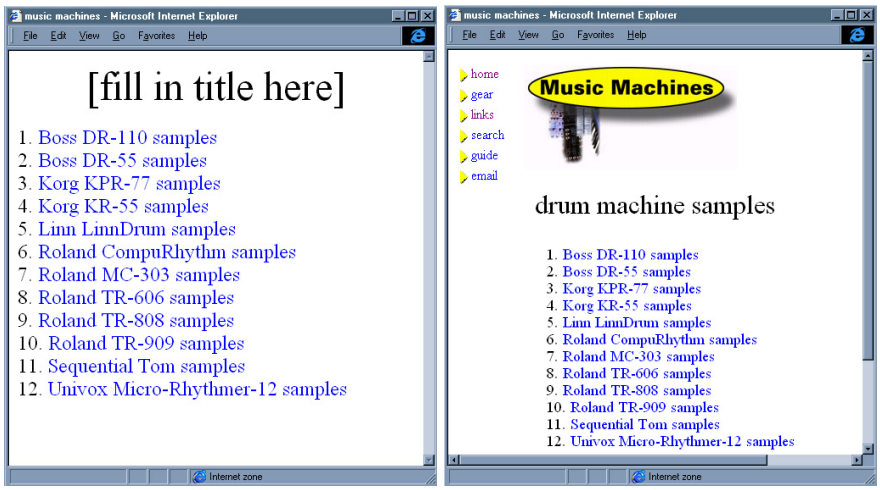


Figure 2: (a) A candidate cluster to be presented to the webmaster for approval and naming. (b) How the final page would appear at the site, properly named and formatted.

ordered by their time-stamps, corresponds to a single session for that visitor.

2. Compute the co-occurrence frequencies between pages and create a similarity matrix. For each pair of pages p_1 and p_2, we compute $P(p_1|p_2)$, the probability of a visitor visiting p_1 if she has already visited p_2 and $P(p_2|p_1)$, the probability of a visitor visiting p_2 if she has already visited p_1. The co-occurrence frequency between p_1 and p_2 is the minimum of these values.

We use the minimum of the two conditional probabilities to avoid mistaking an asymmetrical relationship for a true case of similarity. For example, a popular page p_1 might be on the most common path to a more obscure page p_2. In such a case $P(p_1|p_2)$ will be high, perhaps leading us to think the pages similar. However, $P(p_2|p_1)$ could be quite low, as p_1 is on the path to many pages and p_2 is relatively obscure.

As stated above, our goal is to find clusters of related *but currently unlinked* pages. Therefore, we wish to avoid finding clusters of pages that are already linked together. We prevent this by setting the matrix cell for two pages to zero if they are already linked in the site.

We observed that the similarity matrix could be viewed as a graph, which enables us to apply graph algorithms to the task of identifying collections of related pages. However, a graph corresponding to the similarity matrix would be completely (or almost completely) connected. In order to reduce noise, we apply a threshold and remove edges corresponding to low co-occurrence frequency.

3. Create the graph corresponding to the matrix, and find maximal cliques (or connected components) in the graph. We create a graph in which each page is a node and each nonzero cell in the matrix is an arc. Next, we apply graph algorithms that efficiently extract connectivity information from the graph (e.g., the linear-time algorithm for identifying connected components). The frequency information on the arcs is ignored in this step for the sake of efficiency. By creating a sparse graph, and using efficient graph algorithms for cluster mining, we can identify high quality clusters substantially faster than by relying on traditional clustering methods [15]. We may extract two kinds of subgraphs, leading to two variants of the PageGather algorithm.

Rank	Topic	Coherent?	Useful?
1	Images of Korg keyboards	yes	somewhat
2	Images of Korg keyboards	somewhat	somewhat
3	Sequencers	yes	yes
4	Sequencers	yes	yes
5	Roland Instruments	no	no
6	Samplers	yes	yes
7	Instrument samples	yes	somewhat
8	Samplers	yes	yes
9	Instrument samples	yes	somewhat
10	Books, do-it-yourself info	somewhat	yes

Figure 3: Clusters found by PG_{CLIQUE} with overlap reduction. Clusters are ordered as ranked by PageGather. Topics are provided by the webmaster based on the cluster contents. All clusters are subjectively rated by the webmaster for the following qualities: **coherence** is the degree to which all of the cluster contents correspond to the apparent topic; **usefulness** rates whether the topic would be of interest to the site's users. Note that, in spite of overlap reduction, multiple clusters with the same topic are found. *Korg* and *Roland* are instrument manufacturers; *keyboards*, *sequencers*, and *samplers* are types of instruments.

- PG_{CLIQUE} finds all *maximal cliques* in the graph. A clique is a subgraph in which every pair of nodes has an edge between them; a maximal clique is not a subset of any larger clique. For efficiency, we bound the size of discovered cliques.[9]
- PG_{CC} finds all *connected components* — subgraphs in which every pair of pages has a path of edges between them.

In step 2, we applied a threshold to the similarity matrix. When this threshold is high, the graph will be sparse, and we will find few clusters, which will tend to be of small size and high quality. When the threshold is lower, we will find more, larger clusters. Note that PG_{CLIQUE} and PG_{CC} will generally require different threshold settings; a sparse graph that contains a number of connected components may be too sparse to contain any sizeable cliques. We tune the threshold experimentally so as to find a sufficient number of clusters of the approximate desired size; PG_{CLIQUE}, however, generally finds a small number of small clusters. Generally, we find that PG_{CLIQUE} performs better than PG_{CC} and so is the variant we use. We compare these variants experimentally in section 3.

4. Rank the clusters found, and choose which to output. Step (3) may find many clusters, but we may wish to output only a few. For example, the site's webmaster might want to see no more than a handful of clusters every week and decide which to turn into new index pages. Accordingly, all clusters found are rated and sorted by averaging the co-occurrence frequency between all pairs of documents in the cluster. We find that PG_{CLIQUE} tends to discover many similar clusters. Therefore, we have developed two ways of reducing the number of similar clusters in the final results.

- **Overlap reduction** proceeds through the ranked cluster list and removes any cluster that overlaps highly with a previously seen (i.e. better) cluster.

[9] Note that we place a maximum size on discovered clusters not only in the interest of performance but because large clusters are not useful output — we cannot, practically speaking, create a new web page containing hundreds of links.

- **Merging** walks the ranked list of clusters and, whenever a cluster has sufficient overlap with a previously seen cluster, merges the two and continues.

Both of these approaches require an overlap measure; we use the size of the intersection between two clusters divided by the size of their union. In both variants, the overlap threshold is a parameter that has been tuned experimentally. Note that, as connected components never overlap, neither reduction nor merging will have any effect on PG_{CC}. Reduction and merging each have advantages and disadvantages. In general, reduction preserves the coherence of clusters found (as it makes no changes to cluster contents) but may miss pages that could be clustered together. Merging, on the other hand, may combine all related pages together, but at the cost of reducing the overall coherence of the clusters. By default, we use reduction in order to preserve the coherence of our clusters. These two variants are compared to each other — as well as to the "raw" ranked list — in section 3.

However we process the ranked list of clusters, we return a bounded number of results, and apply a quality threshold – sometimes returning no results at all. The webmaster specifies this bound — e.g. "give me at most ten clusters above quality threshold X...".

5. For each cluster found, create a web page consisting of links to the documents in the cluster, and present it to the webmaster for evaluation. The PageGather algorithm finds candidate link sets and presents them to the webmaster as in figure 2. The webmaster is prompted to accept or reject the cluster, to name it, and remove any links she thinks inappropriate. Links are labeled with the titles of the target pages and ordered alphabetically by those titles. The webmaster is responsible for placing the new page at the site.

2.3 Time Complexity

What is the running time of the PageGather algorithm? We summarize here; a more complete analysis is found in [15]. Let L be the number of page views in the log and N the number of pages at the site. In step (1), we must group the page views by their originating machine. We do this by sorting page views by origin and time, which requires $O(LlogL)$ time. In step (2), we must process the log and create a matrix of size $O(N^2)$, which requires $O(L + N^2)$ time. Note that we implement our algorithm so that we can create the matrix from a collection of logs and use it repeatedly to generate candidate clusters. As we bound the size of discovered clusters, step (3) is polynomial in N.

2.4 Work Related to PageGather

PageGather addresses the problem of finding sets of items based on their access patterns. Previous work in both clustering and data mining can also be applied to this problem. As discussed above, cluster mining is a variation on traditional clustering. PageGather uses a graph-based clustering component which is specialized to cluster mining, but it is possible to adapt traditional clustering algorithms to this problem. In [15] and follow-up work, we compared PageGather's clustering component (both PG_{CLIQUE} and PG_{CC} variants) to two standard algorithms: K-means clustering [18] and hierarchical agglomerative clustering (HAC)[22]. There are literally hundreds of clustering algorithms and variations thereof. We chose K-means as it is particularly fast, and HAC as it is widely used. We found that PageGather's clustering component was faster than both HAC and K-means and found higher-quality clusters.

Frequent set algorithms are designed to find sets of similar items in large collections (see, for example, [1, 2, 19, 21]). We therefore compare PageGather to the standard *Apriori* algorithm for finding frequent sets (see [3]). In a traditional frequent set problem, the data is a collection of

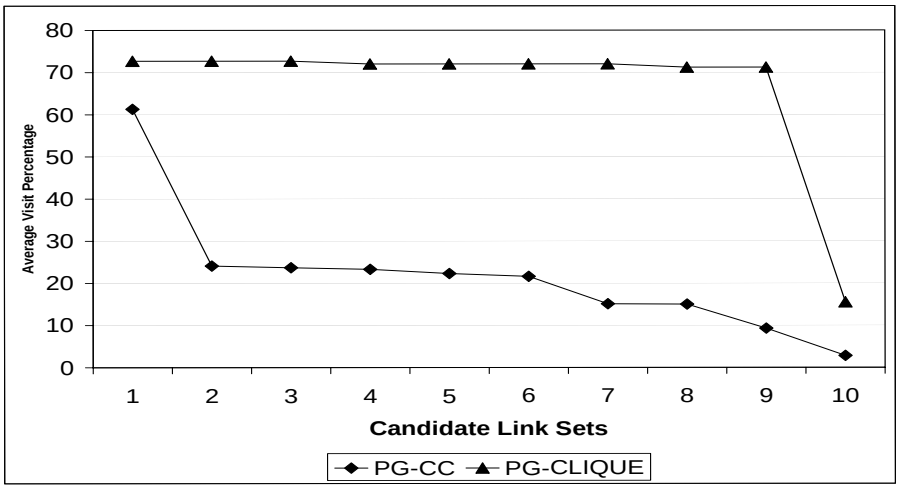


Figure 4: The performance of PG_{CLIQUE} and PG_{CC} using raw ranked-cluster output with no overlap reduction. Although PG_{CLIQUE} apparently performs much better, its clusters are all variations on the same basic set of pages.

market basket information. Each basket contains a set of items, and the goal is to find sets of items that appear together in many baskets. In our problem domain, a user visit corresponds to a market basket, and the set of pages visited by the user corresponds to the set of items in the basket. In section 3, we compare the performance of PageGather to that of Apriori on our test data.

3 Experimental Validation

In this section, we report on experiments designed to test the effectiveness of our approach. We first compare the performance of several variants of our algorithm. We then compare PageGather with the Apriori frequent set algorithm. Finally, we compare PageGather's candidate clusters with human-authored index pages.

Our experiments draw on data collected from the **Music Machines** web site,[10] a site devoted to information about many kinds of electronic musical instruments. The site contains approximately 2500 distinct documents, including HTML pages, plain text, images, and audio samples. Music machines receives approximately 10,000 hits per day from roughly 1200 distinct visitors. In our experiments, the training data is a collection of access logs for six months; the test data is a set of logs from a subsequent one-month period.[11]

We compare the algorithms in terms of the quality of the candidate index pages they produce. Measuring cluster quality is a notoriously difficult problem. To measure the quality of a cluster as an index page candidate, we need some measure of whether the cluster captures a set of pages that are viewed by users in the same visit. If so, then grouping them together on an index page would save the user the trouble of traversing the site to find them. As an approximate measure we ask: if a user visits any one page in the cluster, what percentage of pages in the cluster is she likely to visit overall? More formally, let V be the set of user visits to the site and V_c be the set of $v \in V$

[10] `http://machines.hyperreal.org`

[11] Data sets are publicly available from the authors.

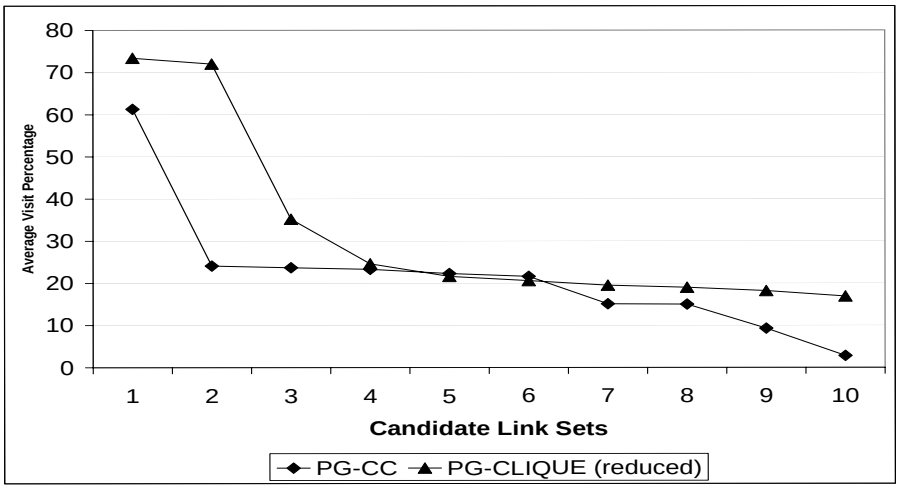


Figure 5: The performance of PG_{CLIQUE} and PG_{CC} using overlap reduction. PG_{CLIQUE} performs much better on the top three clusters and generally better overall.

that include at least one page from cluster c. For a particular visit $v \in V_c$, the number of pages visited in c is $v \cap c$. The average number of hits to a cluster, over all visits, is $\sum_{v \in V_c} \frac{v \cap c}{|V_c|}$. Finally, the average percentage is this average divided by the size of c: $\frac{\sum_{v \in V_c} \frac{v \cap c}{|V_c|}}{|c|}$. In each experiment, each algorithm outputs a ranked list of clusters. In all graphs, these clusters are sorted by average visit percentage for the sake of clarity.

As described in section 2.2, the two PageGather variants PG_{CLIQUE} and PG_{CC} differ in how clusters are found in the graph representation. PG_{CLIQUE} finds all maximal cliques; PG_{CC} finds all connected components in the graph. In our first experiment, we compare these two variants of the algorithm, shown in figure 4. Note that the clusters found by PG_{CLIQUE} apparently perform much better than those found by PG_{CC}. When we examine the clusters closely, however, we find the top clusters found by PG_{CLIQUE} to be very similar to each other — PG_{CLIQUE} tends to produce many slight variations on the same basic cluster. The reason for this is that there may exist a well-connected set of pages that does not, however, form a clique. Many subsets, however, are cliques, and the algorithm returns these many similar subsets as clusters. If we decrease the co-occurrence threshold, allowing more pages to be connected by edges in our graph representation, one such set of pages may become a clique. However, there will generally always be some borderline sets that exhibit this behavior. This observation was the motivation for introducing *overlap reduction* and *merging* as described in section 2.2. In figure 5, we compare PG_{CC} with PG_{CLIQUE} using overlap reduction. Note that PG_{CC} is unaffected by the reduction step, as connected components of a graph never overlap. We see that the two versions of the algorithm now perform comparably. However, PG_{CLIQUE} has a slight edge, and we will use it for our remaining comparisons.

Earlier, we discussed three ways of processing the ranked list of clusters found by PageGather's clustering component. We may simply return the raw ranked list, we may eliminate overlapping clusters, or we may merge similar clusters together. We have already discussed why the raw list is inadequate. In figure 6, we compare the performance of overlap reduction and merging applied to PG_{CLIQUE}. We might expect the merged clusters to show lower quality scores, as this operation brings together documents that were not associated by PageGather's clustering component, and

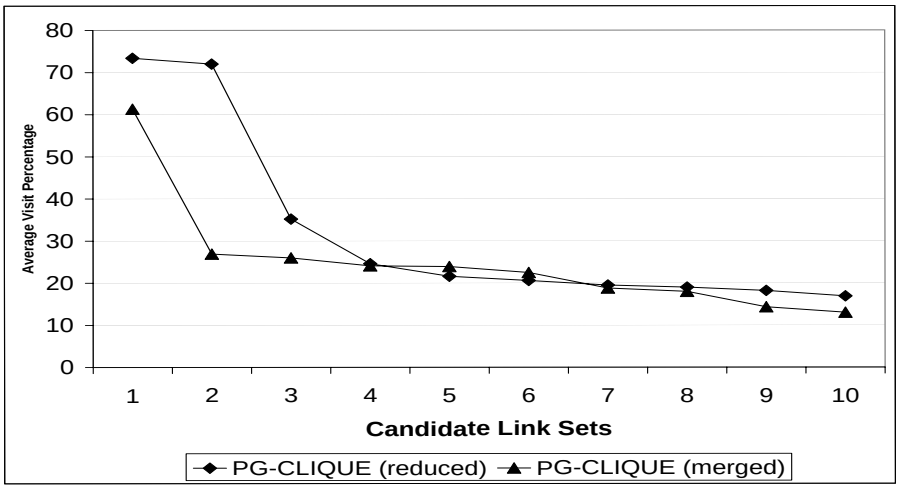


Figure 6: Comparing the performance of PG_{CLIQUE} with overlap reduction to PG_{CLIQUE} with cluster merging. Reduction performs better on the first three clusters, but the two variants are otherwise comparable.

this is apparent in the performance comparison. On the other hand, the best merged cluster is about three times larger (containing 19 links) than the best from the reduced set (7 links). The smaller cluster is a subset of the larger, which is arguably a more *complete* cluster; see the discussion in section 4.

In section 2.4, we discussed how the Apriori frequent set algorithm can be applied to the problem of index page synthesis. We apply Apriori to the collection of path data to find the most frequently occurring sets of pages; the most frequently occurring sets are our candidate clusters and are compared to the output of PG_{CLIQUE}. We now compare the performance of Apriori with PG_{CLIQUE} (see figure 7). Initially, we evaluated the raw output of Apriori. However, as with the raw version of PG_{CLIQUE}, the top clusters are all variations of each other. We therefore also show the results of adding a reduction step to Apriori as well; surprisingly, of the thousands of sets found by Apriori, only two prove to be sufficiently distinct. In either case, we observe that PG_{CLIQUE}'s clusters score much higher than Apriori's.

It is natural to ask how good these clusters really are, and how high an average visit percentage is high enough. To attempt to answer these questions we look to the "ideal" case: index pages created by a human webmaster. Music Machines contains many index pages of different kinds. Expecting all of the site's index pages to score significantly better than PageGather's output, we chose index pages pertaining to four representative topics: (1) instruments produced by a particular manufacturer (Roland); (2) documents pertaining to a particular instrument (the Roland Juno keyboard); (3) all instruments of a particular type (drum machines); and (4) all files of a particular type (audio samples). The set of outgoing links on each page (excluding standard navigation links) was treated as a "cluster" and evaluated on our test data. Figure 8 shows the comparison of these clusters to the output of PG_{CLIQUE} with overlap reduction. We believe there are two reasons why PG_{CLIQUE}'s clusters perform so much better than the existing human-authored index pages. First, we believe that our cluster mining approach finds genuine regularities in the access logs that carry over from training data to test data. PageGather is geared toward finding these regularities and is apparently successful. a high proportion of links when there are so many on the page.

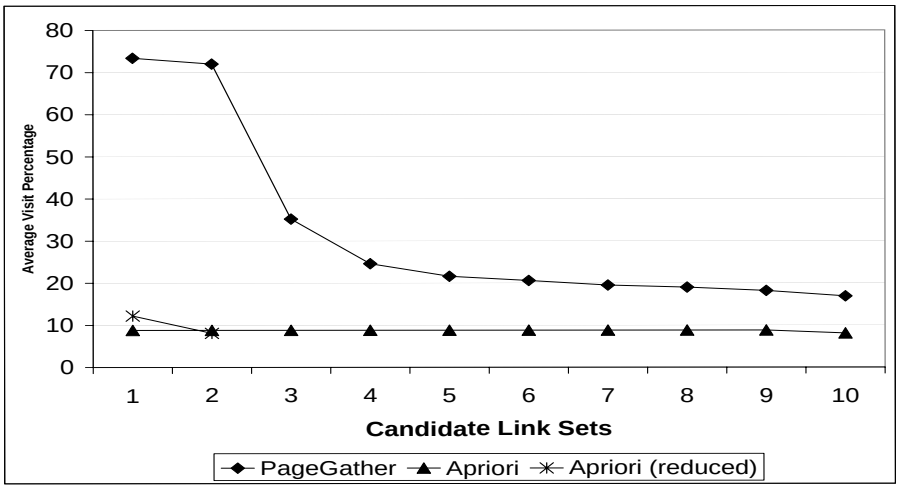


Figure 7: The performance of PG_{CLIQUE} with overlap reduction compared with the Apriori algorithm, both with and without overlap reduction. PG_{CLIQUE} performs significantly better than both variations. Note that Apriori finds only two distinct clusters when overlap reduction is applied.

Second, although creating index pages with high click-through rates is desirable, the webmaster of a site has other considerations in mind. For example, a page titled "drum machines" should be *complete* — that is, it should contain *all* drum machines at the site, not just those that often co-occur in the logs. Similarly, such a page should be *pure* — it should not contain links to guitars, whatever the statistical regularity. These considerations are both motivated by the expectations of human visitors coming to the site, and are not accounted for by our measure. Therefore, while high average visit percentages are promising, they do not tell the whole story.

4 Future Work

The limitations of our quality measure suggest a natural direction for future work. So far, we have focused on finding clusters of frequently co-occurring pages. While this approach can produce useful index pages, it does not address issues of purity or completeness. For example, when presented with a page titled "electric guitars," the typical user would expect the set of links provided to be *pure* — containing only links to guitars — and *complete* — containing links to all guitars at the site. Purity and completeness are analogous, respectively, to the criteria of precision and recall from information retrieval. IR systems are often evaluated in terms of their precision and recall with respect to a labeled data collection; human judges decide which objects match a particular query, and the system is rated on how closely its results accord with the human judges. Precision and recall may be a better metric for evaluating PageGather, but would require hand labeling of many examples — for each cluster found, we would have to judge what topic the cluster corresponds to and what set of pages actually belong in the cluster.

Instead, our goal is to extend our current approach to *automatically* identify topics and find pure and complete sets of relevant pages. We plan to use the candidate link sets generated by PageGather as a starting point, mapping them to the closest pure and complete topics. There are two ways to make the notion of a "topic" available to PageGather. First, if we have an extensional definition of each potential topic at the site as a set of links, then identifying the topic

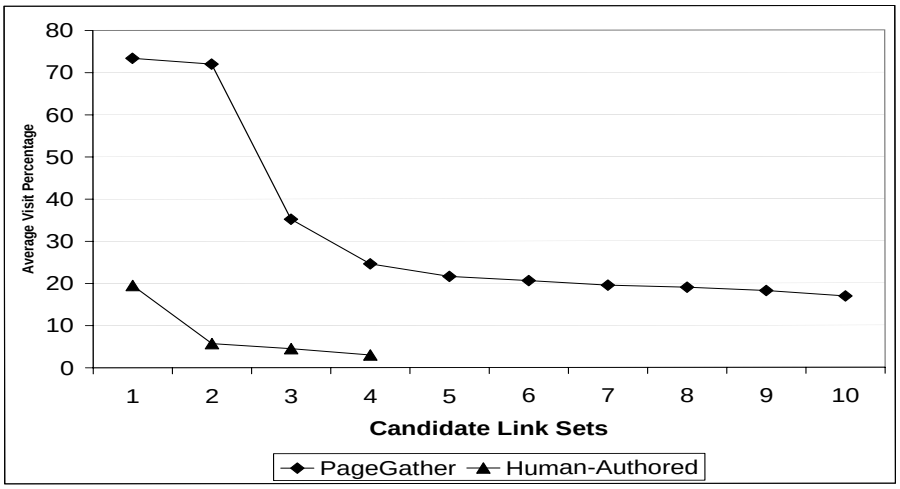


Figure 8: The performance of PG_{CLIQUE} using overlap reduction as compared to the performance of clusters based on human-authored index pages at the Music Machines web site. Clusters found by PG_{CLIQUE} perform significantly better than the existing index pages.

closest to a PageGather-generated link set is straightforward. Alternatively, if we have a predicate language for describing the different pages at the site (in XML or a-la-STRUDEL [6]), then we can apply concept learning techniques [12] to generate a description of the PageGather link set in that language. We would view the PageGather links as positive examples of the target concept, and links outside the set as negative examples, and apply a noise-tolerant symbolic learning technique (such as decision trees) to output a description of the topic most closely matching the PageGather link set.

This mapping from candidate link sets to topics is likely to decrease our statistical measure, but we realize that the measure is only a rough approximation of "true" index page quality. We plan to investigate alternative measures and also to carry out user studies (with both Web site visitors and webmasters) to assess the impact of the suggested adaptations on users in practice.

Index pages also have at least two common uses: as a summary of information on a particular topic (useful to a visitor wanting to get an overview of that topic) and as a directory of specific resources (useful to a visitor with a specific goal). These different uses have different requirements, which may conflict. For example, for the first usage, a few of the most important links may be sufficient — better to make sure all links are topical than to include something irrelevant. For the second, a complete listing is essential, even if some of the included links are only marginally relevant. We have focused on the first usage, but a good index should be able to support both.

Finally, our work thus far has focused on a single web site for convenience. We plan to test our approach on additional web sites, including our department's web site, in the near future.

5 Conclusion

The work reported in this paper is part of our ongoing research effort to develop adaptive web sites and increase their degree of automation. We list our main contributions below:

1. We motivated the notion of adaptive web sites and analyzed the design space for such sites, locating previous work in that space.

2. To demonstrate the feasibility of non-trivial adaptations, we presented a case study in the domain of synthesizing new index pages. We identified a key subproblem that is amenable to an automatic solution.

3. Next, we presented the fully-implemented PageGather algorithm for discovering candidate index page contents based on visitor access patterns extracted from web server logs.

4. We experimentally compared PageGather's output with the frequent sets discovered by the Apriori data mining algorithm and with human-authored index pages.

Finally, we identified the generation of *complete* and *pure* index pages as a key next step in the automation of index page synthesis. Index page synthesis itself is a step towards the long-term goal of *change in view*: adaptive sites that automatically suggest alternative organizations of their contents based on visitor access patterns.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of SIGMOD-93*, pages 207–216, 1993.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo. *Fast Discovery of Association Rules*, pages 307–328. MIT Press, Cambridge, MA, 1996.

[3] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th VLDB Conference*, 1994.

[4] E. André, W. Graf, J. Müller, H.-J. Profitlich, T. Rist, and W. Wahlster. AiA: Adaptive Communication Assistant for Effective Infobahn Access. Document, DFKI, Saarbrücken, 1996.

[5] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. Webwatcher: A learning apprentice for the world wide web. In *Working Notes of the AAAI Spring Symposium: Information Gathering from Heterogeneous, Distributed Environments*, pages 6–12, Stanford University, 1995. AAAI Press.

[6] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. System Demonstration - Strudel: A Web-site Management System. In *ACM SIGMOD Conference on Management of Data*, 1997.

[7] J. Fink, A Kobsa, and A. Nill. User-oriented Adaptivity and Adaptability in the AVANTI Project. In *Designing for the Web: Empirical Studies*, Microsoft Usability Group, Redmond (WA)., 1996.

[8] A. Fox, S. Gribble, Y. Chawathe, and E. Brewer. Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Perspectives. *IEEE Personal Communications*, 5(4):10–19, 1998.

[9] R. Khare and A. Rifkin. XML: A Door to Automated Web Applications. *IEEE Internet Computing*, 1(4):78–87, 1997.

[10] H. Lieberman. Letizia: An agent that assists web browsing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 924–929, 1995.

[11] S. Luke, L. Spector, D. Rager, and J. Hendler. Ontology-based web agents. In *Proc. First Intl. Conf. Autonomous Agents*, 1997.

[12] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[13] M. Perkowitz and O. Etzioni. Adaptive web sites: an AI challenge. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

[14] M. Perkowitz and O. Etzioni. Adaptive web sites: Automatically learning from user access patterns. In *Proceedings of the Sixth Int. WWW Conference*, Santa Clara, CA, 1997.

[15] M. Perkowitz and O. Etzioni. Adaptive Web Sites: Automatically Synthesizing Web Pages. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998.

[16] E. Rasmussen. Clustering algorithms. In W.B. Frakes and R. Baeza-Yates, editors, *Information Retrieval*, pages 419–442. Prentice Hall, Eaglewood Cliffs, N.J., 1992.

[17] T. Rist, E. André, and J. Müller. Adding Animated Presentation Agents to the Interface. In *Proceedings of the 1997 International Conference on Intelligent User Interfaces*, pages 79–86, Orlando, Florida, 1997.

[18] J. Rocchio. *Document Retrieval Systems — Optimization and Evaluation.* PhD thesis, Harvard University, 1966.

[19] A. Savasere, E. Omiecinski, and S. Navathe. An Efficient Algorithm for Mining Association Rules in Large Databases. In *Proceedings of the 21st VLDB Conference*, 1995.

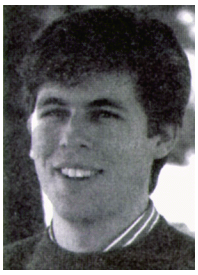
[20] R. Segal. *Data Mining as Massive Search.* PhD thesis, University of Washington, 1996. http://www.cs.washington.edu/homes/segal/brute.html.

[21] H. Toivonen. Sampling Large Databases for Association Rules. In *Proceedings of the 22nd VLDB Conference*, pages 134–145, 1996.

[22] E.M. Voorhees. Implementing agglomerative hierarchical clustering algorithms for use in document retrieval. *Information Processing & Management*, 22:465–476, 1986.

[23] A. Wexelblat and P. Maes. Footprints: History-rich web browsing. In *Proc. Conf. Computer-Assisted Information Retrieval (RIAO)*, pages 75–84, 1997.

[24] P. Willet. Recent trends in hierarchical document clustering: a critical review. *Information Processing and Management*, 24:577–97, 1988.



Oren Etzioni is an Associate Professor in the Department of Computer Science and Engineering at the University of Washington. He received his Ph.D. from Carnegie Mellon University in 1991. After joining the University of Washington he launched the Internet Softbot project. He received an NSF Young Investigator Award in 1993. His research interests include Software Agents, Web Navigation and Search Technology, and Human-Computer Interaction. See http://www.cs.washington.edu/homes/etzioni/.

Mike Perkowitz is a graduate student in computer science at the University of Washington. He received his bachelor's degree in cognitive science from Brown University in 1993. In 1994, he received a three-year NSF graduate fellowship. His research interests include adaptive web sites, machine learning, intelligent agents, and intelligent user interfaces. See http://www.cs.washington.edu/homes/map/.