Adaptive Web Sites: Automatically Synthesizing Web Pages

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

Content Areas: data mining, machine learning, applications, user interfaces

Abstract

The creation of a complex web site is a thorny problem in user interface design. In IJCAI '97, we challenged the AI community to address this problem by creating *adaptive web sites*: sites that automatically improve their organization and presentation by mining visitor access data collected in Web server logs. In this paper we introduce our own approach to this broad challenge. Specifically, we investigate the problem of *index page synthesis* — the automatic creation of pages that facilitate a visitor's navigation of a Web site.

First, we formalize this problem as a clustering problem and introduce a novel approach to clustering, which we call *cluster mining*: Instead of attempting to partition the entire data space into disjoint clusters, we search for a small number of cohesive (and possibly overlapping) clusters. Next, we present PageGather, a cluster mining algorithm that takes Web server logs as input and outputs the contents of candidate index pages. Finally, we show experimentally that PageGather is both faster (by a factor of three) and more effective than traditional clustering algorithms on this task. Our experiment relies on access logs collected over a month from an actual web site.

Adaptive Web Sites

Designing a rich web site so that it readily yields its information can be tricky. 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 (Perkowitz & Etzioni 1997a), we challenged the AI community to address this problem by creating **adaptive web sites:** *web sites that automatically improve their organization and presentation by learning from visitor access patterns.* In this paper we report on our own progress.

Sites may be adaptive in two basic ways. *Customization* is adapting the site's presentation to the needs of individual visitors, based on information about those individuals. In order to specialize itself to individual users, the site must maintain multiple copies of itself and gather quite a bit of information from users. Providing such information to the site can be time-consuming and may be an invasion of privacy. *Optimization* is improving the site's structure based on interactions with all visitors. Instead of making changes for each individual, the site learns from numerous past visitors to make the site easier to use for all, including those who have never used it before.

While previous work has focused on customizing web sites, we chose to investigate web site optimization through the automatic synthesis of index pages. In the next section, we discuss our general approach and present the *index page synthesis problem.* We then present our technique, which we call cluster mining, and its instantiation in the PageGather algorithm; PageGather solves the subproblem of automatically synthesizing the set of links that comprises an index page. Following, we present the results of experiments run using our implemented system. Finally, we discuss related work and future directions.

The Index Page Synthesis Problem

Our approach to adaptive web sites is motivated by four goals: (1) avoiding additional work for visitors (e.g. filling out questionnaires); (2) making the web site easier to use for everyone, not just specific individuals; (3) using web sites as they are, without relying on meta-information not currently available (e.g., XML annotations); and (4) protecting the site's original design from destructive changes. When creating a web site, a human meticulously designs the look and feel of the site, the structure of the information, and the kinds of interactions available. When making automatic changes to such a site, we wish to avoid damaging the site.

Our approach, therefore, is to apply only *nondestructive transformations*: changes to the site that leave ex-

*Copyright (c) 1998, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

isting structure intact. We may add links but not remove them, create pages but not destroy them, add new structures but not scramble existing ones. Such transformations may include highlighting links, *promoting* links to the front page, cross-linking related pages, and creating new pages of related links. Based on the access log, the site decides when and where to perform these transformations. In (Perkowitz & Etzioni 1997b), we sketched several such transformations. In this paper, we focus on a single, novel transformation: the creation of new index pages – pages consisting of links to pages at the site relating to a particular topic. We illustrate this transformation with a simple real-world example.

The *Music Machines* web site contains information about various kinds of electronic musical equipment (see `http://www.hyperreal.org/music/machines/`). The information is primarily grouped by manufacturer. For each manufacturer, there may be multiple entries for the different instrument models available — keyboards, electric guitars, amplifiers, etc. For each model, there may be pictures, reviews, user manuals, and audio samples of how the instrument sounds. We might notice that, when exploring the site, visitors comparing electric guitars from many different manufacturers tend to download the audio samples of each guitar. A comprehensive page of "Electric Guitar Audio Samples" would facilitate this comparison; this is the kind of page we would like our system to generate automatically.

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. 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. An access log is a document containing one entry for each request answered by 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.

In synthesizing a new index page, we must solve several subproblems.

1. **What are the contents of the index page?**
2. How are the contents ordered?
3. What is the title of the page?
4. How are the hyperlinks on the page labeled?
5. Is the page consistent with the site's overall graphical style?
6. Is it appropriate to add the page 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 some subproblems, particularly the last one, are quite difficult and may be solved in collaboration with the human webmaster.

Rather than attempting to understand the content of every page at our site and to figure out which are related, our approach is based on the analysis of each "visit". We define a *visit* to be an ordered sequence of pages accessed by a single visitor in a single session. We make 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. To overcome noise, we accumulate statistics over many visits by numerous users and search for overall trends.

The PageGather Algorithm

In this section, we present a novel approach to clustering, called *cluster mining*, that was motivated by our task; in addition, we introduce *PageGather* — the first index page-contents synthesis algorithm. Given a large access log, our task is to find collections of pages that tend to co-occur in visits. Clustering (see (Voorhees 1986; Rasmussen 1992; Willet 1988)) is a natural technique 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 for 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 (Segal 1996). 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 four 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 cliques (or connected components) in the graph.
4. For each cluster found, create a web page consisting of links to the documents in the cluster.

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 *hits*, or requests made to the web server. 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 be able to view the log as containing a number of discrete visits. We first assume that each originating machine corresponds to a single visitor.[1] A series of hits in a day's log from one visitor, ordered by their time-stamps, corresponds to a single session for that visitor. Furthermore, we make sure that we log every access to every page by disabling caching — every page contains a header saying that it expires immediately; browsers will therefore load a new copy every time a user views that page.

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. Essentially, we create a matrix corresponding to existing connections and subtract it from the similarity matrix created from the log.

[1]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 with visitor-tracking software such as WebThreads.

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. We treat all remaining arcs as being of equivalent strength.[2] By creating a sparse graph, we can use graph algorithms to find clusters, which turn out to be faster than traditional clustering methods.

3. Create the graph corresponding to the matrix, and find 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. In this graph, a cluster corresponds to a set of nodes whose members are directly connected with arcs. A clique – a subgraph in which every pair of nodes has an edge between them – is a cluster in which every pair of pages co-occurs often. A connected component – a subgraph in which every pair of nodes has a path of edges between them – is a cluster in which every node is similar to at least one other node in the cluster. While cliques form more coherent clusters, connected components are larger, faster to compute, and easier to find.

4. For each cluster found, create a web page consisting of links to the documents in the cluster. Our research so far has focused on generating the content of index pages — the set of links — rather than the other aspects of the problem. We therefore use simple solutions which will be improved in future work. Page titles are generated by the human webmaster. Pages are linked into the site at one particular location as part of a "helpful tour guide" metaphor. Links on pages are ordered alphabetically by their titles. Page layouts are based on a template defined for all pages at the site.

What is the running time of the PageGather algorithm? Let L be the number of hits in the log, N the number of pages at the site, E be the number of edges in our graph, and C be the largest cluster we wish to find.[3] In step (1), we must group the hits by their originating machine. We do this by sorting hits by origin and time, so step (1) requires $O(L log L)$ time. In step (2), we must create a matrix of size $O(N^2)$ and examine each cell in the matrix. Step (2) is therefore $O(N^2)$. In step (3) we may look for either cliques or connected components. In general, finding maximal cliques in a graph is NP-complete. However, since we search for cliques whose size is bounded by a constant C, this step is a polynomial of order C. Finding a connected component requires a depth-first search in the graph. The complexity of depth-first search is $O(E)$, where E

[2]While we consider all arcs equivalent for the purpose of finding clusters, we use the arc strengths for ranking clusters later.

[3]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.

may be $O(N^2)$ in the worst case but is less in our sparse graphs. Note that the asymptotically most expensive part of PageGather is the creation of the similarity matrix; even a version that did not use our cluster mining technique would have at least this cost.

Experimental Validation

In this Section, we report on experiments designed to test the effectiveness of our approach by comparing it with traditional clustering methods; we also experiment with several PageGather variants to assess the impact of key facets of the algorithm on its performance. Our experiments draw on data collected from `http://www.hyperreal.org/music/machines/`. The site is composed of about 2500 distinct documents and receives approximately 10,000 hits per day from 1200 different visitors. We have been accumulating access logs for over a year.

In our experiment, each algorithm chooses a small number k of high-quality clusters. We then compare the running time and performance of each algorithm's top k clusters. We modified traditional clustering algorithms to return a small number of clusters (not necessarily a partition of the space), converting them into cluster mining algorithms as needed for our task. In all cases, clusters are ranked by their *average pairwise similarity* — calculated by averaging the similarity between all pairs of documents in the cluster — and the top k clusters are chosen. In our experiment, the training data is a collection of access logs for an entire month; each algorithm creates ten clusters based on these logs. The test data is a set of logs from a subsequent ten-day period.

There are literally hundreds of clustering algorithms and variations thereof. To compare PageGather with traditional methods, we picked two widely used document clustering algorithms: hierarchical agglomerative clustering (HAC)(Voorhees 1986), and K-Means clustering (Rocchio 1966). HAC is probably the most popular document clustering algorithm, but it proved to be quite slow. Subsequently, we chose K-Means because it is a linear time algorithm known for its speed. Of course, additional experiments are required to compare PageGather with other clustering algorithms before general conclusions can be drawn. We also compared two versions of PageGather — one using connected components and one using cliques.

Our first experiment compared the speed of the different algorithms as shown in Figure 1. Because all algorithms share the cost of creating the similarity matrix, we compare only the clustering portion of PageGather. We implemented two of HAC's many variations: complete link clustering, in which the distance between two clusters is the distance between their *farthest* points, and single link, in which the distance between two clusters is the distance between their *nearest* points. To generate k clusters, we had HAC iterate until it created a small number of clusters (about $2k$)

	Run time (min:sec)	Average cluster size
PageGather		
Connected Component	1:05	14.9
Clique	1:12	7.0
K-Means	48:38	231.9
K-Means (modified)	3:35	30.0
HAC	48+ hours	—

Figure 1: Running time and average cluster size of PageGather and standard clustering algorithms.

and then chose the best k by average pairwise similarity. We found HAC to be very slow compared to our algorithm; the algorithm ran for over 48 hours without completing (three orders of magnitude slower than PageGather).[4] HAC algorithms, in this domain, are asymptotically (and practically!) quite slow. We therefore decided to investigate a faster clustering algorithm.

In K-Means clustering, a target number of clusters (the "K") is chosen. The clusters are seeded with randomly chosen documents and then each document is placed in the closest cluster. The process is restarted by seeding new clusters with the centroids of the old ones. This process is iterated a set number of times, converging on better clusters. To generate k clusters for our experiment, we set K-Means to generate $2k$ clusters and then chose the top k by average pairwise similarity. As with the HAC algorithm, the costly similarity measure in our domain makes K-Means quite slow — the algorithm takes approximately 48 minutes in our experiment. However, just as we make our clique algorithm tractable by limiting the size of clusters to 30, we limited cluster size in the K-Means algorithm to 30 as well. We compared this modified version to PageGather and still found it to be slower by a factor of three.

We also compared PageGather using cliques in step 3 with PageGather using connected components. We found relatively little difference in speed. Finding connected components in a graph is a very fast operation. Although finding maximal cliques is generally intractable, it too is extremely fast when we limit the maximum clique size and the graph is sparse; in our experiment, the graph contained approximately 2500 nodes and only 13,000 edges after applying the threshold.

Next, we compared the algorithms in terms of the quality of candidate index pages they produce. Measuring cluster quality is a notoriously difficult problem.

[4]We chose simple implementations of two popular HAC algorithms; more efficient algorithms exist which may run faster. Our domain, however, differs from more standard domains such as document clustering. Documents can be represented as word vectors; word vectors can be compared or even averaged to create new centroid vectors. We, however, have no vectors, but only a similarity matrix. This limitation makes certain kinds of algorithmic optimizations impossible.

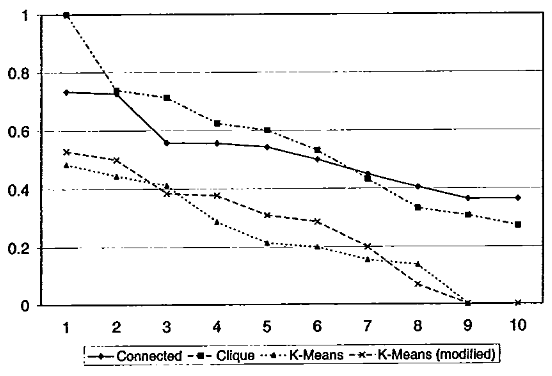

Figure 2: Predictive performance of PageGather (using both connected components and cliques) and K-Means clustering on access data from a web site.

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 will save the user the trouble of traversing the site to find them. Thus, as an approximate measure we ask: if a user visits any one page in the cluster, how likely is she to visit more? More formally, if $n(i)$ is the number of pages in cluster i that a person examines during one visit to the site, then the quality $Q(i)$ of a cluster is $P(n(i) \geq 2 | n(i) \geq 1)$. The higher this conditional probability, the more valuable the candidate index page represented by the cluster. Of course, this measure is imperfect for a several reasons. Most seriously, the measure is biased toward larger clusters. We do not penalize a cluster for being overly inclusive or measure how much of a cluster a user visits. Figure 1 shows average cluster size for each algorithm. Note that the K-Means algorithm consistently generates larger clusters than PageGather. Thus, the bias in the metric works against PageGather. Although the K-means algorithm found, on average, larger clusters, it did not find significantly *fewer* than PageGather. PageGather, therefore had no advantage in being able to select the best from among a larger number of clusters.

Figure 2 shows the performance of four different algorithms. For each algorithm, we show the quality Q of its chosen ten best clusters. We graph each algorithm's top ten ordered by performance. As both K-Means variants produce significantly larger clusters than PageGather, we might expect better performance from K-means, but we see immediately that the PageGather variants perform better than either of the K-Means variants. PageGather is both faster and more accurate.

Comparing the two variants of PageGather, we find that neither is obviously superior. The connected component version, however, creates clusters that are, on average, about twice as large as those found with the clique approach. The best clique clusters are also somewhat better. The clique approach, we conclude, finds smaller, more coherent clusters.

To test our hypothesis that creating overlapping clusters is beneficial in this domain, we created a variant of PageGather that creates mutually exclusive clusters by forcing each page into exactly one cluster. We compared the performance of PageGather with and without overlapping. For readability, we omit the non-overlapping version of the clique algorithm in Figure 2, but the performance of the non-overlapping version drops substantially, though it is still better than either K-Means variant. The removal of overlapping did not substantially change the performance of PageGather using connected components.

Having applied this performance measure, we might also ask: qualitatively, how good are the clusters we find? Do they seem to correspond to concepts people would understand? We have not yet performed the kind of user testing necessary to answer this question in a fully satisfying way. However, a number of the clusters output by PageGather are convincingly coherent. For example, PageGather created one cluster containing most of the audio samples found at the site. Other clusters grouped similar keyboards from various manufacturers and downloadable software from across the site. Most clusters appear highly coherent — most of the less useful ones are composed entirely of pages from a single section of the site and hence do not represent an interesting "discovery". However, if PageGather makes one or two interesting discoveries out of every ten suggestions to the webmaster, it may still be quite useful.

Related Work

Automatic customization has been investigated in a variety of guises. (see (Joachims, Freitag, & Mitchell 1997; Fink, Kobsa, & Nill 1996)). These approaches tend to share certain characteristics. First, the web site or agent dynamically presents information – typically an enhancement of the currently viewed web page – to the visitor. Second, that information is customized to that visitor or a class of visitors based on some model the system has of that individual or class. Third, that model is based on information gleaned from the visitor and on the actions of previous visitors. In contrast, our approach makes offline changes to the entire site, makes those changes visible to all visitors, and need not request (or gather) information from a particular visitor in order to help her.

(Perkowitz & Etzioni 1997b) presented the web site optimization problem in terms of *transformations* to the web site which improve its structure. We sketched several kinds of transformations and discussed when to automatically apply them. Index page synthesis may be viewed as a novel transformation of this sort. Whereas that work broadly described a range of possible transformations, we have now implemented and tested one in depth.

Footprints (Wexelblat & Maes 1997) also takes an optimizing approach. Their motivating metaphor is that of travelers creating footpaths in the grass over time. Visitors to a web site leave their "footprints" behind; 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 is similar to our approach in that it makes changes to the site, based on user interactions, that are available to all visitors. However, Footprints provides essentially *localized* information; the user sees only how often links between adjacent pages are traveled. We allow the site to create new pages that may link together pages from across the site.

Future Work and Conclusions

This work is part of an our ongoing research effort; it is both an example of our approach and a step toward our long-term goal of creating adaptive web sites. We list our main contributions toward this goal below.

1. We formulated the novel task of automatic index page synthesis and decomposed the task into five subproblems, defining an agenda for future work on adaptive web sites. We focused on the subproblem of page-contents synthesis and formalized it as a clustering problem. In the process, we introduced key ideas including the distinction between optimization and customization, the visit-coherence assumption, and the principle of "nondestructive transformations".
2. We introduced PageGather, the first page-contents synthesis algorithm, and demonstrated its feasibility; PageGather takes Web site access logs as input and appears to produce coherent page contents as output. This is a proof of concept; in future work we will investigate its generality and seek to extend it to cover the full index page synthesis problem.
3. PageGather is based on *cluster mining*, a novel approach to clustering that, instead of partitioning the entire space into a set of mutually exclusive clusters, attempts to efficiently identify a small set of maximally coherent (and possibly overlapping) clusters. We demonstrated the benefits of cluster mining over traditional clustering experimentally in our domain. We believe that cluster mining will also prove beneficial in other domains, but this is a topic for future work.

Although the PageGather algorithm finds promising clusters in access logs, it is far from a complete solution to index page synthesis. Cluster mining is good for finding clumps of related pages — for example, several audio samples of guitars — but it has trouble finding the *complete* set of pages on a topic — e.g., *all* the guitar samples at the site. Yet visitors expect complete listings — a page titled "Audio Samples" had better list all of them. One way to create complete clusters is to use cluster mining to generate the original clusters, but then use the elements of the cluster as training data for learning a simple, general rule that defines membership in the cluster. The index page generated will contain links to *all* pages that match the rule.

PageGather could also potentially *flatten* the site's structure by grouping pages from across the web site onto a single page; important structural and hierarchical information may be lost, or fundamentally different kinds of pages might be grouped together. If we have *meta-information* about how the site is structured — for example, a simple ontology of the types of pages available — we should be able to find more homogeneous clusters. The use of meta-information to customize or optimize web sites has been explored in a number of projects (see, for example, (Khare & Rifkin 1997; Fernandez *et al.* 1997; Luke *et al.* 1997) and Apple's Meta-Content Format).

References

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

Fink, J., Kobsa, A., and Nill, A. 1996. User-oriented Adaptivity and Adaptability in the AVANTI Project. In *Designing for the Web: Empirical Studies.*

Joachims, T., Freitag, D., and Mitchell, T. 1997. Webwatcher: A tour guide for the world wide web. In *Proc. 15th Int. Joint Conf. AI*, 770–775.

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

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

Perkowitz, M., and Etzioni, O. 1997a. Adaptive web sites: an AI challenge. In *Proc. 15th Int. Joint Conf. AI.*

Perkowitz, M., and Etzioni, O. 1997b. Adaptive web sites: Automatically learning from user access patterns. In *Proceedings of the Sixth Int. WWW Conference.*

Rasmussen, E. 1992. Clustering algorithms. In Frakes, W., and Baeza-Yates, R., eds., *Information Retrieval.* Prentice Hall, Eaglewood Cliffs, N.J. 419–442.

Rocchio, J. 1966. *Document Retrieval Systems — Optimization and Evaluation.* Ph.D. Dissertation, Harvard University.

Segal, R. 1996. *Data Mining as Massive Search.* Ph.D. Dissertation, University of Washington. http://www.cs.washington.edu/homes/segal/brute.html.

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

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

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