Artificial Intelligence 118 (2000) 245–275

Artificial Intelligence

www.elsevier.com/locate/artint

Towards adaptive Web sites: Conceptual framework and case study

Mike Perkowitz[*], Oren Etzioni[1]

Department of Computer Science and Engineering, Box 352350, University of Washington, Seattle, WA 98195, USA

Received 1 February 1999; received in revised form 28 July 1999

Abstract

Today's Web sites are intricate but not intelligent; while Web navigation is dynamic and idiosyncratic, all too often Web sites are fossils cast in HTML. In response, this paper investigates *adaptive Web sites*: *sites that automatically improve their organization and presentation by learning from visitor access patterns*. Adaptive Web sites mine the data buried in Web server logs to produce more easily navigable Web sites.

To demonstrate the feasibility of adaptive Web sites, the paper considers the problem of index page synthesis and sketches a solution that relies on novel clustering and conceptual clustering techniques. Our preliminary experiments show that high-quality candidate index pages can be generated automatically, and that our techniques outperform existing methods (including the Apriori algorithm, K-means clustering, hierarchical agglomerative clustering, and COBWEB) in this domain. © 2000 Published by Elsevier Science B.V. All rights reserved.

Keywords: Adaptive Web sites; Conceptual clustering; Data mining

1. Introduction and motivation

Designing a complex Web site so that it readily yields its information is tricky. 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 use, but may be used in unanticipated ways in practice; the designer's *a priori* expectations may be violated. All too often Web sites are fossils cast in HTML.

[*] Corresponding author. Email: map@cs.washington.edu.
[1] Email: etzioni@cs.washington.edu.

In [25], we challenged the AI community to address this problem by creating *adaptive Web sites*: *sites that automatically improve their organization and presentation by learning from visitor access patterns.* Many AI advances, both practical and theoretical, have come about in response to such challenges. The quest to build a chess-playing computer, for example, has led to many advances in search techniques (e.g., [4]). Similarly, the autonomous land vehicle project at CMU [37] resulted not only in a highway-cruising vehicle but also in breakthroughs in vision, robotics, and neural networks. We believe that the adaptive Web sites challenge will both drive AI advances and yield valuable technology.

While adaptive Web sites are potentially valuable, their feasibility is unclear *a priori*: can nontrivial 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 focus on a case study—the problem of *index page synthesis*. An *index page* is a page consisting of links to a set of pages that cover a particular topic (e.g., electric guitars) at a site; index pages are similar to the "hub pages" described in [17]. Index page synthesis is the problem of automatically generating index pages to facilitate efficient navigation of a site, or to offer a novel "view" of the site. While most previous work has focused on creating *customized* versions of a site for individual visitors, our goal is to *transform* the site into a better one; many visitors to the site should benefit from these transformations. Index page synthesis is a first step in this direction; new index pages represent new views of the site that may accord with the way visitors view the information available.

Our basic approach is to analyze the Web site's access logs to find groups of pages that often occur together in user visits; we assume these groups of pages represent coherent topics in users' minds. We desire a reliable technique for finding such sets of pages and turning them into index pages. Previous work in *concept learning* and *clustering* is relevant to this endeavor. The goal of a concept learning algorithm (see [22]) is to find a conceptual description of a class of objects from a data collection based on examples of the class (positive examples) and examples of objects not in the class (negative examples). Essentially, the algorithm must determine what the objects have in common that distinguishes them from other objects in the collection. However, concept learning algorithms are *supervised* learning algorithms: they require that their inputs be classified in advance. In our domain, all we have is a collection of objects (Web pages) and some data about usage patterns. Clustering algorithms, on the other hand, are *unsupervised*: they take a collection of objects as their input and produce a *partition* of the collection—a classification of each and every object into exactly one class, or "cluster". However, clustering has its own drawbacks for our domain: we do not want a partition of all the Web pages, merely to find small sets of pages that belong together. Furthermore, in our domain, pages may be placed into more than one cluster.

Accordingly, we present in this paper new *cluster mining* algorithms whose design was motivated by our problem domain. A cluster mining algorithm is an unsupervised algorithm for efficiently identifying a small set of high-quality (and possibly overlapping) clusters with limited coverage. Although the development of cluster mining approaches was motivated by our problem domain, we believe these approaches have broader applications in learning and data mining.

As mentioned above, we wish to find clusters of objects that share some common topic from which we can create new index pages; these topics should be intuitively understandable to users, and every page in the cluster should match the topic. Statistical approaches, however, often produce clusters with no intuitive common topic, and give no guarantee that all objects in the cluster will accord with the topic. We therefore also explore *conceptual* cluster mining, in which all discovered clusters have intuitive descriptions that can be expressed to human users.

The remainder of this paper is organized as follows. First, we analyze the problem space of adaptive Web sites, present previous work in this area, and discuss our approach. In Section 3, we present PageGather, our statistical cluster mining algorithm, and experimentally compare PageGather to classical clustering and data mining algorithms. In Section 4, we describe IndexFinder, our conceptual cluster mining algorithm, and experimentally compare IndexFinder with both PageGather and a leading conceptual clustering algorithm. We conclude with a summary of our contributions and a discussion of future work.

2. Overview and our approach

We now discuss the problem space of adaptive Web sites and present examples of previous work in this area. With this background in place, we then discuss our own general approach.

2.1. Adaptivity as search

We view the automatic improvement of a Web site as search in the space of possible Web sites. Different approaches to creating adaptive Web sites correspond to different ways of searching the space. In this section, we discuss the search space, quality measure, initial state, goal state, and state transitions. We then discuss related work in terms of the distinctions presented.

2.1.1. Definition of a state

A state in our search is a Web site. We view a Web site as a function from

(1) a user model,
(2) the sequence of pages viewed during the current visit to the site, and
(3) the sequence of links clicked on during the current visit, to the Web page shown to the user.

A static Web site is a special case where the page shown is determined only by the most recent page in the sequence and the last link clicked on. In this case, our site representation is equivalent to a graph in which nodes represent pages and edges represent hyperlinks.

Adaptive sites, however, may use *customization* or *transformation*. Customization is modifying a Web site to suit an individual user; the page returned by the site depends on the user model. A site may also perform *online* customization—customizing the pages served to the user as she browses. In this case, the site takes into account not only the user model, but also the pages just visited in choosing the next page to show the 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.

2.1.2. The quality measure

The quality of a Web site is a function of the following variables:

- *Recall*: how often users find what they are looking for.
- *Effort*: how much work users exert to find what they are looking for measured in, for example, the number of links users have to make and the amount of time they spend reading link text and scrolling through pages.

A site that is difficult to use will require more time and effort on the part of the user. For example, suppose a user is trying to determine the price of Ford’s cheapest truck. One site simply has a page titled “Trucks” containing a link to every known truck model by every major manufacturer, in alphabetical order by model name. To answer her question, the user would have to scroll and find the links to all the Ford models, and then click on every such link to determine the cheapest price. In contrast, another site has a page for each manufacturer; the “Ford” page has a list of Ford trucks, sorted by price. She need only look at the first entry on this list to answer her question. Clearly, the second site is better organized to support this user’s task. Our long term goal is to develop methods to automatically transform a site from one to the other.

Of course, an adaptive Web site does not have direct access to the true quality measure. Instead, we must approximate it with the data we have available. Available data comes in two basic forms: the content of the Web site itself, and the access patterns of visitors to the site. A *content-based* adaptive approach attempts to improve the site based on what the pages say and what they are about. For example, one approach would be to analyze the text of Web pages at the site, and add links between pages that have similar texts. Presumably, such adaptations would make the site easier to use and thus improve the “true” quality. An *access-based* approach uses the way past visitors have interacted with the site to guide how information is structured. A simple example would be to find the most-accessed pages at the site and add links to them on the front page. Such an adaptation would reduce the effort of searching the site for many visitors. Content-based and access-based approaches are naturally complementary, and could be combined in a single adaptive site.

2.1.3. The initial and goal states

The initial state is typically an existing Web site. The goal state maximizes the quality measure. Web site design embodies complex tradeoffs: between the needs of different user groups, between ease of use and ease of maintenance, between the desire to provide comprehensive information and the desire to avoid clutter, etc. For example, adding links increases navigational choices for the user, but can also increase cognitive effort required to decide where to go next. In practice, Finding an “optimal” Web site is unrealistic. Most adaptive methods tend to focus on hill climbing from the initial state to another state that is likely to be better.

2.1.4. Defining state transitions

A state in the search space is a Web site; a state transition, therefore, is an alteration that creates a new, slightly different site. Many kinds of state transitions are possible. Local alterations include adding or removing links, rearranging links on a page, and highlighting links. Larger-scale transitions may create or destroy whole pages or reorganize an entire section of the site. Other state transitions make changes that are relevant only to individual visitors.

When evaluating the usefulness of a site transformation, we examine two factors:

- *Benefit*: how much we improve the site for those who use it.
- *Impact*: how many users actually benefit.

For example, one transformation might provide a small benefit to a large number of users while another only impacts a few users but helps them enormously; which is more appropriate depends upon the Web site.

2.2. Previous work

Many different kinds of adaptive Web sites have been explored recently. We discuss common approaches and provide examples. It is quite common for Web sites to allow users to customize the site for themselves. Most "portal" sites, for example, allow manual customizations such as lists of favorite links, stock quotes of interest, and local weather reports. A site might also, for example, keep track of pages that have changed since a user's 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). Path prediction may be done online, by predicting the user's goal based on her path so far, or it may be done offline, statically computed based on user models. The WebWatcher[2] [15] learns to predict what links users will follow on a particular page as a function of a model of their 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. Upon entering a site, visitors are asked, in broad terms, what they are looking for. Before they depart, they are asked if they have found what they wanted. WebWatcher uses 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 individual customization and pure transformation.

A site may use both the user's path and her model to guess what pages she will be interested in seeing. The AVANTI project[3] [10] 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

[2] http://www.cs.cmu.edu/∼webwatcher/.
[3] http://zeus.gmd.de/projects/avanti.html.

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. AVANTI also tries to infer the user's interests from her path through the site; a visitor to a museum site who looks at a number of paintings will tend to be shown paintings rather than sculptures.

Yan et al. [42] describe an automatic customization approach. They perform clustering over a Web site's access logs in order to identify categories of users; visitors to the site who visit similar pages presumably have similar interests. These categories may be used to classify new visitors to the site, in order to customize the site as they browse. For example, links that are often followed by other members of the category may be highlighted.

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. Ringo [36] (later known as Firefly) 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; Ringo then compares this profile with those of other members to make new recommendations. The GroupLens project [13,31] also uses collaborative filtering to recommend movies and news articles. Recent work has explored hybrid systems combining collaborative filtering, content-based methods, and personal agents. Balabanovic and Shoham's Fab system [6] is another hybrid system which marries collaborative filtering with content-based methods for recommending Web pages. Users provide ratings of pages, and these ratings are used to tune a user profile and determine the user's nearest neighbors: other users with similar tastes. Fab also maintains *collection agents* whose profiles represent topics of potential interest to a group of users.

Footprints [40] takes a transformation approach. Their motivating metaphor is that of travelers creating footpaths in the grass over time; in the Web site domain, visitors 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. However, Footprints provides *local* information; the user sees only how often links between adjacent pages are traveled. Fundamental re-organization is not supported.

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 [16], Apple's Meta-Content Format, and other projects [9,19]). One example of this approach is the STRUDEL Web site management system [9] 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 [5,32] 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 customization by the Web server as well. Letizia [18] is a personal agent that explores the Web ahead of the user (investigating links off of the current page) and uses a user model to recommend pages it thinks the user will want to visit next. Letizia learns a model of its user by observing her behavior. Syskill and Webert [24] is a similar system that allows a user to rate Web pages and constructs a profile for each of the user's distinct interests.

2.3. 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. 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) *Minimize additional work for the webmaster.* Although human-authored meta-information (e.g., XML annotations) may facilitate site adaptivity, we should weigh the benefits of this additional information against the cost of authoring it.
(4) *Protect the site's original design from destructive changes.* 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.
(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. Finally, we restrict ourselves to generating nondestructive adaptations and presenting them to the human webmaster—any nontrivial changes to the Web site remain under webmaster control.

The main source of information we rely on is the site's Web server log, which records the pages viewed by each visitor to the site. We rely on the *visit-coherence assumption*: the pages that a user visits during one session at the site One visitor may be interested in electric guitars, and view several pages on that topic; another may be looking for inexpensive keyboards and view all the pertinent pages. 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 [26]. 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 [25], we found a significant number of visitors used these automatic shortcuts when deployed on our experimental Web site.

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,[4] which has been our primary testbed; it is maintained by one of the authors, and we have full access to all pages 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.

[4] http://machines.hyperreal.org.

3. The index page synthesis case study

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.

3.1. 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 contains one entry for each page requested of the Web server (see Fig. 1). Each request lists, among other details, 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.

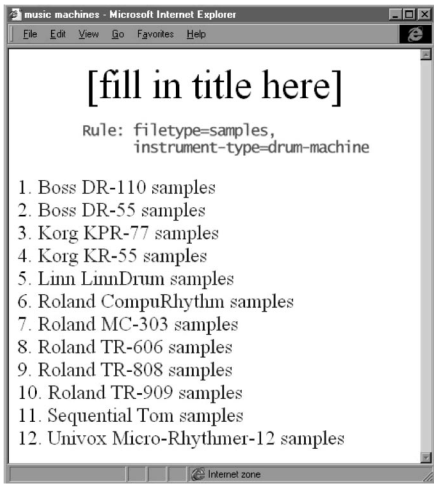
The problem of synthesizing a new index page can be decomposed into several subproblems.

(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?

We solve the first four subproblems automatically. The second and third can be solved trivially; in this paper we focus on the first and fourth. The fifth step is left to the judgment of the webmaster. Our implemented system is capable of producing collections of links like those shown in Fig. 2(a). Hyperlinks are labeled with the title of the target page and ordered alphabetically. The webmaster decides whether to add the page to the site and where to put

```
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
```

Fig. 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).

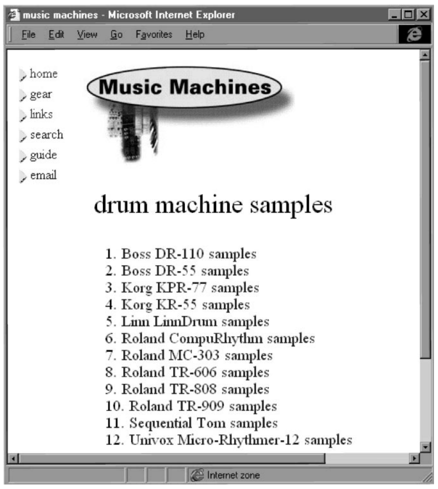

music machines - Microsoft Internet Explorer
File Edit View Go Favorites Help
Internet zone

[fill in title here]

Rule: filetype=samples,
      instrument-type=drum-machine

1. Boss DR-110 samples
2. Boss DR-55 samples
3. Korg KPR-77 samples
4. Korg KR-55 samples
5. Linn LinnDrum samples
6. Roland CompuRhythm samples
7. Roland MC-303 samples
8. Roland TR-606 samples
9. Roland TR-808 samples
10. Roland TR-909 samples
11. Sequential Tom samples
12. Univox Micro-Rhythmer-12 samples

(a)


music machines - Microsoft Internet Explorer
File Edit View Go Favorites Help
home
gear
links
search
guide
email
Music Machines
Internet zone

drum machine samples

1. Boss DR-110 samples
2. Boss DR-55 samples
3. Korg KPR-77 samples
4. Korg KR-55 samples
5. Linn LinnDrum samples
6. Roland CompuRhythm samples
7. Roland MC-303 samples
8. Roland TR-606 samples
9. Roland TR-808 samples
10. Roland TR-909 samples
11. Sequential Tom samples
12. Univox Micro-Rhythmer-12 samples

(b)

Fig. 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.

it. The final page is combined with a standard graphical template to produce Fig. 2(b). Note that candidate index pages must be of a reasonable size. We cannot realistically present users with index pages containing hundreds of links; similarly, an index page of two or three links is not very useful. In the remainder of the paper, we discuss our approach to the first and fourth subproblems; we first discuss related work that could be applied to these problems.

3.2. Previous work

The task we address in this paper is that of examining a large space of objects (a Web site containing many documents) and finding small sets of related objects (documents pertaining to a single topic). This task is common in the data mining community, where two kinds of approaches are commonly used: statistical clustering algorithms and frequent set algorithms.

Statistical clustering (see [30,39,41]) is a technique for partitioning a set of objects into "clusters" of objects that are similar. Objects are represented in an N-dimensional space (for example, as word vectors). Roughly, a cluster is a collection of objects close to each other in this space and relatively distant from other clusters. Standard clustering algorithms *partition* the objects into a set of non-overlapping clusters. Clustering algorithms are numerous, but two common techniques are Hierarchical Agglomerative Clustering (HAC) [39], which is popular for its high-quality output, and K-means clustering [33], which is particularly fast. A well-known application of clustering algorithms to information browsing is found in Scatter/Gather [8]. The authors present a system which applies hierarchical clustering to the results of database queries; users can interactively browse the cluster hierarchy and have the system re-cluster portions of the document space. They also present two fast clustering algorithms appropriate for this application. Scatter/Gather's clustering algorithms, like many, are oriented toward document clustering; documents are represented by word vectors, and the distance between two documents is the angle between the vectors. Essentially, documents with many of the same words are considered similar. In our domain, we construct a *similarity* matrix for the pages at the Web site; similarity is based on how often pages co-occur in user visits rather than keyword comparison. We use the similarity matrix to find pairwise similarities rather than computing the distance between any kind of vectors.

Frequent set algorithms are designed to find sets of similar items in large collections (see [1,2,34,38]). In a traditional frequent set problem, the data is a collection of *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. For example, given information about grocery store purchases, the goal is to discover that, for instance, many people buy milk, butter, and yogurt at the same time. The classical frequent set algorithm is Apriori [3].

3.3. The PageGather algorithm

As described in Section 1, clustering algorithms take a collection of objects as their input and produce a *partition* of the collection—a classification of each and every object into exactly one cluster. Cluster *mining* is a variation on traditional clustering that is appropriate for our task. Instead of attempting to partition the entire space of objects, we try to find a small number of high quality clusters. Furthermore, whereas traditional clustering is concerned with placing each object in exactly one cluster, cluster mining may place a single object in multiple overlapping clusters. We formalize the cluster mining problem in Fig. 3. The relationship between traditional clustering and cluster mining is parallel to that between classification and data mining described in [35]. 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* is a cluster mining technique for finding collections of related pages at a Web site, relying on the visit-coherence assumption (see Section 2.3). In essence, PageGather takes a Web server access log as input and maps it into a form ready for

---

Given:
- A data collection D (e.g., a set of pages at a Web site).
- A pairwise similarity measure m defined over D (e.g., page co-occurrence frequencies derived from access logs).

Output all subsets c of D of size less than k such that
- c is highly cohesive with respect to m (e.g., the average pairwise similarity of objects in c exceeds some threshold).

Fig. 3. The cluster mining problem. Given a data collection and a similarity measure, find highly cohesive sets of objects.

clustering; it then applies cluster mining to the data and produces candidate index-page contents as output. The algorithm has six 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 clusters in the graph.
(4) Rank the clusters found.
(5) Eliminate overlap among the clusters.
(6) Present the clusters to the webmaster for selection; for each selected cluster, create a Web page consisting of links to the pages in the cluster.

We discuss each step in turn. As we draw upon experimental results to motivate key design decisions, we first discuss our experimental method.

3.3.1. Experimental method

Our experiments draw on data collected from three distinct collections of Web pages on two Web sites.

Our first site, *Music Machines*,[5] is a site devoted to information about many kinds of electronic musical instruments. Music Machines contains approximately 2500 distinct pages, 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.[6] Music Machines has been our primary testbed; the site is large enough and gets enough traffic to be useful. Furthermore, we have full access to all server logs and may alter the site at will.

Our second site, *Homes*, is a collection of home pages belonging to faculty, staff, and students in the Computer Science Department at the University of Washington.[7] Homes contains approximately 11,000 distinct pages and gets approximately 10,000 hits a day from 900 distinct visitors. Our training data is a collection of access logs from one month; the test data is a set of logs from the subsequent ten days.

Our third site, *Research*, consists of pages devoted to research projects in the computer science department at the University of Washington.[8] Research contains approximately 4,300 distinct pages and gets approximately 2,700 hits a day from 250 distinct visitors. Our training data is a collection of access logs from one month; the test data is a set of logs from the subsequent ten days.

Any evaluation of the effectiveness of index page synthesis ought to be based on three factors:

(1) *Recall*: how much of the information sought by each user was actually found.
(2) *Impact*: how many people use the new pages and how often.
(3) *Benefit*: how much effort is saved by those who visit the pages.

[5] http://machines.hyperreal.org.
[6] Data sets are publicly available from the authors.
[7] http://www.cs.washington.edu/people/.
[8] http://www.cs.washington.edu/research/.

Recall is the most difficult to measure. In order to gauge how much of the user's information goal has been satisfied, we must know what she sought in the first place. This information is difficult to obtain without interviewing users in some fashion regarding their goals.[9] Typically, such studies are expensive and thus only apply to fairly small populations of users. In our work, we do not measure recall. Instead, we focus on automated measurements that can be used to study the behavior of thousands of users.

In our experiments, all measurements are based on each site's access logs. In the case of Music Machines (where one of the authors is its webmaster) we were able to modify the Web site and directly measure the response of visitors to the automatically synthesized index pages that were placed on the site. In Research and Homes (and some Music Machines measurements) we report on approximate measurements that are based on historical log data. As discussed above, we broke the log data into two distinct segments: training data and test data. The algorithms were trained on the training data, and then evaluated on previously unseen test data.

While we acknowledge up front that historical data is very much an approximation, we draw some comfort from consistency of measurements across the different sites and from consistency between the approximation and the "live" user study performed on Music Machines. Nevertheless, additional user studies are necessary to corroborate and refine our results. We discuss the approximations employed below.

Impact is measured by the number of visits to the synthesized index page over the test period. When we are unable to transform the site, we approximate the impact of a new page by counting visitors who visited one or more pages that the synthesized index page links to. Because this measure counts all users who *might* have benefited from the new page, it overstates the impact; however, the measure is a good indication of the potential of the approach, and is useful for comparison between algorithms.

Benefit is more difficult to measure. User effort is a function of the number of links a user clicks on and the difficulty of finding those links. In other words, if we can not only save visitors clicks, but also decrease the complexity of pages they must navigate, we have benefited them greatly. Ideally, we would compare the average effort exerted by users with and without our transformations, but this comparison is difficult in practice. Instead, we approximate this measure by counting the number of links on a synthesized index page that a visitor clicked on. When we are unable to transform the site, we count the number of links in the synthesized page that were also present in the visitor's session at the site (as recorded by the access log). While this measurement may well underestimate the benefit of the synthesized page, this underestimate affects all the algorithms studied.

We compare competing algorithms with respect to impact and benefit. For each cluster, we count the number of pages in the cluster viewed by each visitor to the site, and compute the total number of visitors who view at least one page, the number who view at least two pages, and so on. We then average over all clusters generated by a particular algorithm. For each algorithm, we plot a line of the number of pages viewed (benefit) vs. the number of people who viewed that many pages (impact). If no users viewed more than, say, five pages from any cluster for an algorithm, then the line for that algorithm will stop at five.

[9] It's possible, of course, to place a form at a site and ask users to record their goals, but we believe that response rates to such requests are quite low and the data generated is often very noisy.

In all graphs, impact is presented on a log scale, as it often drops off exponentially with increasing benefit.

In each experiment, each algorithm outputs a ranked list of clusters, from which we choose the top ten. As discussed above, index pages cannot be too large or too small. Candidate index pages have a minimum and maximum size (5 and 100, respectively). Additionally, each algorithm we evaluate has tunable parameters. When possible, we tune the parameters to produce clusters whose average size is between 20 and 30. If an algorithm cannot produce enough clusters of this size, they may be smaller.

We now discuss each step in the PageGather algorithm.

3.3.2. 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.[10] 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.[11] A series of page views in a day's log from one visitor,[12] ordered by their time-stamps, corresponds to a single session for that visitor.

3.3.3. 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

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

[11] 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 substantially; if necessary, however, more accurate logs can be generated using cookies or visitor-tracking software such as WebThreads.

[12] 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.

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. This threshold must be tuned for a particular domain.

3.3.4. Create the graph corresponding to the matrix, and find clusters 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 [27]. We may extract clusters in two ways, leading to two variants of the PageGather algorithm.

- PG_{CC} finds all *connected components*—subgraphs in which every pair of pages has a path of edges between them.
- 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. We bound the size of discovered clusters both for efficiency—finding maximal cliques is exponential in the worst case—and because large clusters are not useful output—we cannot, practically speaking, create a new Web page containing hundreds of links.

In Section 3.3.3, we described how we find the graph threshold experimentally. In our experiments, we did this for both algorithm variants. For PG_{CC}, the threshold was 0.04 for Music Machines, 0.15 for Homes, and 0.12 for Research. For PG_{CLIQUE}, the threshold was 0.05 for Music Machines, 0.13 for Homes, and 0.08 for Research.

3.3.5. Rank the clusters found

Step (3) may find many clusters, but we 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. The webmaster may choose whether to see all clusters in ranked order, all clusters exceeding a quality threshold, or only the top-ranked clusters. In PageGather, clusters found are rated and sorted by averaging the co-occurrence frequency between all pairs of pages in the cluster.

3.3.6. Eliminate overlap among the clusters

Although some amount of cluster overlap is desirable, showing multiple highly similar clusters to the webmaster is a waste of her time. When multiple clusters overlap, there are two basic ways of reducing them to a single cluster: removing all but one cluster, or merging them together. More formally, our algorithm has two variants for overlap elimination.

- *Reduction* proceeds through the ranked cluster list in order and removes any cluster that overlaps highly with a previously seen (i.e., better) cluster.
- *Merging* also proceeds through 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 and a threshold. Our overlap measure is the size of the intersection between two clusters divided by the size of their union. We choose the threshold experimentally for each method of overlap elimination in each experimental domain. When using reduction, a low threshold means that many clusters will be eliminated; when the threshold is too high, there will be clusters remaining that are very similar. When using merging, however, a low threshold means that many disparate clusters may be combined, resulting in a small number of large, imprecise clusters; a high threshold will leave multiple similar clusters. When selecting the threshold, we set it as high as we can without producing multiple clusters that are, in our judgment, similar.

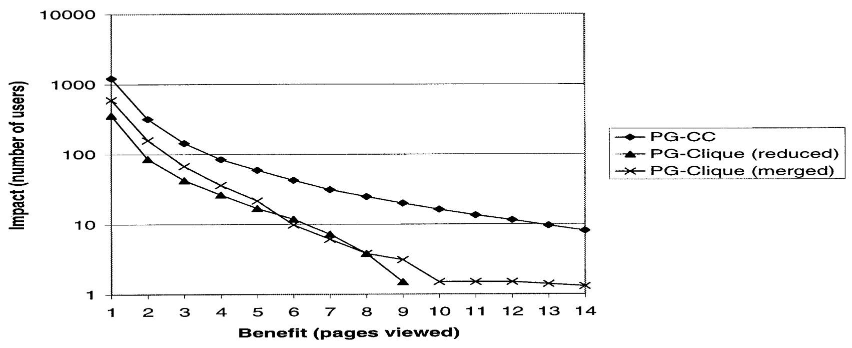
In Section 3.3.4, we discussed two different ways of producing a set of clusters. We have found experimentally that PG_{CLIQUE} tends to discover many similar clusters. This is due to the strict definition of a clique; the data may contain a set of closely connected pages that do not, however, form a clique. Many subsets of those pages, however, will form cliques, and PG_{CLIQUE} will return all of these variations on a theme, producing many overlapping clusters. Note that PG_{CC} will never produce overlapping clusters, as the connected components of a graph are, by definition, disjoint.

As overlap elimination has no effect on PG_{CC} and we may apply either reduction or merging to PG_{CLIQUE}, we have three variations of the PageGather algorithm: PG_{CC}, $PG_{CLIQUE\ (reduced)}$, and $PG_{CLIQUE\ (merged)}$. We experimentally tune the threshold as described above; thresholds for merging ranged from 0.05 for Music Machines to 0.25 for Research, and reduction thresholds ranged from 0.1 for Music Machines to 0.4 for Research. Fig. 4 compares our three variations in our three domains. We find that PG_{CC} is consistently the best or close to the best PageGather variant. Other variants sometimes perform better, but no single variant is always better and all are often much worse. PG_{CC} will therefore be the variant we use in all subsequent experiments.

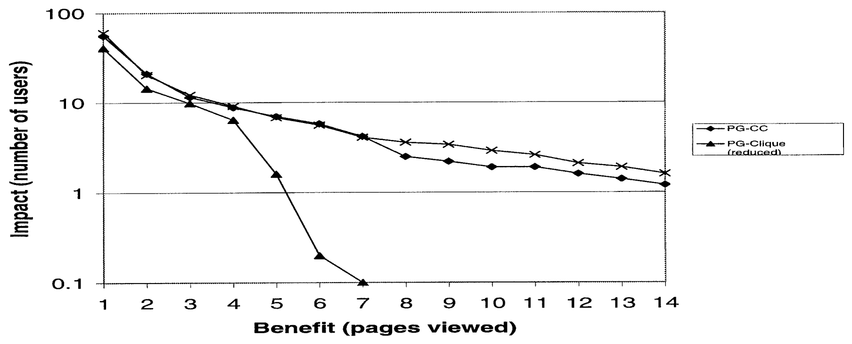
It might have been expected that some variant of PG_{CLIQUE} would produce far better clusters than PG_{CC}, as all pairs of pages in a clique are guaranteed to be highly co-occurring, while a connected component may consist merely of a long, tenuous string. On the other hand, large cliques are hard to come by, and PG_{CLIQUE} tends to find much smaller clusters than PG_{CC}. As our impact/benefit measure has a bias toward larger clusters (after all, a cluster of size five cannot have an impact greater than five on *anyone*), PG_{CLIQUE}'s clusters tend to suffer when evaluated. In some sense, PG_{CLIQUE}'s definition of a cluster is too strict: tight clusters are bought at the price of being small and difficult to find. Reduction cannot really address this problem, as it does not make the clusters any more inclusive. However, Merging addresses this problem by combining similar cliques into a single cluster which is not, itself, a clique. However, we find that merging is inexact; some cliques will be merged that should not be, and some that should be merged are not.

3.3.7. Present the clusters to the webmaster for selection

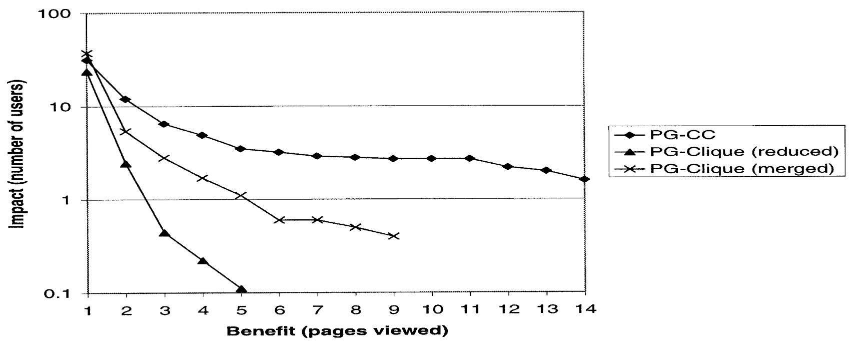
After ranking the clusters and eliminating overlap, PageGather presents them to the webmaster, who makes the final decision about what to incorporate into the Web site. The system may present only the top-ranked clusters, all clusters over some quality threshold, or the entire ranked list. PageGather presents candidates to the webmaster as in Fig. 2. The webmaster is prompted to accept or reject the cluster, to name it, and to remove any links



(a)



(b)



(c)

Fig. 4. The impact/benefit performance of three variations on our PageGather algorithm (PG_{CC}, $PG_{CLIQUE\ (reduced)}$, and $PG_{CLIQUE\ (merged)}$) on (a) Music Machines, (b) Homes, and (c) Research. PG_{CC} is consistently the best or close to the best performing variant and will be the variant used in PageGather.

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.

3.4. Time complexity

What is the running time of the PageGather algorithm? 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 $\mathrm{O}(L \log L)$ time. In step (2), we must process the log and create a matrix of size $\mathrm{O}(N^2)$, which requires $\mathrm{O}(L + N^2)$ 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. In step (3) we may find either connected components or cliques. Finding the connected components in a graph is linear in the size of the graph. In general, finding cliques is exponential; however, as we bound the size of discovered clusters to k, this step is a polynomial of degree k. Furthermore, our algorithm for finding maximal cliques implicitly divides the graph into disconnected subgraphs. When the graph is relatively sparse, the running time is only polynomial in the size of the largest subgraph; note that, as the co-occurrence threshold is increased, the graph becomes more sparse. The dominant step in our algorithm is the creation of the co-occurrence matrix; thus, overall, our algorithm requires $\mathrm{O}(L + N^2)$ time.

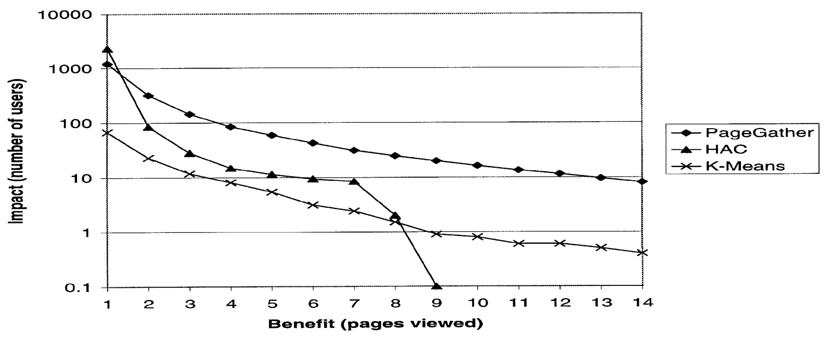
3.5. Experimental validation

In the previous section, we presented experimental comparisons among variants of our algorithm. In this section, we compare PageGather with other clustering and data mining algorithms as well as with human-authored index pages. We use the same domains and experimental methodology described in Section 3.3.1. As explained in Section 3.3.6, we use PG_{CC}—simply labeled “PageGather”—in all comparisons.

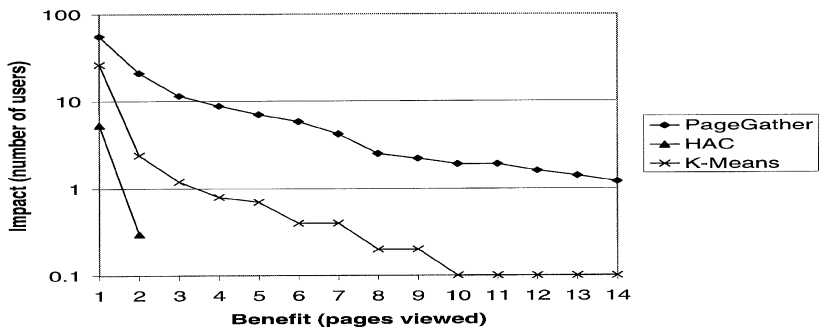
3.5.1. Clustering algorithms

In Section 3.2, we discussed traditional statistical clustering. A traditional clustering algorithm can be adapted to perform cluster mining by ranking the clusters it finds and choosing the best. In order to compare PageGather with standard clustering algorithms, we adapted both HAC and K-means to produce a large number of clusters and rank them. In the case of K-means, we set K much larger than our target of ten clusters; as in PageGather, clusters are ranked by their average pairwise co-occurrence and the top ten are evaluated. We tune K to produce the clusters of the desired size. In the case of HAC, we also set a parameter K much larger than our target number of clusters. HAC continues merging clusters until only K remain, at which point the clusters are ranked and the top ten chosen. For each algorithm in each domain, we choose K experimentally by finding the value of K that produces a sufficient number of clusters of an average size—as with PageGather, around 20–30 if possible. For HAC, K was 75 for Music Machines, 200 for Homes, and 200 for Research. For K-means, K was 60 for Music Machines, 130 for Homes, and 110 for Research.

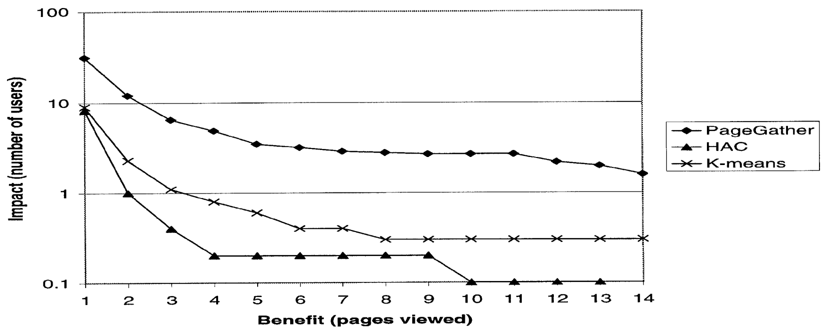
In Fig. 5, we compare PageGather with HAC and K-Means in all three domains. In all three cases, PageGather substantially outperforms both clustering algorithms.



(a)



(b)



(c)

Fig. 5. The impact/benefit performance of PageGather, HAC, and K-means on (a) Music Machines, (b) Homes, and (c) Research. PageGather consistently outperforms both traditional clustering algorithms in all domains.

3.5.2. The Apriori frequent set algorithm

In Section 3.2, 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 PageGather. In Section 3.3.6, we discussed how the strict definition of a clique causes PG_{CLIQUE} to find many similar clusters. Like PG_{CLIQUE}, Apriori has a very strict definition of a cluster. In fact, Apriori's is even stricter; whereas PG_{CLIQUE} requires that every *pair* of pages in a cluster co-occur frequently, Apriori requires that the *entire* set of pages co-occur frequently. Consequently, we find that Apriori also finds many similar clusters. We therefore examined the effect of applying overlap elimination (both reduction and merging; see Section 3.3.6) to the unprocessed output of Apriori. As described, we experimentally tune the threshold for reduction and merging in each domain; we found that thresholds of 0.2 worked in all cases. We do not present the results here, but we found that the merged results and unprocessed results both outperformed the reduced results but were comparable to each other. However, because the highly overlapping unprocessed results would be inappropriate to show a webmaster, we choose the merged results for comparison purposes.

In Fig. 6, we compare PageGather and Apriori on all three domains. On Music Machines and Research, PageGather performs substantially better, showing higher impact that is sustained with increasing benefit while Apriori falls off sharply. In the Homes domain, Apriori actually shows higher impact at low benefit but falls off sharply. In both domains, Apriori's clusters are substantially smaller than PageGather's.

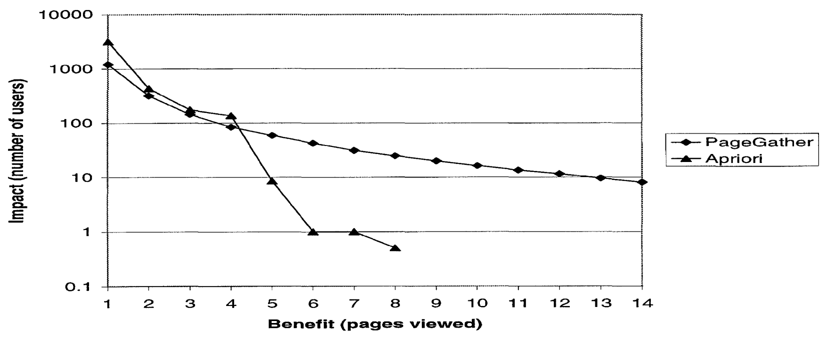
Apriori also uses a significantly different notion of a cluster than PageGather. With Apriori, two pages are likely to be clustered together if the number of times both have appeared in a user visit is high. On the other hand, in PageGather, two pages are likely to be clustered together only if their *co-occurrence* is probable relative to each page's occurrence (see Section 3.3.3). For example, let us consider two pages at a site: the front page, that is visited 1000 times a day, and an internal page which is visited 100 times a day. Because most visitors come through the front page, 90 people a day visit both pages. Apriori would consider these two pages good candidates to be clustered together, since their co-occurrence count of 90 is fairly high. PageGather, on the other hand, would calculate that their co-occurrence (90) is fairly unlikely compared to the front page's occurrence (1000) and would not cluster them together. Apriori displays the *frequent item problem*: commonly occurring items will tend to appear in many clusters. However, in this domain, this is inappropriate; we want to discover clusters of similar pages that relate to each other reciprocally.

3.5.3. PageGather and human-authored pages

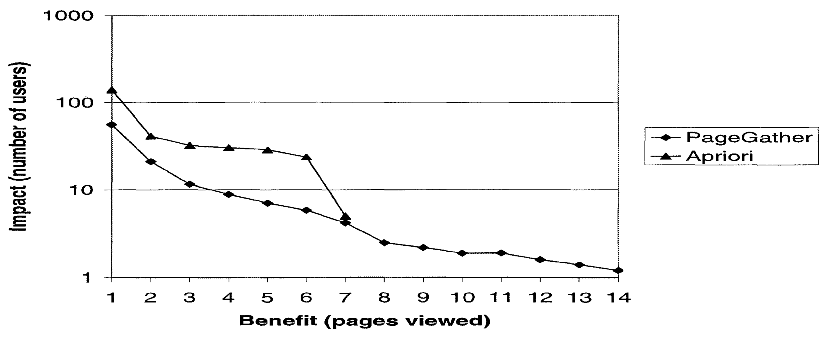
It is natural to ask how good PageGather's clusters really are, and how well an "ideal" cluster would perform. To attempt to answer these questions we look to the "ideal" case: index pages created by a human webmaster. Music Machines contains many index pages of different kinds; we chose four pertaining to representative topics:

(1) Instruments produced by a particular manufacturer (Roland).
(2) Pages pertaining to a particular instrument (the Roland Juno keyboard).
(3) All instruments of a particular type (drum machines).
(4) All files of a particular type (audio samples).

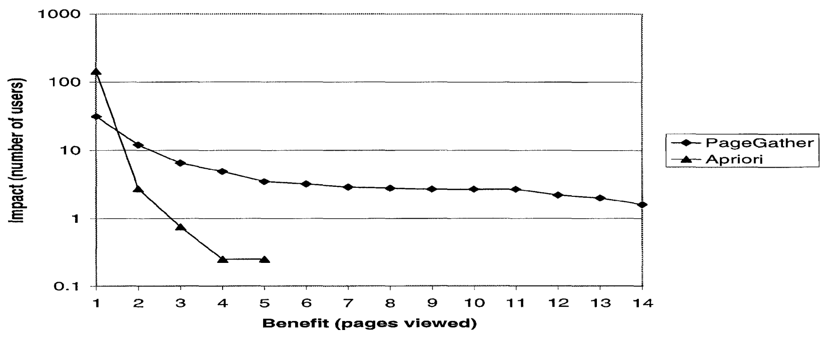
The set of outgoing links on each page (excluding standard navigation links) was treated as a "cluster" and evaluated on our test data. Fig. 7 shows the comparison of these clusters to the output of PageGather. We see that the human-authored clusters perform better than PageGather's.



(a)



(b)



(c)

Fig. 6. The impact/benefit performance of PageGather and Apriori on (a) Music Machines, (b) Homes, and (c) Research. PageGather outperforms Apriori in two of our three domains. In all domains, however, PageGather shows greater benefit to users than Apriori.

4. Conceptual cluster mining

PageGather relies on a statistical approach to discovering candidate link sets; its candidates do not correspond precisely to intuitive concepts, whereas human-authored

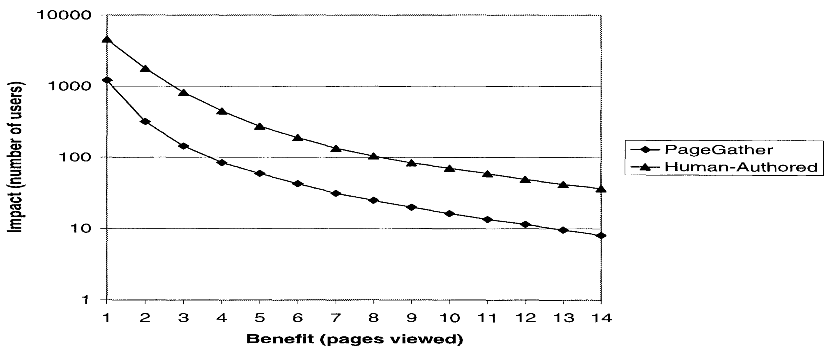


Fig. 7. The impact/benefit performance of PageGather and clusters derived from human-authored index pages on Music Machines. The human-authored clusters outperform PageGather.

Given:

- A data collection D (e.g., a set of pages at a Web site).
- A pairwise similarity measure m defined over D (e.g., page co-occurrence frequencies derived from access logs).
- A conceptual language L for describing elements of D (e.g., conjunctions of descriptive features).
- A description in L of each object in D.

Output all subsets c of D of size less than k such that

- c is highly cohesive with respect to m (e.g., the average pairwise similarity of objects in c exceeds some threshold).
- c corresponds to a concept expressible in L.

Fig. 8. The conceptual cluster mining problem. Given a data collection, a similarity measure, a conceptual language L, and conceptual descriptions of data objects, find sets of objects that are both similar and correspond to concepts expressible in L.

index pages do. We therefore desire an algorithm that will find only candidate link sets that are conceptually coherent. Our solution is a key extension to PageGather that guarantees that we will generate only link sets that correspond to topics, given a language for describing topics. Our extension also constitutes a novel approach to the long-standing problem of conceptual clustering [20]. We next define the *conceptual* cluster mining problem, describe previous work, and present our own approach. We then describe the IndexFinder algorithm and present experimental evaluation.

4.1. Problem definition

In the previous section, we presented cluster mining, and described the PageGather algorithm. We now introduce *conceptual cluster mining*: given a collection of objects, a pairwise similarity measure over those objects, and a conceptual language for describing them, we search for a small set of cohesive clusters that *correspond to concepts expressible in the language.* We formalize the conceptual cluster mining problem in Fig. 8.

4.2. Previous work

Relevant previous work is of three types: *statistical* approaches to clustering and data mining, *conceptual* clustering, and *concept learning* algorithms. Statistical approaches include those surveyed in Section 3.2 and PageGather itself. All of these statistical approaches are useful for finding cohesive sets of objects in large collections of data, but make no attempt to ensure that their results correspond to an intuitive concept.

Conceptual clustering algorithms (see [20]), like their statistical counterparts, partition a data collection into clusters of similar objects. In a conceptual clustering problem, however, objects are described in a conceptual description language, and each cluster corresponds to a concept expressible in that language. These conceptual languages may be simple conjunctions of attributes as in [20], probabilistic descriptions as in [11,12], or complex and cognitively inspired as in [14]. Perhaps the most familiar conceptual clustering algorithm is Fisher's COBWEB [11,12]. COBWEB constructs a hierarchical partition of an object space based on conceptual structure found among the objects. As discussed with respect to statistical clustering algorithms, these algorithms attempt to find the best partition of the entire space rather than the best single concepts. Note, however, that a conceptual hierarchy could be of great use for Web sites. Whereas we have focused on finding single concepts that correspond to what users seek, a conceptual hierarchy provides a complete view of the site's pages all at once. A key difference between such an approach and our approach is that the hierarchy is based entirely on the conceptual structure of the pages at the site, without any reference to how the site's visitors use them; although this could be of great use for automating Web site creation, it is not really *adaptive* to the needs of users.

As described in Section 1, the goal of a Concept learning algorithm (see [22]) is to find a conceptual description of a set of objects from a data collection. Concept learning algorithms are typically either decision-oriented (e.g., [28]) or rule-oriented (e.g., [7,35]). A decision-oriented learning algorithm constructs a decision process which determines whether any object is in the target class. A rule-oriented algorithm finds predictive rules; objects that match the rule are likely to be in the class. This distinction mirrors that between clustering algorithms, which classify all objects by partitioning the object space, and cluster mining algorithms, which place only some objects in clusters. Whereas conceptual clustering algorithms both partition the data collection into classes and provide conceptual descriptions of each class, concept learning algorithms require that the data be classified in advance. As such, they are not, in themselves, cluster mining algorithms; however, they could be a partial solution when paired with some technique (e.g., statistical clustering) for finding sets of objects. In particular, rule-oriented approaches may be effective when combined with cluster mining techniques.

Mirkin [21] presents an approach that draws from both clustering and concept learning. He describes a method that, given a clustering over a space of objects described by attributes, computes each attribute's "contribution" to each cluster. He then presents an algorithm for finding an approximate conjunctive description of each cluster using these contribution values. Attributes with high contribution values are not necessarily the best attributes for a conjunctive description; Mirkin's algorithm is a greedy technique that uses the contribution values as a heuristic to select attributes to consider. The algorithm does not, in itself, represent a solution to the cluster mining problem; however, as we next describe, a concept-learning algorithm of this kind can be part of an effective solution to the conceptual cluster mining problem.

4.3. The SCML algorithm schema

The naive algorithm for the conceptual cluster mining problem might look like this:

(1) Enumerate all concepts l expressible in L.

(2) For each l, compute l's cohesiveness with respect to the similarity measure m.

(3) Return all l with a cohesiveness score over a certain threshold.

Of course, the number of expressions in L could be quite large. For example, when L consists of conjunctions of attributes, the number of expressions in L is exponential in the number of attributes, making this algorithm impractical for real applications. How can we devise a tractable algorithm for this problem? We've already described cluster mining, a tractable technique for finding cohesive *statistical* clusters; if we could efficiently transform these clusters into conceptual ones, we would have a tractable algorithm. In essence, we would like to find the conceptual cluster that best approximates each statistical one.

Our key insight was that the members of a statistical cluster can be viewed as positive examples of the desired concept, and objects outside the cluster as negative examples, enabling us to apply concept learning techniques to efficiently generate the desired approximations. This observation led us to the SCML algorithm schema (Fig. 9), which uses statistical cluster mining to find cohesive clusters and concept learning to find a concept that describes each statistical cluster. As the match between statistical and conceptual clusters will be imperfect, we will require a noise-tolerant learning algorithm. The output of this schema is a set of clusters (and their conceptual descriptions). This approach is not guaranteed to find the optimal set of conceptual clusters, but it is tractable and it enables us to leverage continuing advances in concept learning research.

```
The SCML Algorithm Schema [$\Omega, \Gamma$]
1. Run statistical clustering algorithm $\Omega$ on the data collection $D$, producing a set $C$ of clusters.
2. For each cluster $c$ in $C$
    (a) Tag the objects in $c$ as positive examples.
    (b) Tag remaining objects $(D - c)$ as negative examples.
    (c) Use these as input to concept learning algorithm $\Gamma$.
    (d) Find the concept $v = \Gamma(c, D - c, L)$.
    (e) Find the set of objects $c_v$ which is the extension of $v$.
3. Return all the sets $c_v$ found.
```

Fig. 9. The SCML algorithm schema, which is instantiated with a statistical cluster mining algorithm and a concept learning algorithm.

4.4. SCML and index page synthesis

4.4.1. Instantiations of SCML

We have presented the general conceptual cluster mining problem. In this section, we instantiate the SCML schema to generate three algorithms for the index page synthesis problem. We then compare these three instantiations experimentally.

We instantiate the SCML schema using PageGather for Ω. We use three different algorithms for Γ. Because Music Machines was the only Web site for which we could obtain conceptual descriptions, all experimental evaluations of our conceptual approach were performed on that site. For Music Machines, our conceptual language L consists of conjunctions of attributes describing pages and musical instruments (e.g., type of instrument, price, and type of file; see Fig. 14 for examples).

As a baseline, we wished to use a simple algorithm that would perform a greedy search from general to specific, constructing a conjunctive rule, as this approach seems well-tailored to our problem. Accordingly, we based our algorithm on GREEDY-3 [23]. The algorithm iteratively builds a conjunctive rule, choosing a conjunct that maximizes a scoring function at each iteration.

With GREEDY-3, we use two scoring functions, producing two instantiations of SCML. As positive examples in our domain are typically vastly outnumbered by negatives and are therefore more important, the first scoring function—called PG_{POS}—is a simple count of positives covered by the rule, with negatives used to break ties. We use a maximum rule length to limit the complexity of rules. The second—PG_{MDL}—is based on minimum description length or MDL [29]. MDL is a measure of how compactly a collection of data can be represented by a classification rule or decision tree. MDL is measured in terms of the number of bits necessary to encode the classification rule and the data examples that are exceptions to the rule. In many cases, a more specific classification rule may correctly classify a few more examples, but the additional complexity of the rule is not worth the marginal improvement in classification; the MDL measure takes this tradeoff into account. In our PG_{MDL} algorithm, whenever we consider adding a new conjunct to our classification rule, we compare the description length of the current rule to that of the new rule. Only if adding the conjunct decreases the description length do we extend the rule. We use the formula Quinlan and Rivest give for MDL:

$$L\left(n, k, \frac{n+1}{2}\right) = \lg(b+1) + \lg\left(\binom{n}{k}\right),$$

where n is the total number of examples, and k is the number of positive examples. We also take into account the cost of specifying the attribute ($\lg(A)$, where A is the number of attributes) and the value of the attribute ($\lg(V)$, where V is the number of values) for each conjunct. Because the MDL measure takes this tradeoff into account, we do not need a maximum rule length as we did with PG_{POS}.

The third algorithm is PG_{RIP}, which uses RIPPER*k* [7] for the concept learning algorithm Γ. RIPPER*k* is a rule-learning algorithm designed to work well with large, noisy datasets and is competitive with the classic concept learning systems C4.5 and C4.5 rules in both speed and performance. We chose RIPPER*k* over a decision-oriented algorithm because RIPPER*k* learns sets of rules—which correspond nicely to the single concepts we wish to find—rather than entire decision trees. In Section 4.4.2, we present results from PG_{POS}, PG_{MDL}, and PG_{RIP}.

In Section 3.1, we decomposed the index page synthesis problem into five subproblems. We have primarily focused on finding the contents of index pages. However, conceptual cluster mining allows us to also generate the titles of synthesized pages automatically. For example, if our algorithm finds a candidate index page with the conceptual description `"age=vintage, instrument-type=sequencer"`, it may name that page "vintage sequencers" when displaying it to the webmaster for approval. The webmaster, in fact, need only look at the title to determine whether the page is appropriate for the site.

4.4.2. Experimental comparison of SCML instantiations

We compare the three SCML instantiations described above. We use the same experimental data and evaluation metric described in Section 3.3.1. However, in order to apply conceptual algorithms to a Web site, we need a conceptual description of every document at the site, provided by the expert webmaster. We have this information only for Music Machines—it is the only Web site for which one of the authors was webmaster. Fig. 10 shows the performance of three instantiations of SCML on the Music Machines site. PG_{MDL} shows a slight edge over PG_{POS}, and both perform better than PG_{RIP}.

For a second comparison between conceptual algorithms, we place automatically generated index pages at the Music Machines site and observe how people use them. Each algorithm generates a set of clusters, of which the top ten are chosen. Each cluster is converted into a Web page containing a link to each page in the cluster. Link text is generated from the title of the linked-to page (or the filename if no title is available). Each cluster is presented to the webmaster, along with its conceptual description, for naming. For example, presented with the rule `"filetype=image, price=cheap, instrument-type=synth"`, the webmaster would name the page "images of cheap synthesizers". Each day, five of these index pages are randomly chosen and made accessible on the Web site. The index pages appear in the site's navigation bar, which appears on all site pages. Over the course of the day, we record the number of hits and

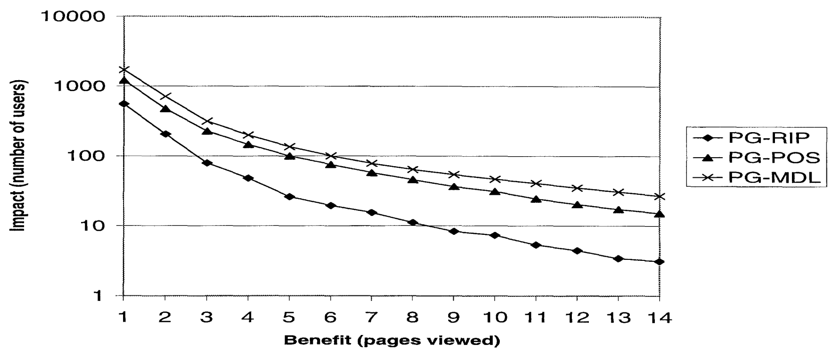


Fig. 10. The impact/benefit performance of PG_{RIP}, PG_{POS}, and PG_{MDL} on Music Machines. PG_{MDL} performs somewhat better than the other two and will be used for our IndexFinder algorithm.

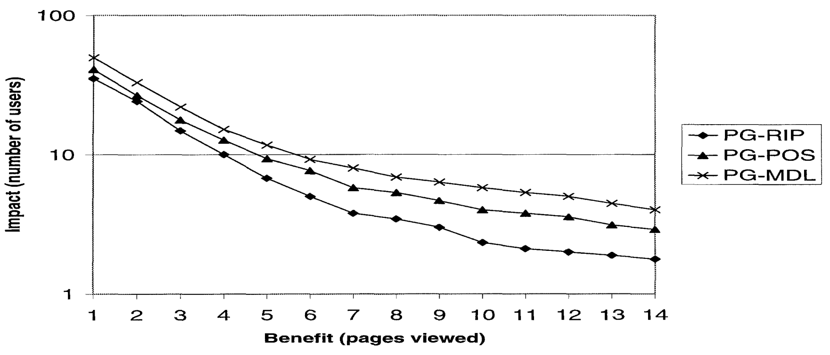


Fig. 11. The impact/benefit performance of index pages generated by PG_{RIP}, PG_{POS}, and PG_{MDL} and actually placed at the Music Machines site. PG_{MDL} outperforms both other SCML instantiations.

clicks-through for each page. Over approximately one month, we track how many users visit each index page, and how many of the page's links each user clicks on per day. We use the same impact/benefit metric as above; the only difference is that we now directly count links in the index page that a user visits.

Fig.11 shows the performance for the three instantiations of SCML. As before, PG_{MDL} performs better than both other algorithms. We will be using the PG_{MDL} instantiation of SCML—called IndexFinder—in further experimental comparisons.

4.5. Experimental evaluation

In this section, we evaluate IndexFinder, comparing it to PageGather, COBWEB [11,12], and clusters derived from human-authored pages at a Web site. We use the impact/benefit measure described in Section 3.1 as well as placing automatically generated index pages on the Music Machines site as described in Section 4.4.2.

COBWEB builds a hierarchical partition of the object space. However, in our domain, we require a ranked list of clusters from which we can choose the top ten. We therefore consider *all* clusters found by COBWEB, at all levels of the hierarchy. Clusters below a minimum size are discarded, and all others are ranked by computing the average pairwise co-occurrence as used in PageGather's similarity matrix. The top ten clusters by this ranking are used in our experiments.

In Fig. 12, we compare IndexFinder to PageGather, COBWEB, and human-authored index pages. IndexFinder outperforms both COBWEB and PageGather and is close to the performance of the human-authored index pages.

4.5.1. On-site experiments

We have experimentally compared IndexFinder to other algorithms using training and testing sets drawn from historical log data; we now present preliminary results from placing automatically-generated index pages at the Music Machines Web site. In this experiment, conceptual clusters were generated using IndexFinder, COBWEB, and human-authored index pages. Naturally, the human-authored pages already exist at the site, but we include

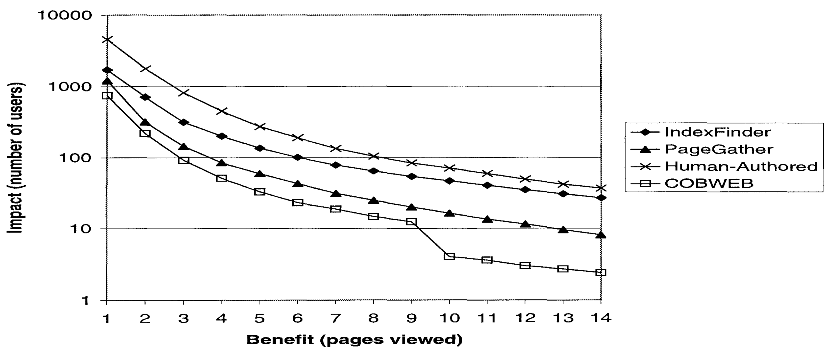


Fig. 12. The impact/benefit performance of IndexFinder, PageGather, COBWEB, and clusters derived from human-authored index pages on Music Machines. IndexFinder outperforms both PageGather and COBWEB and is close to the performance of the human-authored index pages.

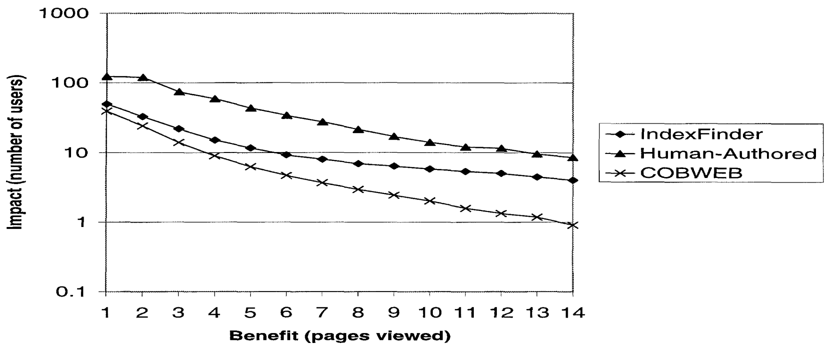


Fig. 13. The impact/benefit performance of index pages generated by IndexFinder and COBWEB and clusters derived from human-authored index pages when actually placed at the Music Machines site. IndexFinder does substantially better than COBWEB but not as well as the human-authored pages.

them for the purposes of experimental comparison; they are placed in the site's navigation bar alongside other experimental pages.

In Fig. 13, we show the impact/benefit performance for IndexFinder, COBWEB, and human-authored index pages. IndexFinder does substantially better than COBWEB but not as well as the human-authored pages.

4.5.2. Discussion

The main purpose of using a conceptual approach is to guarantee that any index pages we generate make sense to the users of the Web site. Accordingly, these concepts should be fairly simple—i.e., without too many conjuncts. Fig. 14 shows the top five concepts found by both PG_{RIP} and COBWEB. Although both algorithms produce clusters of a reasonable size, COBWEB's concepts tend to be more complex.

	Rule
IndexFinder	filetype=index, age=vintage, instrument-type=sequencer
	filetype=index, instrument-type=controller
	instrument-type=sequencer, price=expensive
	filetype=samples, age=vintage, price=cheap
	filetype=samples, form=tabletop, price=cheap
COBWEB	age=vintage, filetype=mods, form=keyboard, instrument-type=synth, midi=no, pagetype=instrument, price=midpriced, synthesis=analogue, sonic=monophonic
	age=modern, instrument-type=synth, pagetype=instrument, price=midpriced, synthesis=analogue, sonic=monophonic
	filetype=info, instrument-type=synth, pagetype=instrument
	filetype=info, form=module, instrument-type=sampler, midi=yes, pagetype=instrument, price=midpriced, synthesis=digital, sonic=sampler
	age=vintage, form=keyboard, instrument-type=sampler, midi=yes, pagetype=instrument, synthesis=hybrid, sonic=sampler

Fig. 14. Conceptual descriptions of the top five clusters found by IndexFinder and COBWEB.

5. Conclusion and future work

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 have motivated the notion of adaptive Web sites and analyzed the design space for such sites, locating previous work in that space. To demonstrate the feasibility of nontrivial adaptations, we presented a case study in the domain of synthesizing new index pages. Our key technical contributions are the following:

(1) As an approach to index page synthesis, we presented the PageGather algorithm, the SCML algorithm schema, and the IndexFinder algorithm.
(2) In experiments on three Web sites, we showed that our methods outperform traditional methods including the Apriori data mining algorithm, standard clustering algorithms, and the COBWEB conceptual clustering algorithm on the index page synthesis task. We also compared the output of automatic methods with human-authored index pages. We have placed automatically generated index pages on an actual Web site and observed user response.
(3) PageGather and IndexFinder are instances of novel, domain-independent approaches to unsupervised data mining, which we called *cluster mining* and *conceptual cluster mining*. We conjecture that these approaches will find extensions and applications outside the domain of adaptive Web sites.

In future work, we plan to automate the placement of new index pages at the Web site. We will explore both automatically suggesting names for new pages based on the predicate description and deciding where in the site they should be located. Finally, index page synthesis itself is a step towards the long-term goal of *change in view*: adaptive sites that automatically suggest re-organizations of their contents based on visitor access patterns.

References

[1] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases, in: Proc. ACM SIGMOD Conference on Management of Data, 1993, pp. 207–216.

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

[3] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: Proc. 20th VLDB Conference, 1994.

[4] T. Anantharaman, M. Campbell, F. Hsu, Singular extensions: Adding selectivity to brute-force searching, Artificial Intelligence 43 (1) (1990) 99–109.

[5] E. André, W. Graf, J. Müller, H.-J. Profitlich, T. Rist, W. Wahlster, AiA: Adaptive communication assistant for effective infobahn access, Document, DFKI, Saarbrücken, 1996.

[6] M. Balabanovic, Y. Shoham, Fab: Content-based, collaborative recommendation, 1997.

[7] W. Cohen, Fast effective rule induction, in: Proc. 12th Internat. Conference on Machine Learning, Tahoe City, CA, 1995.

[8] D. Cutting, D. Karger, J. Pedersen, J. Tukey, Scatter/gather: A cluster-based approach to browsing large document collections, in: Proc. 15th Annual International SIGIR92, 1992.

[9] M. Fernandez, D. Florescu, J. Kang, A. Levy, D. Suciu, System demonstration—strudel: A Web-site management system, in: Proc. ACM SIGMOD Conference on Management of Data, 1997.

[10] J. Fink, A. Kobsa, A. Nill, User-oriented adaptivity and adaptability in the AVANTI project, in: Designing for the Web: Empirical Studies, Microsoft Usability Group, Redmond, WA, 1996.

[11] D. Fisher, Knowledge acquisition via incremental conceptual clustering, Machine Learning 2 (1987) 139–172.

[12] D. Fisher, Iterative optimization and simplification of hierarchical clusterings, J. Artificial Intelligence Res. 4 (1996).

[13] N. Good, J. Schafer, J. Konstan, A. Borchers, B. Sarwar, J. Herlocker, J. Riedl, Combining collaborative filtering with personal agents for better recommendations, in: Proc. AAAI-99, Orlando, FL, 1999.

[14] S. Hanson, M. Bauer, Conceptual clustering, categorization, and polymorphy, Machine Learning 3 (1989) 343–372.

[15] T. Joachims, D. Freitag, T. Mitchell, Webwatcher: A tour guide for the World Wide Web, in: Proc. IJCAI-97, Nagoya, Japan, 1997, pp. 770–775.

[16] R. Khare, A. Rifkin, XML: A door to automated Web applications, IEEE Internet Computing 1 (4) (1997) 78–87.

[17] J. Kleinberg, Authoritative sources in a hyperlinked environment, in: Proc. 9th ACM-SIAM Symposium on Discrete Algorithms, 1998.

[18] H. Lieberman, Letizia: An agent that assists Web browsing, in: Proc. IJCAI-95, Montreal, Quebec, 1995, pp. 924–929.

[19] S. Luke, L. Spector, D. Rager, J. Hendler, Ontology-based Web agents, in: Proc. First Internat. Conference Autonomous Agents, Marina del Rey, CA, 1997.

[20] R. Michalski, R. Stepp, in: Learning from Observation: Conceptual Clustering, Morgan Kaufman, San Mateo, CA, 1983, pp. 331–363.

[21] B. Mirkin, Concept learning and feature selection based on square-error clustering, Machine Learning 35 (1999) 25–39.

[22] T. Mitchell, Machine Learning, McGraw Hill, New York, 1997.

[23] G. Pagallo, D. Haussler, Boolean feature discovery in empirical learning, Machine Learning 5 (1990) 71–100.

[24] M. Pazzani, J. Muramatsu, D. Billsus, Syskill and Webert: Identifying interesting Web sites, in: Proc. AAAI-96, Portland, OR, 1996.

[25] M. Perkowitz, O. Etzioni, Adaptive Web sites: An AI challenge, in: Proc. IJCAI-97, Nagoya, Japan, 1997.

[26] M. Perkowitz, O. Etzioni, Adaptive Web sites: Automatically learning from user access patterns, in: Proc. 6th Internat. WWW Conference, Santa Clara, CA, 1997.
[27] M. Perkowitz, O. Etzioni, Adaptive Web sites: Automatically synthesizing Web pages, in: Proc. AAAI-98, Madison, WI, 1998.
[28] J.R. Quinlan, Induction of decision trees, Machine Learning 1 (1986) 81–106.
[29] J.R. Quinlan, R.L. Rivest, Inferring decision trees using the minimum description length principle, Inform. and Comput. 80 (1989) 227–248.
[30] E. Rasmussen, Clustering algorithms, in: W.B. Frakes, R. Baeza-Yates (Eds.), Information Retrieval, Prentice Hall, Englewood Cliffs, NJ, 1992, pp. 419–442.
[31] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, J. Riedl, GroupLens: An open architecture for collaborative filtering of netnews, in: Proc. ACM Conference on Computer-Supported Cooperative Work, 1994, pp. 175–186.
[32] T. Rist, T.E. André, T.J. Müller, Adding animated presentation agents to the interface, in: Proc. 1997 International Conference on Intelligent User Interfaces, Orlando, FL, 1997, pp. 79–86.
[33] J. Rocchio, Document retrieval systems—Optimization and evaluation, Ph.D. Thesis, Harvard University, Cambridge, MA, 1966.
[34] A. Savasere, E. Omiecinski, S. Navathe, An efficient algorithm for mining association rules in large databases, in: Proc. 21st VLDB Conference, Zurich, Switzerland, 1995.
[35] R. Segal, Data mining as massive search, Ph.D. Thesis, University of Washington, Seattle, WA, 1996. http://www.cs.washington.edu/homes/segal/brute.html.
[36] U. Shardanand, P. Maes, Social information filtering: Algorithms for automating "word of mouth", in: Proc. Conference on Human Factors in Computing Systems—CHI-95, 1995.
[37] C. Thorpe (Ed.), Vision and Navigation: The Carnegie Mellon Navlab, Kluwer Academic, Boston, MA, 1990.
[38] H. Toivonen, Sampling large databases for association rules, in: Proc. 22nd VLDB Conference, Bombay, India, 1996, pp. 134–145.
[39] E.M. Voorhees, Implementing agglomerative hierarchical clustering algorithms for use in document retrieval, Inform. Process. Management 22 (1986) 465–476.
[40] A. Wexelblat, P. Maes, Footprints: History-rich Web browsing, in: Proc. Conference Computer-Assisted Information Retrieval (RIAO), 1997, pp. 75–84.
[41] P. Willet, Recent trends in hierarchical document clustering: A critical review, Inform. Process. Management 24 (1988) 577–597.
[42] T. Yan, H. Jacobsen, H. Garcia-Molina, U. Dayal, From user access patterns to dynamic hypertext linking, in: Proc. 5th Internat. WWW Conference, Paris, France, 1996.