Adaptive Web Sites: Cluster Mining and Conceptual Clustering for Index Page Synthesis

Mike Perkowitz

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

2001

Program Authorized to Offer Degree: Computer Science and Engineering

University of Washington

Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Mike Perkowitz

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of Supervisory Committee:

Oren Etzioni

Reading Committee:

Pedro Domingos

Henry Kautz

Date:

University of Washington

Abstract

Adaptive Web Sites: Cluster Mining and Conceptual Clustering for Index Page Synthesis

by Mike Perkowitz

Chair of Supervisory Committee

Associate Professor Oren Etzioni
Computer Science and Engineering

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 dissertation 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.

Specifically, this dissertation addresses the problem of **index page synthesis**: creating new web pages on topics of interest to users. The new class of *cluster mining* algorithms is defined; whereas traditional clustering algorithms partition an entire space of objects, cluster mining algorithms mine the space for interesting clusters. This dissertation presents the PageGather cluster mining algorithm. Index pages must be comprehensible to human users, a requirement that is difficult to meet with a purely statistical data mining algorithm such as PageGather. Accordingly, the class of *conceptual cluster mining* algorithms — cluster mining algorithms that produce only clusters corresponding to a well-defined concept — is defined, and the IndexFinder algorithm is presented.

These algorithms are useful variants on traditional data mining algorithms for problems — such as page synthesis — requiring both the mining of large amounts of data and the generation of human-readable output. The comparison of PageGather and IndexFinder to

other data mining algorithms also sheds light on the tradeoffs inherent in choosing a data mining algorithm for a particular problem.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I cannot reflect on the years behind this dissertation without feeling gratitude to the people who have contributed to my academic life. First, I would like to thank my advisor Oren Etzioni. Oren has guided my academic development both by example and with an astute sense of when to guide and when to let me make my own mistakes. The AI faculty here at UW have always been stimulating and supportive. Thank you to Dan Weld, whose perceptive criticism and breadth of knowledge astound and to Steve Hanks, who has offered many suggestions, insights, and incredible cookies. Thank you also to Pedro Domingos and Henry Kautz for reading this dissertation and offering comments.

The AI students at UW have also been a great community. Neal Lesh, Nick Kushmerick, and Keith Golden cultivated a warm culture that sometimes even wandered onto AI topics. AnHai Doan and Oren Zamir have offered great help and advice with ideas and papers. Thank you also to my real and honorary office mates — Brad Chamberlain, Alec Wolman, Geoff Voelker, Jim Fix, E. Lewis, Ben Dugan, and Sung Choi.

Thank you of course to my parents, who have been inspiring and supportive, and who finally figured out to stop asking if I was done writing.

Chapter 1

INTRODUCTION

1.1 Motivation

A Web site is a complex interaction between content, user interface, and implementation. The designer of a site must perform several difficult tasks: choosing and organizing the content, designing the user interface (both the style and the forms of interaction), and anticipating user needs and making sure the organization and interface answer those needs.

At a site of any size, the amount of content may be immense and complexly organized. Complex relationships among the data may mean that multiple "views" are possible, each enabling different kinds of interaction. For example, an ecommerce site that provides a catalog of products could be viewed by *manufacturer* — facilitating comparisons between different products from one company — or by *product type*, facilitating comparisons between similar products from different companies. Many such views are possible, and the views a site provides determine which kinds of user interactions are well-supported.

However, the webmaster cannot simply throw views onto the site willy-nilly. Increased complexity in the site's organization leads to increased complexity in its interface. An interface in which all things are possible is often one in which no things are easy. The webmaster must trade off increased complexity against the benefits of increased flexibility and decide what is vital to users. Understanding user needs requires understanding how users view the data available and how they actually use the site. For a complex site this can be a difficult task; user tests are expensive and time consuming, and the site's server logs contain massive amounts of data.

We believe that web sites must become *adaptive* — capable of improving their organization and presentation by learning from the way people use them. The disciplines of artificial intelligence, data mining, and machine learning offer techniques to find common patterns in the actions of web surfers and learn from those patterns. We propose to use these techniques to develop a *semi-automated web management assistant* — a system that can process massive amounts of data about site usage and suggest useful adaptations to the webmaster. In this dissertation, we will both discuss general issues in the development of adaptive web sites and describe IndexFinder, a web management assistant we have designed, implemented, and tested.

1.2 IndexFinder

IndexFinder will be described in detail in the remainder of this thesis; we here give a brief overview of the system. IndexFinder is a web management assistant that suggests to the webmaster new index pages to be added to a web site. An *index page* is a page consisting of links to a set of pages that cover a particular topic (e.g., minivans). On the assumption that visitors to the site will tend to look at multiples pages pertaining to particular topics, IndexFinder does this by examining the site's server logs and finding sets of pages that are often viewed together in visits to the site. IndexFinder then figures out the topic these pages have in common and proposes a new page devoted to that topic, containing links to all relevant pages.

For example, imagine a site devoted to information about automobiles. The webmaster initially decides to organize the site by manufacturer; each auto company will be represented by a dedicated page, containing links to each of its models. However, many visitors to this site intend to comparison shop; a visitor wanting to compare minivans, for example, would have to go to each manufacturer in turn and look up its minivan offerings. IndexFinder will observe this common behavior pattern and suggest that the webmaster add a "minivans" page containing a link to each minivan directly.

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

1. What is the content (i.e. set of hyperlinks) of the index page?

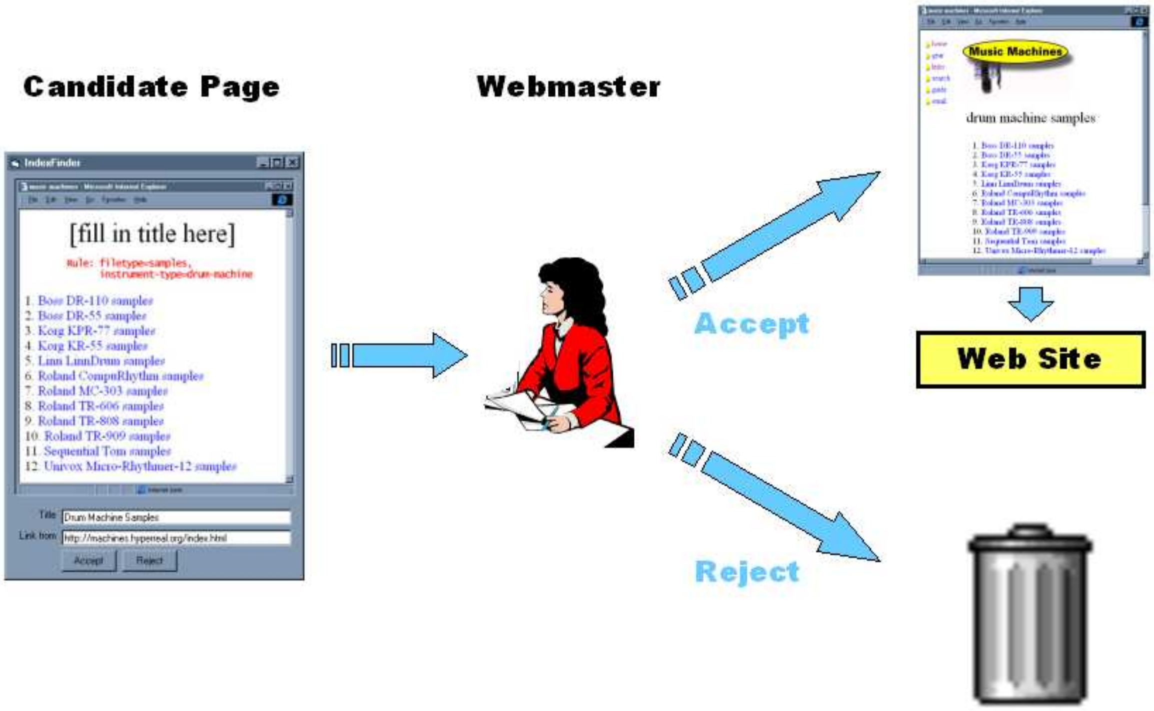

Figure 1.1: IndexFinder suggests candidate index pages to the webmaster.

2. Does it have a coherent topic? What should its title be?

3. How are the hyperlinks on the page ordered?

4. How are the hyperlinks labeled?

5. Is it appropriate to add the page to the site? If so, where?

IndexFinder does most of this work automatically, leaving to the webmaster the questions of whether and where the page should be added to the site. The first two subproblems have been the focus of our research. The third and fourth are adequately handled with simple rules; hyperlinks are labeled with the title of the target page and ordered alphabetically. Naturally, future work may explore other options. The fifth is left up to the judgement of the webmaster. When IndexFinder generates a candidate page, the contents are presented to the webmaster. IndexFinder also generates a conceptual description of the page's topic, defining what appears on the page.

Figure 1.1 shows a model of how IndexFinder and the human webmaster interact. IndexFinder processes the site's access logs and generates a candidate index page. The rule shown on the candidate index page specifies the topic of the page (here, audio samples of drum machines). The webmaster is prompted to accept or reject the candidate page and to provide a title based on the description. The webmaster also specifies where in the site the page should be linked. If the page is accepted, IndexFinder creates a final version of the page, using the supplied title and a graphic template for the site. Finally, the page is linked into the site. This model has been partially implemented — the algorithms have been realized although the graphical interface has not.

The main source of information we rely on is the site's web server log, which records the pages visited by a user at the site. We also make the **visit-coherence assumption**: the pages that a user visits during one session at the site are likely to be closely related. 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; visitors may pursue multiple distinct interests in a single visit, follow the wrong links, or simply get distracted. 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.

Knowing what pages are related in users' minds is difficult. Actually asking users is time-consuming and intrusive. On the other hand, analyzing pages for similarity is not only difficult, as computers do not yet understand what they read, but is also insensitive to the particular needs of the site's actual users. By providing a manageable definition of "relatedness," the visit-coherence assumption allows us to use the access logs to approximate what the site's users would tell us if we could ask them all.

IndexFinder's basic algorithm is shown in Figure 1.2. Initially, we hoped, based on the visit-coherence assumption, that the results of step (2) would be coherent sets of pages

1. Examine the log of each user's visit to the site, and record pages that appear together in each user visit.
2. Find sets of pages that often occur together in many visits.
3. For each such set, find the topic that most closely ties those pages together.
4. Remove any pages in the set that do not fit the topic, and add any missing pages from the site that do.
5. Construct an index page from the topic and set of pages, and present it to the webmaster for approval.
6. If the new index page is approved, add it to the site.

Figure 1.2: The basic IndexFinder algorithm.

corresponding to some specific topic. The first version of IndexFinder — called PageGather — had no steps (3) and (4); the webmaster examined the sets output by step (2), figured out the common topic, and gave the page an appropriate name. However, the statistical data mining technique used in step (2), though effective in finding sets of commonly co-occurring pages, is prone to noise and has no way to guarantee that the pages will all pertain to the same topic. Therefore, we added conceptual descriptions of each web page (e.g., one page is about an inexpensive, American-made guitar, while another contains audio samples of an expensive Japanese keyboard) and an algorithmic step that uses those descriptions to try to find the common topic among a set of pages. The topic found is enforced by removing any pages in the set that do not fit the topic, and including any other pages from the site that do.

IndexFinder has been evaluated using data from three web sites — **Music Machines**,

Homes, and **Research** — and is in active use on **Music Machines**. **Music Machines**[1] is devoted to information about many kinds of electronic musical instruments. The site is maintained by the author. **Homes** is a collection of pages from the web sites of faculty, staff, and students our computer science department's web server. **Research** is the collection of pages about research projects at the departmental server. For each site, IndexFinder processed a large collection of server logs and produced candidate index pages for the webmaster. Competing algorithms were also used to generate index pages, and human-generated index pages were considered as well. All index pages were evaluated with respect to how many of the site's users would benefit from the new index pages (based on what pages they viewed at the site) and how much effort they would be saved. Additionally, index pages were added to **Music Machines** and monitored to see which pages were actually used most. In general, IndexFinder's index pages outperformed those generated by all other algorithms and were competitive with human-authored index pages.

1.3 Design Decisions in IndexFinder

Our approach to creating adaptive web sites was motivated by a number of considerations. We here discuss two that we consider of particular importance and interest: the distinction between customization and transformation, and non-destructive adaptations.

1.3.1 Customization vs. Transformation

Most work on adaptive web sites, whether it be in academic departments, industrial research labs, or commercial endeavors, has been on *customization* — the adaptation of site presentation to individual users based on their individual tastes and requirements. Examples abound: most *web portals* allow users to define home page customizations such as local weather, favorite stock quotes, and news topics of interest; *web surfing assistants* learn a model of a user's interests from the pages she browses and suggest pages accordingly; *ecommerce sites* develop models of customers' buying habits over time and recommend products.

Customization can be very useful; it allows information to be selected and targeted so

[1] at `http://machines.hyperreal.org`

that, ideally, each user sees only what she wants to see. A customization system works by forming a *model* of each user — some representation of her interests and requirements — in order to adapt to her needs. A model must either be solicited from the user directly or learned from her behavior over time. Soliciting a model (essentially asking questions such as "Are you a baseball fan?" or "What city do you live in?") lets the site build a detailed, focused profile of the user, if she is willing to take the time to answer questions. Learning from behavior allows the site to build a profile without troubling the user; with time, a learned model can be an accurate reflection of the user's interests and preferences. In either case, to protect privacy, no information should be collected without the user's consent.

In contrast, *transformation* is the adaptation of the site itself for all users. Rather than constructing a model of each user, a transformation system analyzes the behavior of users en masse and makes changes to the site intended to benefit all. Because no individual models are maintained, users are not asked to input information, and privacy issues are less critical. In a transformation system, a site with millions of users can leverage that massive amount of data in making improvements for all, rather than using only each individual's own history. A "top ten list" of the most popular pages at a site is a simple transformation; the list is available to all users (including first-time visitors), and it is based on data collected from all users.

Transformation, however, lacks the central advantage of customization: the ability to fine-tune web pages for the individual. Fortunately, customization and transformation are the endpoints of a spectrum. A site that customizes based on a small set of user types and then classifies each user into a type, for example, is a combination of transformation and customization. Classifying each user in this manner is easier than full customization and would not require as much data collection, and new users could benefit from adaptations as soon as they were classified. The two approaches may also be used side-by-side on a web site. A web portal might, for example, create a customized news page for each user but also maintain page shortcuts and a list of popular links based on data collected from all users.

One of the main goals of the IndexFinder project has been to build a system which would improve web sites for *all* visitors while requiring minimal additional work by the users or the webmaster. Almost all serious research and development on adaptive web sites, however, was on customization, an approach we found unsuited to this goal. Accordingly, we have taken a purely transformation approach with IndexFinder: the system learns from the logs of *all visits* and the index pages IndexFinder creates are available to all users.

1.3.2 Non-Destructive Adaptation

Learning is, by its nature, a hit-and-miss proposition; an automated computer system that analyzes data, learns from it, and produces results will necessarily sometimes make mistakes. A human webmaster would be justified in being nervous about letting a computer program make free with her carefully designed web site. Accordingly, it may be a good idea to limit the range of adaptations in order to contain mistakes.

One way to do this is to keep the human webmaster firmly in the loop. Our model of an adaptive web site is based on a web management assistant; the assistant's job is essentially to examine data and make suggestions to the webmaster, who has the final say.

Another way to contain possible damage is to focus on *non-destructive transformations*: changes to the site that leave existing structure intact. Non-destructive transformations may add information or hyperlinks to the site, or overlay different site organization or "views" of the data, but must preserve what is there so that users can always find their favorite pages. We may add links but not remove them, create pages but not destroy them, add new structures but not scramble existing ones.

The user interface of the site must also be preserved. Whatever adaptations are used, they should be presented in such a way as to minimize user confusion. An ever-changing site would be extremely confusing to users; adaptations should be easy to understand and/or somewhat predictable in their form. Nondestructive adaptations such as highlighting links with color or graphics can be effective but unobtrusive. Alternatively, adaptations could be limited to certain areas of the site (e.g., a "top ten pages" list on the site's front page) or of each page (e.g., useful links added to the navigation bar on each page). For example, links to the index pages IndexFinder created for the **Music Machines** site were placed at the top of each page and presented as the suggestions of an agent named Winky. Figure 1.3 shows a typical **Music Machines** page. The top of the page is the navigation bar which appears on every page; Winky is represented by the triangular face in the navigation bar, and his suggestions are in the top right. That area of each page was always filled with Winky's suggestions so users would not be too surprised if those links changed regularly. The webmaster always has the final say in what links may be added, but Winky is used in the site's user interface to convey the idea that those links are changeable and automatically generated.

We believe an adaptive web site will not appeal to users and webmaster — or even be useable — unless its basic integrity is preserved through all adaptations. Therefore, the use of non-destructive transformations and a division of labor between the system and the webmaster are key elements of the IndexFinder design. IndexFinder's index pages are nondestructive, as the system adds only *new* pages to the site; it never removes any page or alters the content of any existing page. Also, since IndexFinder is a subordinate assistant to the webmaster, it can do no harm not specifically approved by a human being.

1.4 Contributions

When we started work on IndexFinder in late 1996, there was little existing research on adaptive web sites, and most of it was concerned with customization-oriented approaches. Our 1997 IJCAI challenge paper [38] was one of the first efforts to categorize various kinds of research and to suggest new approaches (e.g., transformation) for future work. Some of the key contributions of this research, therefore, have been in defining the problem and categorizing kinds of adaptive web sites. Part of this effort has also been exploring the key design issues (such as those discussed above) and defining the tradeoffs between various methods.

The approach taken in developing IndexFinder falls naturally into the realm of data mining. However, the problem has some significant differences from previously-studied problems in data mining. Accordingly, another major contribution of this research has been the definition of new data mining problems (cluster mining and conceptual cluster mining), the adaptation of existing algorithms to these problems, and the development of new algorithms suited to these tasks. *Cluster mining* is a variation of traditional clustering;

whereas clustering algorithms partition an entire space of objects, finding the set of clusters that constructs the best partition, cluster mining algorithms search the space for high-quality individual clusters. Similarly, *conceptual cluster mining* is a variation of traditional conceptual clustering that mines a space of objects for clusters of objects that correspond to well-defined concepts. This effort has also led to insights into the relationships between various kinds of data mining algorithms.

Developing effective adaptive web sites is not simply a matter of developing effective data mining algorithms; web sites are meant to be used, and the user interface is a key aspect of the site. This work has led to insights about the importance of user interface design to this kind of research. Although these insights have not been formally tested, we have identified and conformed to several UI design principles in the development of IndexFinder.

Finally, this research has resulted in the development of a system that has been used to generate useful adaptations on a web site used by hundreds of people every day. The techniques described could be used to advantage by other webmasters.

1.5 Organization

Chapter 2 discusses related and previous work. We discuss other research on adaptive web sites, as well as the work in statistical and conceptual techniques upon which our work is built.

Chapter 3 provides a formal problem definition, makes our assumptions explicit, and presents and motivates our approach. We also describe a preliminary proof-of-concept experiment and formally define the problem addressed in the remainder of the thesis.

The IndexFinder algorithm sketched above has two key steps that were the focus of most of this research: finding sets of pages that often occur together, and finding the topic that most closely ties together each set. The first of these is covered in Chapter 4, which defines the statistical cluster mining problem, describes the PageGather algorithm, and presents an experimental evaluation of PageGather. Topic-finding is covered in chapter 5, which defines the conceptual cluster mining problem, describes the IndexFinder algorithm, and presents an experimental evaluation of IndexFinder.

Chapter 6 summarizes the contributions of this research and discusses future directions for exploration.

Figure 1.3: A typical page at **Music Machines**, the main testbed for IndexFinder.

Chapter 2

RELATED WORK

This chapter examines previous work of relevance to this dissertation. This work falls into two broad categories: previous work upon which our own algorithms are built (divided into statistical and conceptual data mining techniques) and related work in adaptive web sites. We discuss each of these three classes in turn.

2.1 Statistical Data Mining Techniques

The fundamental task addressed in this thesis 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. The statistical techniques we describe share a basic approach: objects in the space are described according to certain attributes (e.g., with a vector of values) and grouped together based on their similarity (e.g., the angle between their descriptive vectors or their co-occurrence frequency).

2.1.1 Statistical Clustering

There are two types of statistical clustering: model-based and distance-based. In model-based clustering (see, for example, [11, 10]), clusters are represented by Markov models; objects are assigned probabilistically to clusters and then assumed to obey the models. Model-based clustering is a powerful technique, but we will not discuss it here.

Distance-based clustering (see [52, 41, 54]) 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, text documents are typically represented as word vectors, where each vector position records how often a particular word occurs in the document. The dimensionality of this space is the number of distinct words in the language. 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) [52], which is popular for its high-quality output, and K-Means clustering [26], which is particularly fast.

A well-known application of clustering algorithms to information browsing is found in Scatter/Gather [14]. 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.

2.1.2 Frequent Set Algorithms

Agrawal et al. ([2]) first introduced the problem of mining *association rules* in massive collections of *market basket* data. Market basket data is a collection of transactions, each of which contains some set of items. For example, the data may record purchases at a grocery store; the available items are all the products sold at the store, and each basket is the set of items purchased by a customer at a particular time. An association rule connects sets of items that often occur together in basket data, as in the example "90% of transactions that include bread and butter also include milk" ([1]). This rule might also be written $(BREAD, BUTTER) \Longrightarrow (MILK)$. In [3], Agrawal and Srikant formally define the market basket problem as follows:

> Let $I = i_1, i_2, ..., i_m$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. Associated with each transaction is a unique identifier, called its *TID*. We say that a transaction T *contains* X, a set of some items in I, if $X \subseteq T$. An *association rule* is an implication of the form $X \Longrightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$. The rule $X \Longrightarrow Y$ holds in the transaction set D with *confidence* c if $c\%$ of transactions in D that contain X also contain Y. The rule $X \Longrightarrow Y$ has *support* s in the transaction set D if $s\%$ of transactions in D contain $X \cup Y$.

Agrawal and Srikant divide the market basket problem into two steps:

1. Find all sets of items that have transaction support above a specified minimum support (frequent itemsets).

2. Use the frequent itemsets to generate association rules with at least a specified minimum confidence.

As algorithms for step (2) are fairly straightforward, most research in this area (see, for example, [2, 45, 51]) has focused on efficient algorithms to find frequent itemsets in large collections of market basket data. We refer to such algorithms as *frequent set algorithms.*

The basic frequent set algorithm is Apriori [3]. The key intuition behind Apriori is that any subset of a frequent itemset must itself be frequent, since the subset appears in the transactions at least as often as the larger itemset. Apriori runs in stages, computing the k-itemsets at each stage, where the k-itemsets are all frequent sets of size k, and k is increased with each iteration. At each step, the frequent sets from the previous step are combined to form candidate itemsets; the candidates without minimum support are then winnowed out. Apriori's advantage over other frequent set algorithms is that it generates far fewer candidate itemsets and is therefore significantly faster.

In the web domain, we consider a visit to the web site to be the market basket, and each page visited an item in the basket. In this domain, a frequent set is a set of pages that are often viewed together by visitors to the site.

Sequence Mining [27, 4] is the extension of frequent set mining to the time domain. Whereas market baskets are essentially unordered sets of items, sequences are ordered sets; in this case, the goal is to find common sequences of events and produce rules that predict future events based on what has happened recently. The application to the web domain is clear: user sessions can be viewed as ordered sequences of pages, where each pageview is an event. If we can predict future pageviews based on what the user has done so far, we can pre-cache those pages or even proactively offer them to the user.

2.2 Conceptual Techniques

As described, statistical techniques group objects together based on their similarity. In conceptual techniques, more emphasis is placed on the *language* used to describe the objects and the *structure* of the groupings created. Sets of objects are allowed only when the set corresponds to a concept expressible in the description language. This section presents two basic conceptual techniques: conceptual clustering and concept learning.

2.2.1 Conceptual Clustering

Conceptual clustering algorithms (see [28]), like their statistical counterparts, partition a data collection into clusters of similar objects. In a conceptual clustering problem (see Figure 2.1), 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 [28], probabilistic descriptions as in [17, 18], or complex and cognitively inspired as in [20]. Perhaps the most widely used conceptual clustering algorithm is Fisher's COBWEB [17, 18]. COBWEB constructs a hierarchical partition of an object space based on conceptual structure found among the objects.

2.2.2 Concept Learning

The goal of a *concept learning* algorithm (see [30]) is to find a conceptual description of a set of objects from a data collection. For our problem, we are interested in human-readable output, so we limit discussion to techniques that produce conceptual descriptions,

Given:

- A data collection D
- A pairwise similarity measure m defined over D
- A conceptual language L for describing elements of D
- A description in L of each object in D.

Output: a set of clusters c_i such that:

- Each c_i is a subset of D.
- Every object in D is a member of exactly one c_i.
- The pairwise similarity between members of a cluster tends to be higher than between members of different clusters.
- c_i corresponds to a concept expressible in L, and includes exactly those objects belonging to that concept.

Figure 2.1: The Conceptual Clustering Problem.

excluding neural nets and other classification algorithms that do not produce comprehensible descriptions. In contrast to conceptual clustering, concept learning requires a set of *labeled examples* as input. Objects in the space are marked as either positive examples of the target concept or negative examples. The goal is to find the concept that fits the positive examples. Concept learning algorithms include decision-oriented (e.g., [39]) and rule-oriented (e.g., [46, 13]) approaches. 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.

2.2.3 Hybrid Approaches

Mirkin [29] 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 in the IndexFinder algorithm, a concept learning algorithm of this kind can be part of an effective solution to the conceptual cluster mining problem.

2.3 Adaptive Web Sites

The explosion of the web has been followed by an explosion in web-oriented computer science research. The area of adaptive web sites, in particular, has seen a variety of approaches. We now examine instances of the different approaches suggested in Section 1.3.

2.3.1 Customization

Customization is the adaptation of the site to each individual user. Each user sees a different presentation of the site based on her interests. Customization is the most common form of adaptivity on the web; it provides direct benefit to users and encourages return traffic.

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[1] [22] 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[2] [16] focuses on dynamic customization based on users' needs and tastes. As with the WebWatcher, AVANTI relies partly on users providing information about themselves when they enter the site. Based on what it knows about the user, AVANTI attempts to predict both the user's eventual goal and her likely next step. AVANTI will prominently present links leading directly to pages it thinks a user will want to see. Additionally, AVANTI will highlight links that accord with the user's interests. 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. [55] describe another kind of automatic customization approach. They per-

---

[1] `http://www.cs.cmu.edu/~webwatcher/`

[2] `http://zeus.gmd.de/projects/avanti.html`

form 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.

Pazzani and Billsus [34, 35] discuss a different kind of adaptive web site using what they call adaptive web site agents. Such an agent adapts its behavior to a specific user and recommends documents to that user, based on the user's interests, document similarity, popularity of pages, and the navigation patterns of other users. These agents are resident at the site and customize themselves to individual users. The authors argue that these agents can make use of multiple strategies for adaptation, including those used in this thesis and other related work.

Anderson, Domingos, and Weld [6] have studied the problem of adapting web sites to the needs of those browsing from wireless handheld devices. The limited screen real estate and network bandwidth of such devices raises special problems in presenting web content. The authors propose an approach based on user modeling and expected utility. Visitors to a site are modeled in terms of their past browsing behavior and available demographic information; the model records both links followed and keywords found on pages viewed. Expected utility is computed as a function of the likely usefulness of a given page to a particular user and the probability of the user finding the page, given probabilities of scrolling in the browser and following links. Their system PROTEUS can add shortcut links (similar to those described in Section 3.5.2, but customized to individual users), rearrange blocks of content, or "elide" content (move it off-page and provide a link to it). In [5], Anderson et al. describe an algorithm to create shortcuts based on a more sophisticated model of expected benefit than the one described in Section 3.5.2. They describe several models, all based on computing probable future pages based on the sequence of pages a user has recently viewed. Their algorithms could be used to generate either individually customized shortcuts as in [6] or shortcuts suitable for classes of users as in [22].

2.3.2 Transformation

Transformation is the adaptation of the entire site in response to aggregate user interaction patterns. Transformation does not provide the sense of individual service that customization does, but can improve the site for new users as well as repeat visitors.

A simple form of transformation is seen in sites that maintain a list of the most popular pages. A "top ten list" can be computed by counting how many hits each page gets from all visitors; placed on the site's front page, this list provides a set of shortcuts to new visitors unsure of where to go.

Footprints [53] takes a more sophisticated transformational 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.

2.3.3 Collaborative Filtering

Lying somewhere between customization and transformation is *collaborative filtering*, which uses information drawn from many people to make customized recommendations for each individual. 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. The set of people interested in that particular book is presumed to share some tastes, and recommendations are made based on other books those people were interested in.

Firefly [47] uses a more individualized form of collaborative filtering in which members may rate hundreds of CDs or movies, building up a very detailed personal profile; Firefly then compares this profile with those of other members to make new recommendations. The GroupLens project [42, 19] 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 [8] 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.

2.3.4 Meta-Information

A web site's ability to adapt can be enhanced by providing it with *meta-information*: information about its content, structure, and organization. Although not a specific technique for creating adaptivity, meta-information makes it possible for an automated system to reason about what changes can or should be made to the site, how new pages might be organized, and what elements of the site should or should not be connected together. Good meta-information can improve the comprehensibility of an adaptive site and keep it from becoming a mess.

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 [23], STRUDEL [15], and SHOE [25]). One example of this approach is the STRUDEL web-site management system [15] 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.

The page descriptions discussed in Chapter 5 are another form of meta-information. Automatic classification systems have been applied to the problem of automatically generating document descriptions. One example is found in [21], which classifies HTML pages according to Dewey Decimal Classification (DDC). DDC classes also have associated keywords which can be used to further describe the documents. IndexFinder could be adapted to use DDC classes as a concept language, facilitating automatic description of pages at a site with minimal additional effort by the webmaster.

2.3.5 Client-Side Customization

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. These kinds of projects share similarities with work in adaptive web sites and may produce applicable techniques.

The AiA project [7, 43] 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 [24] 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. Personal Webwatcher [31] is a similar system, which observes the user's browsing behavior and builds up a profile of her interests, represented as a naive Bayesian classifier. Syskill and Webert [36, 33] is a similar system that allows a user to rate web pages and constructs a profile for each of the user's distinct interests.

2.3.6 Web Mining

Web Mining is the application of data mining techniques to web-related domains, particularly the mining of information from server access logs. In recent years a variety of research has explored this area, particularly the ways that standard data mining techniques can be adapted or generalized for the web domain.

Various researchers have explored the problem of adapting the classic frequent set and sequence mining algorithms (see Section 2.1.2) to the web mining domain. The obvious translation to the web mining domain is to view each user visit as a single shopping cart or a sequence of events. However, these navigation trails often contain far more items than the average shopping cart, and algorithms optimized for the cart problem may not be efficient for web mining.

Pei et al. [37] explore a more complex representation for web log data. Transaction data is generally represented as a flat list of items; because web sites are frequently hierarchical and web navigation involves backtracking and movement up and down these hierarchies, web log data can be stored in a tree-based data structure they call a Web access pattern tree (or WAP-tree). Their WAP-mine algorithm looks for frequent sequences in these tree structures and is at least as efficient as classical data-mining algorithms adapted to the web.

Similarly, Borges and Levene [9] have also generalized mining for association rules using a new representation for log data. They represent navigation data for a site as a probabilistic grammar. In this representation, highly probable strings in the grammar correspond to preferred navigation trails at the site. They present an algorithm to efficiently mine such trails. Their representation is also based on an N-gram model of web navigation, in which only the most recently viewed N pages effect the probability of the next page to be seen.

Chen, Park, and Yu [12] cast the similar problem of finding frequently occurring sequences into the web mining domain. Web navigations, however, can represent long sequences of pages, so the authors choose to partition their visits into smaller units — chains of *forward references*. A forward reference is a page the user has not yet visited; when the user backtracks to a previously-seen page, the chain is broken. Backward references are assumed to be for ease of navigation rather than representing interesting browsing behavior, and so sequences are partitioned at these backward references — the authors assume that each chain of forward references represents a single instance of a goal-oriented navigation ending in either success or failure. These chains of forward references are the units in which they search for common patterns.

Spiliopoulou, Faulstich, and others have formalized the notion of a web navigation pattern and developed WUM, the Web Usage Miner [48, 49, 50]. Spiliopoulou et al. view users as having both interests and navigation habits. For example, two users may both be interested in classical music, but one may prefer to browse a hierarchy while the other favors a search engine. WUM is a web mining system for discovering usage patterns in access data. One of WUM's advantageous features is its MINT mining language. MINT allows the webmaster to provide predicates describing criteria to be used in deciding what discoveries are interesting. Thus, WUM is not limited to simply finding statistically dominant patterns or the most frequent patterns, but can be guided by the webmaster's interests. Spiliopoulou has also addressed the more general question of how to evaluate whether a web site is meeting its design goals based on web usage mining in the Communications of the ACM [48]. Spiliopoulou argues, as we do, for an evaluation methodology based on web usage mining. She describes a multi-step process to evaluate a site that uses sequence mining to identify common user patterns and then compares those with the site designers' expectations. One example of this approach, for an e-commerce site, is to measure *conversion efficiency*: how many visitors actually go on to buy something.

Zaiane et al. [56] present WebLogMiner, another web mining tool for discovering user navigation patterns in web server logs. WebLogMiner's main feature is its construction of a data cube representation and use of on-line analytical processing (OLAP) to manipulate the data cube. A data cube is simply a way of viewing multidimensional data where each dimension is some attribute of the data (e.g., for access log data: URL requested, file type, time of request, origin of request, and so on). The data cube allows the application of OLAP operations to specialize or generalize data or slice the data range into subclasses. The authors discuss how to perform standard data mining operations (e.g. develop classification rules, find association rules, summarize data) on this representation.

Cadez et al. [10] describe a technique that uses model-based clustering to model classes of users who visit the MSNBC web site. Visitors are probabilistically assigned to clusters, each of which is represented by a Markov model giving transition probabilities from one section of the site (e.g. politics, sports, business) to another. These clusters of users are presented to the webmaster with a visual representation of common browsing behaviors.

Chapter 3

PROBLEM DEFINITION

In this chapter, we formally define the adaptive web site problem. In doing so, we provide formal definitions and descriptions of important terms used throughout the thesis. We first define what we mean by a web site and how user navigation is represented. Next, we explore how we may evaluate the quality of a site and decide when a site has been improved. We then discuss what it means to adapt a site and define adaptation as search in the space of possible web sites. Finally, we assemble these definitions into a formalization of the adaptive web site problem.

3.1 Web Site

A web site is commonly viewed as a directed graph in which nodes represent pages and arcs represent hyperlinks. A visitor's path through the site may be represented by the sequence of nodes visited and arcs traversed. Note that which page is served to a visitor depends solely on what page she was on and which link she chose; different visitors following the same links will always see the same result. Similarly, whatever a user's path through the site, the same link from the same page will always lead to the same place.

Adaptive web sites, however, may customize their presentation to different users or reorganize the information based on what a user has already seen. Therefore, a model of an adaptive web site needs to account for both individualized user models and user histories. A *user model* is some representation of a user's interests. A user model may be a list of pages a user has viewed in the past, a set of positive and negative ratings for pages that the user has supplied, some explicit description of the user's tastes, or other information. We do not define what the model is, only that an adaptive site may maintain a model of each user (updating it automatically or with the user's input) and use it to decide what information to display. A *user history* is simply a record of what information the user has seen in her current visit to the site and what actions she has taken. Minimally, this could just be an ordered list of HTML pages viewed, but the history could also include links clicked upon, forms filled out, pages bookmarked or printed out, time spent on each page, and so on. An adaptive site may rearrange its content, combine various elements on one page, reorder content, and much more.

An adaptive web site, then, has any or all of the following components:

- A collection of units of content (documents, web pages, text, images, etc.).
- Connections between different units of content.
- A model of each visitor to the site.
- A history for each visitor to the site.
- A function from user actions to web documents returned to the user.

The site presents its content to visitors in the form of web pages (or other documents) and makes the connections available in the form of hyperlinks. How these pages and links are presented may be influenced by the model and history. For example, a relatively simple adaptive site may store its content in the form of HTML pages with hyperlinks that are then modified at runtime to add personalized elements based on the user model.

Suppose a user is at a web page and selects a link; the interaction might be handled as shown in Figure 3.1. If the site is a simple static one, no user model or history information need be maintained. The typical customizable portal site will maintain a user model listing the user's stock picks, favorite sports teams, and so on; history information may be ignored entirely. A "tour guide" type of adaptive site might use the history to recommend pages the user has not yet seen.

3.2 The Quality of a Site

Before we can improve web sites, we need to have a way of evaluating how good a site is, and whether a change to the site is actually an improvement. We need this evaluation

1. The web server looks up the user model for the user.

2. The link just selected is added to the user's history and the user model may be updated.

3. Based on the current page and the link selected, as well as the user model and the history, the server presents an HTML page to the user.

4. If the user clicks on a link, the process is repeated.

Figure 3.1: Basic interaction between an adaptive web site and a user.

both so that an adaptive site can decide when a change should be made, and so that we, as scientists, can evaluate the success of our adaptive sites. In this section, we will formalize an intuitive measure of site quality, show that this measure requires information to which we do not have access, and present approximations suited to our problem.

3.2.1 True Site Quality

While there are many different kinds of web sites, and many ways of evaluating them, our work has implicitly assumed a web site containing a large variety of information, with visitors seeking certain pages that meet their interests. A good web site, then, will be one at which visitors have little trouble finding the pages they want. We do not assume that visitors necessarily seek a specific piece of information or that they have explicit, conscious goals; instead, we assume that users have general interests that match some of the site's pages, and that the more visitors find the more of these pages, the better the site is.

We assume that, for each user, there is a set of pages at the site of particular interest to that user. A high-quality site is one in which the typical user easily finds many of those pages. In other words, the quality of a site is a function of the following variables:

- **Recall**: how many pages the typical user sees of those at the site of interest to her.

- **Precision**: how many of the pages the typical user sees in her visit are actually of interest to her.

- **Effort**: how much work the typical user must exert to find the pages she sees.

Precision and recall are measured in terms of the proportion of pages. Effort is a function of the number of clicks visitors have to make, the amount of time spent reading and scrolling, and so on. Precision and effort are closely connected; a site with low precision, where one must search through many pages to find the nuggets of interest, requires a great deal of effort. In the rest of this dissertation, we will focus on recall and effort as the primary indicators of site quality. We now derive more precise mathematical models of recall and effort which could be combined into a quality measure.

Recall

Recall is the proportion of pages of interest at a site that a user actually manages to find. To measure this value, we must know which pages at the site the user would find interesting and which she actually found. The recall will be the proportion of the latter to the former. We will find the recall for each user, and then average over users. Users may make multiple visits to a site; we will count each visit as a separate user. This provides a de facto weighting on the importance of each user; those who visit often are given more weight in rating the site's quality.

Let D_s be the set of documents at a web site s. Let U_s be the set of all users who visit s. Let u be a user and D_u be the set of documents he downloads when he visits the site. Let I_u represent the (very large) set of all possible documents of interest to u. The set of documents at s of interest to some user u is simply the intersection between u's interests and the documents at the site: $I_u \cap D_s$. Similarly, the documents of interest that u actually sees in her visit is the intersection $I_u \cap D_u$. The site's recall with respect to u is the proportion of these two values:

$$R_u = \frac{I_u \cap D_u}{I_u \cap D_s}$$

In other words, the recall for a user is the number of interesting documents the user saw divided by the number of interesting documents at the site. Perfect recall is when the user sees every document of interest to her. To evaluate recall for the site overall we average over the set U_s of all users who visit s:

$$R_s = \frac{\sum_{u \in U_s} R_u}{|U_s|}$$

Recall may also be viewed as the site's "success rate" — perfect recall means every user is shown every page that could be of interest to him. We may also view recall as a distribution over the possible visitors to the site of the expected recall experienced by each visitor.

Effort

What do we mean by "effort"? Effort is the amount of work a user has to do to see the pages she sees at the site. Clicks are typically taken as a simple measure of effort; a user who clicks on six links has exerted twice the effort of one who clicks on only three. Is this really true? Consider a typical site with a large collection of information structured (approximately) as a tree hierarchy. At **Music Machines**, for example, information is structured in a three-level tree where the first level is divided by manufacturer, the second by specific model, and the third by file type. At the minimum, then, a user must click on three links to reach a document such as a sound file or instrument review. **Music Machines** could be restructured, however, so that all files are linked directly from the front page. Every file could now be found by clicking on one link, but clearly the effort of scrolling through thousands of links on a single page would more than outweigh the clicks saved. Finding interesting information on one very very long page requires a great deal of effort; it is hard work reading through tens or hundreds of links to find the handful of interest. The effort involved in navigating a web site should be a function of not only the number of clicks one must make, but also of the difficulty of deciding, on each page, which is the best link to follow.

Each link at the site may be viewed as requiring an amount of effort to find. A link at the top of a page takes less effort than one at the bottom; a link in boldface with a blinking box around it is easier to find than one in small type and unobtrusive colors; a single link on a page containing ten links requires less effort to find than one on a page containing a thousand. We will not define a function to determine effort, only assume that it is a function from a link and the page it is on to a real-valued number representing the effort required.

In addition to the terms defined above, let L_s be the set of links at the web site s. Let L_u be the set of links user u traverses during her visit to the site. Let e_l be the effort associated with the link l. We may then define the effort of one user as:

$$E_u = \sum_{l \in L_u} e_l$$

In other words, the amount of effort a user exerts during a visit is the sum of the effort she exerts finding each link she traverses. As with recall, the site's overall effort rating should be the average over all users:

$$E_s = \frac{\sum_{u \in U_s} E_u}{|U_s|}$$

Effort is how much difficulty users have in getting around the site — a low-effort site is one at which users move quickly and easily to their desired pages with a minimum of clicks or confusion. We may also view effort as a distribution over the possible visitors to the site of the expected effort exerted by each visitor.

Quality

The quality of a site increases with recall and decreases with effort. Therefore, we represent site quality as the weighted difference:

$$Q_s = \alpha R_s - \beta E_s$$

where α and β are weights controlling the relative importance of recall and effort. The ideal site is one at which every user sees all the pages he finds interesting with no effort at all.

3.2.2 Using the Information Available

Our evaluation of a web site must be based on whatever information we have about how people interact with it. Our main source of this data is the web server's access log. An access log contains one entry for each "hit" — a request to the server for a particular page. Each request lists at least the origin (IP address) of the request, the URL requested, and

```
24hrlab-214.sfsu.edu - - [21/Nov/1996:00:01:05 -0800] "GET /home/jones/collectors.html HTTP/1.0" 200 13119

24hrlab-214.sfsu.edu - - [21/Nov/1996:00:01:06 -0800] "GET /home/jones/madewithmac.gif HTTP/1.0" 200 855

24hrlab-214.sfsu.edu - - [21/Nov/1996:00:01:06 -0800] "GET /home/jones/gustop2.gif HTTP/1.0" 200 25460

x67-122.ejack.umn.edu - - [21/Nov/1996:00:01:08 -0800] "GET /home/rich/aircrafts.html HTTP/1.0" 404 617

x67-122.ejack.umn.edu - - [21/Nov/1996:00:01:08 -0800] "GET /general/info.gif HTTP/1.0" 200 331

203.147.0.10 - - [21/Nov/1996:00:01:09 -0800] "GET /home/smith/kitty.html HTTP/1.0" 200 5160

24hrlab-214.sfsu.edu - - [21/Nov/1996:00:01:10 -0800] "GET /home/jones/thumbnails/awing-bo.gif HTTP/1.0" 200 5117
```

Figure 3.2: Typical user access logs, these from a computer science web site.

the time of the request (see Figure 3.2). Every web server produces an access log of this kind, many with far more information about each hit. Access logs may be organized in many ways, but one form we will find particularly useful is to group the hits into *visits*. A visit is an ordered sequence of pages accessed by a single user in a single session. Visits can be computed directly from an access log for any site with fairly high accuracy; with the use of cookies and other techniques, they provide precise data about each user's interaction with the site.

For significantly more insightful information — e.g., how well a user liked a particular page, whether he found what he was looking for, or whether he read the page at all — the site must use less automatic means. Useability tests, feedback forms, and asking users to rate pages are all ways of gathering additional information. One such approach is described in [48], in which users are given specific tasks to perform at a site, and their actions are monitored to evaluate how well the site facilitates those tasks. Spiliopoulou notes, however, that this methodology comes with a high price — test subjects must be recruited, they must be representative of the target body of users, and we must establish an environment in which to run experiments. Furthermore, this must be done every time a change to the site is evaluated. If evaluations are voluntarily performed at the site by actual users, we cannot guarantee that they are representative users. Furthermore, these methods are not automatic in that they require deliberate effort from users, who may decline to participate, make mistakes, or even lie. While a web site may benefit from the use of this kind of data, we believe it unrealistic to base an automated adaptive system on it. We will therefore assume only data that can be computed (or inferred) from access logs.[1]

Most of the data used in the quality measure defined above — pages viewed, links traversed, etc. — can be derived either from access logs or from the site itself. Effort can be approximated by the number of links followed, and how far down the page those links are located. However, one key piece of data cannot: I_u, the set of all possible pages of interest to a particular user. No amount of log information can tell us for certain what a user's interests are, which pages she enjoyed, or which pages she would have enjoyed had she seen them. We will therefore have to approximate user interest information. We do this by assuming that, generally, users view pages they find interesting. In other words, the pages in a user's visit to the site are an approximation of her interests. We will not, however, assume that every page is interesting, as users may make mistakes or traverse many uninteresting pages to get to one good one. This assumption allows us, for example, to infer a model of a user's interests by looking for common themes in the pages she visits and to assume that other "similar" pages would also be of interest. This assumption was key in making the visit-coherence assumption (discussed in Sections 1.2 and 3.5.3).

3.2.3 Approximating Quality by Evaluating Improvements

We do not have available the data needed to evaluate the quality measure described above. However, in our problem domain, we do not need an independent measure of a site's quality; our goal is not really to evaluate a web site in isolation but to decide when a *change* to a site is an improvement. To do so, we must compare the site "before" and "after" or simply evaluate the effect a certain change has on the site's quality. This evaluation is easier than determining the site's quality in the abstract, and can reasonably be approximated with access log data.

Consider an adaptive site weighing two possible adaptations. One, say, adds a couple

[1] Note that standard access logs do not include unique identifiers for each visitor; this limitation can be overcome with the use of cookies or visitor-tracking software. In practice, we user IP numbers as user identifiers; this issue is discussed further in Section 4.3.1.

of useful links to the top of every page at the site. Doing this will prove helpful to many users who will click through the links and find what they want slightly faster. On the other hand, there are some users who will not benefit, since they have no use for the links. In fact, these users will be slightly inconvenienced, as every page is now slightly more cluttered with extra links — this clutter, in our terminology, increases the effort e_l associated with each link. The second modification creates a new page of links relating to a particular topic. This is very useful for the small subset of users interested in that topic, but has virtually no impact (positive or negative) on the vast majority of users.

On the one hand, a change that slightly helps many; on the other, a change that greatly helps a few — which of these modifications is preferable to the other? Is either preferable to the original site? As this example should make clear, the effect a modification has on a site's quality is dependent on two factors:

- **Impact:** how many people are affected by the change.

- **Benefit:** how much they are affected.

In the above example, the first modification is high impact/low benefit, and the second is low impact/high benefit.

Impact is often easy to measure a posteriori from visit log data; the number of visitors who actually make use of (or are exposed to) the new features can be counted. The logs can also be used to estimate the impact of new features before implementing them, by counting the number of users likely to make use of them; an example of this approach is described in Section 4.4.1. Measuring benefit is more difficult; we can compare the visits of users before and after the change to see who exerts more effort, or use heuristics to conjecture how much effort has been saved.

What is the relationship between recall/effort and impact/benefit? Benefit is essentially measured in terms of improvements in the recall (R_u) and effort (E_u) values for each user who is affected by the adaptations. A shortcut added to a site, for example, would decrease the effort of those visitors who use it, by allowing them to skip over several pages. An index page containing links to everything on a particular topic would increase recall and reduce for any visitor interested in that topic, as she would now be able to find all of the relevant documents in one location. Impact is defined not in terms of recall and effort but as the number of people whose recall or effort values are affected.

3.2.4 Evaluating Quality in Practice

In practice then, we expect an adaptive web site to evaluate the impact and benefit of adaptations based on access log data. Benefit will be measured in terms of improvement in recall or decrease in effort.

An adaptive site may itself use an *implicit* quality measurement — i.e. its algorithms are intended by construction to produce useful adaptations. For example, a site might find *shortcuts* to popular pages that cut down on user effort or *highlight* popular links, making them easier to find. Although the site does not explicitly measure the impact and benefit of each shortcut before adding it, the shortcuts are assumed to be helpful. A researcher evaluating the site's performance can explicitly measure the impact and benefit of the shortcuts to establish that they are indeed an improvement.

IndexFinder, for example, uses an implicit quality measure in generating index pages. IndexFinder's algorithm is designed to find sets of pages that many users will be interested in viewing together. IndexFinder essentially assumes that index pages found this way will provide significant impact and benefit and does not explicitly try to measure the quality of each index page once it is generated. When we evaluate IndexFinder, of course, we evaluate the impact and benefit of its index pages. We perform two basic kinds of experiments with index pages generated by IndexFinder, competing algorithms, and (for comparison) by the webmaster. In the first experiment, we do not place the index pages at the site, but examine a collection of test log data (distinct from the log data used to generate the index pages). The impact is measured by counting the number of users who view pages that are included on the index page. These are the people who, we assume, might have visited the index page had it been available. Benefit is measured by counting the number of pages linked from the index page each user visits. A user who looks at a large number of these pages would have benefited greatly by having them collected together on one index page. In the second experiment, the generated index pages are placed at the site, and we count both visits to each index page and clicks on the pages' links. In this experiment, the impact is the number of users who actually visit an index page. The benefit is the number of links a visitor actually clicks on. As described in Section 1.3.2, the changes to the site are designed to minimize negative effects: the only change to existing pages is a small number of links added to the navigation bar. Accordingly, we do not explicitly measure negative benefit. In Chapters 4 and 5, we present these experiments and evaluate IndexFinder and competing algorithms with respect to impact and benefit.

Another way to view impact and benefit is as a distribution over all visitors to the site of the expected benefit experienced by each visitor. Making the assumption that the user who visited the site are a representative sample, we could estimate benefit as the average of the benefits for those users.

3.3 Adaptation as Search

We may 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. A search problem is defined by the search space, the state transitions, and the initial and goal states. We discuss each in turn and draw key distinctions between different approaches to creating adaptive web sites. Viewing adaptation as search is useful for placing this and similar research in the context of past work in computer science.

Definition of a State. A state in our search is a web site. As defined above, a web site consists of content, links, user models, user histories, and a function from user actions to documents. Possible sites may differ from each other in page content, links between pages, or how the model and history effect either. One site may be wholly static, while another may completely reconfigure itself based on what the user does.

Defining State Transitions. A state in the search space is a web site; a state transition, therefore, is an alteration that creates a different site. Many kinds of alterations 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. These transitions may consist of changes to the site's content, to its links between content, or to the way it responds to user actions.

The Initial and Goal States. The initial state is typically an existing web site. The goal states maximize 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 approaches attempt continual improvement, without a well-defined stopping point.

Search. The space of possible web sites is impossibly large and complex; searching for an optimal site in this space is certainly intractable. Instead, most adaptive sites employ a hill climbing approach: start with the site in a fairly good state and make incremental improvements. A space this complex is likely to have many local maxima, and the relatively limited choices of an adaptive site may not be sufficient to move out of a local maximum. In such a situation, it is up to the human webmaster to redesign the site, effectively jumping to another part of the search space.

3.4 The Adaptive Web Site Problem

What precisely is an adaptive web site? Previously, we said an adaptive web site was one that semi-automatically improves its organization and presentation by learning from visitor access patterns. Based on the definitions above, we more precisely define an adaptive site as one that:

1. Changes itself over time, either by updating and utilizing user models or by making changes directly to site content and structure.

2. Explicitly or implicitly uses a quality measure to guide adaptation.

3. Uses only automatically gathered data to guide adaptation.

And a *good* adaptive site is one that measurably improves the site's quality by an independent measure.

3.5 Index Page Synthesis

3.5.1 Desiderata for Adaptive Web Sites

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 cooperate.

2. **Make the web site easier to use for everyone, including first-time users, casual users, etc.** Pure customization has the most benefit for repeat visitors, when the site has time to develop a detailed profile of the user. Because a transformation or hybrid approach makes changes influenced by and available to all users, it can provide immediate improvement for first-time and casual users.

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 *non-destructive 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 non-trivial changes to the web site remain under webmaster control. Initially, we do not assume any annotations on Web pages beyond HTML. However, we also explore site transformations that are possible only with some degree of meta-information.

3.5.2 Simple Transformations

It is possible to devise a number of simple, non-destructive transformations that could improve a site; we briefly discuss several: promotion, highlighting, and linking. We also describe another — shortcutting — in more detail and present a simple experiment.

Promotion makes a link or page easier to find by placing a reference to it closer to the front page of the site (on the front page or a nearby index page) or by moving a link closer to the top of a page. Promotion would decrease effort by making frequently accessed pages easier to find and possibly improve recall by allowing users to find some pages they might otherwise have missed. However, excessive promotion would lead to clutter that increases user effort — like all adaptations, promotion should be selectively practiced. A site could also practice demotion — clearing away links that are rarely used, though this adaptation could be quite destructive.

Highlighting draws attention to an existing link on a page by emphasizing it with fonts, colors, or graphics. Highlighting, like promotion, can make popular pages easier to find and thus decrease effort and improve recall. Also like promotion, excessive highlighting can increase clutter and make the site harder to use.

Linking connects two pages that were previously unconnected by adding new hyperlinks between them. Linking is based on inferring connections between pages based on similarity or correlations in user visits. Linking can drastically improve recall and effort, as it allows users to find groups of related pages of interest. Furthermore, because these links appear only on the two pages involved, clutter is kept to a minimum, though excessive linking could get confusing.

Shortcutting

Shortcutting provides links to users' eventual goals, skipping over the in-between pages. As opposed to linking, which connects similar pages, shortcutting tries only to guess where users will want to go from particular pages and provide the shortcut. We have implemented shortcutting at **Music Machines** and, as reported in [38], found a significant number of visitors used them.

1. Initialize a two-dimensional array of counters, with one row and one column for each page at the web site. Each row represents a web page p, and the counters in that row represent how often users who visit p subsequently visit each other page at the site.

2. For each page p in each visit in the access log, find all pages q that occur (at least two pages) *after* that page in the visit. In the row for page p, increment the counters for each q found.

3. For each page, add hyperlinks to the N pages in its row with the highest counters.

Note that we count only pages that occur at least two pages later in order to avoid creating useless shortcuts to pages that are only one click away.

Figure 3.3: The shortcutting algorithm for generating shortcuts to common user destinations for each page at a web site.

We use a simple method to find shortcuts: for each page p, find the N pages users visit most often after visiting p. The algorithm is shown in Figure 3.3. This transformation can be easily computed nightly and updated on the website. As a preliminary experiment, this algorithm was implemented and put in place at the **Music Machines** web site. For the reasons described in Section 1.3.2, the shortcuts are presented as the "suggestions" of Winky, the friendly site agent. Similar to the suggestions shown in Figure 1.3, the shortcuts were limited to five per page and placed in the navigation section of each page. Shortcuts were page-specific and so differed from page to page. The URLs provided in the shortcuts were specially tagged by adding "`?Shortcut`" after each URL — this did not change the page loaded, but made sure that the access logs would clearly show how often the shortcuts were used. Shortcuts were left in place for one week and the logs were examined. Although we did not formally compute the impact and benefit of the transformations, we found that approximately 15% of *all* hits to the site were made via shortcut links. This is a significant impact on the site's users; and considering that each shortcut taken saved the user at least one whole page (and often more), the benefit is considerable as well.

This was only a proof-of-concept experiment, and the transformation could have been made more effective. For example, shortcuts could be limited to "content" pages. The transformation impacts every user, as the shortcuts appear on every page, but that impact is designed to be minimal. The number more strongly impacted is simply the number of users who use at least one shortcut. Benefit is the summed effort (e_l; see Section 3.2.1) of all the links the user would have had to traverse had the shortcut not been available.[2] Furthermore, because shortcuts are labeled with the title of the target page, we make the simplifying assumption that users click on the shortcut only if they want to see the target page. A more careful study of this transformation would evaluate whether some users experienced negative benefit by being led down the garden path. This simple experiment, however, demonstrated that nondestructive transformations could be feasible and effective for a web site.

[2]When there are multiple paths to the target, the minimum-effort path should be chosen. In general it could be costly to compute the minimal path. If necessary, it can be reasonably approximated either by computing the cost along standard paths in the site (i.e. treating the site's organization as a tree) or by averaging the cost of actual paths followed by users, as found in the access logs (which could be computed at the same time as the data that determines the shortcuts).

3.5.3 Index Page Synthesis

Although shortcutting proved useful, our long-term goal was 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. As described previously, 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 instrument type.

This "comparison shopping" pattern is one which we should be able to automatically detect in the site's visit logs. We look for groups of pages that users often visit together (such as the "keyboards" pages in the example above) and create a new "index page" for each group with links to those pages. To find groups of related pages, we rely on the visit-coherence assumption (see Section 1.2): the pages that a user visits during one session at the site are likely to be closely related. This automatic index page synthesis is a first step toward creating new views of a web site.

We propose a *web management assistant* that monitors the access logs, generates candidate index pages, and presents them to the human webmaster for approval. The assistant should process the daily access logs every night; index pages could be presented to the webmaster every morning or, more likely, every week. This would give the assistant more data to work with before making each suggestion, and make sure that it does not demand too much of the webmaster's attention. Furthermore, at many web sites content changes slowly, and daily suggestions would be redundant.

Given:

- A web site S (a set of HTML and other documents connected by hyperlinks).
- An access log L (a record of page requests made to the site).

Output all index pages P, where P consists of links to a set of pages in S such that:

- Pages in P frequently co-occur in user visits in L (for some definition of "frequently").

Figure 3.4: 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 (see Figure 3.4). As described above, an access log contains one entry for each page requested of the web server, listing 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. Figure 3.4 formalizes this problem definition.

3.6 Assumptions

In describing and formalizing the problem domain, we have made a number of implicit and explicit assumptions. Before describing specific solutions in the next two chapters, we first review these assumptions.

We have made several implicit assumptions about the nature of web sites and user behavior. In general, we assume that users are pursuing some kind of information-gathering task, and that the value of a site to a user lies in how much interesting information she finds there. In other words:

- The web site includes many pages containing a variety of information.

- Each user has (conscious or unconconscious) interests; some pages will be of interest to a user and some will not.

- A good site is one that maximizes pages of interest seen by its users (recall) and minimizes the effort those users must exert (effort).

We have also made implicit assumptions about what can be concluded from a user's history. We do not know what a particular user is actually interested in or whether she found it. We therefore draw conclusions from the evidence at hand: her browsing history. We assume:

- Users view pages they find interesting — in other words, the pages in a user's visit to the site are an approximation of her interests.

- Visit-coherence: a subset of the pages that a user visits during her session at the site is likely be conceptually related.

Finally, in implementing an adaptive system, we have made assumptions to simplify the process of data collection and avoid re-engineering of web sites.

- Each separate visit by a user is counted as a separate user. Frequent users should be weighted in proportion with the frequency of their visits. However, we do not weight users by the amount of time or number of pages they view.

- Each originating machine corresponds to a single visitor to the site.

These assumptions have facilitated our exploration of the problem space. Some of them could be relaxed or discarded; we will revisit them in Section 6.2.

Chapter 4

STATISTICAL CLUSTER MINING

We have defined the index page synthesis problem. As discussed previously (see Section 1.2), page synthesis can be broken down into five subproblems:

1. What is the content (i.e. set of hyperlinks) of the index page?
2. Does it have a coherent topic? What should its title be?
3. How are the hyperlinks on the page ordered?
4. How are the hyperlinks labeled?
5. Is it appropriate to add the page to the site? If so, where?

As stated in Section 1.2, the first two are the focus of our research, the third and fourth are adequately handled with simple rules, and the last is decided by the webmaster. We will here discuss the first problem: what are the contents of the index page? I first describe previous work in data mining which could be applied to this problem. We then define *cluster mining* and discuss why it is suited to this problem. Finally, we present the PageGather cluster mining algorithm for index page synthesis.

4.1 Clustering

Our definition of the index page synthesis problem (see Figure 3.4) states that we are given a large set of objects (web pages at our site) and must find sets of these objects that are "similar" according to some measure (co-occurrence frequency in user visits). This task is very similar to the clustering problem (see Figure 4.1), often confronted in the data mining community.

Statistical clustering (see [52, 41, 54]) 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; in a partition, each object is placed into exactly one cluster — all objects are clustered, and no object appears in more than one cluster.

Given:

- A data collection D
- A pairwise similarity measure m defined over D

Output: a set of clusters c_i such that:

- Each c_i is a subset of D.
- Every object in D is a member of exactly one c_i.
- The pairwise similarity between members of a cluster tends to be higher than between members of different clusters.

Figure 4.1: The Clustering Problem.

Clustering algorithms are numerous, but two common techniques are K-Means clustering [44], which is simple and fast, and Hierarchical Agglomerative Clustering (HAC) [52], which is popular for its high-quality output. K-Means clustering is given a target number k of clusters. The algorithm begins by randomly choosing k objects to be the center of these clusters. On each iteration, all objects are placed with the nearest cluster, then the center of each cluster is recomputed (see Figure 4.2). The number of clusters k must be tuned for a particular problem.

Input: a data collection D, a similarity measure m, target number of clusters k

1. Randomly choose k objects; create a singleton cluster for each.
2. Repeat until quiescence:
   (a) For each object, place it in the closest of these k clusters.
   (b) Recompute the center of each cluster.
3. Return the k clusters.

Figure 4.2: The K-Means Clustering algorithm.

HAC is an agglomerative algorithm: clusters are formed by merging clusters (including singleton clusters) together into larger ones (see Figure 4.3). Every merge is represented in a tree where the merged clusters are the children of the resulting larger cluster. When the algorithm is complete, the tree represents both a hierarchy of clusters and a record of how the objects were merged into successively larger clusters. Merges continue until no two clusters are above a provided threshold in similarity; this threshold typically must be tuned for a particular problem.

4.2 Cluster Mining

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. For index page synthesis, however, we wish to find small highly-coherent clusters but need not classify most of the objects in the space. A cluster mining algorithm, therefore, may be more efficient than a clustering algorithm. Furthermore, requiring the algorithm to place every object in a cluster may result in clusters that are larger and less coherent; there may be many objects in the space that are not particularly similar to any cluster, but as they must be placed somewhere, they will dilute whatever cluster to which they are added. Also, many

Input: a data collection D, a similarity measure m, a similarity threshold t

1. For each object, create a singleton cluster.
2. While the number of clusters exceeds 1 and the similarity between the nearest pair of clusters exceeds a threshold t:
   (a) Find the two clusters i and j that are closest together according to the similarity measure.
   (b) Create a new cluster k which is the union of i and j
   (c) Create a tree node for k, the parent of i and j.
3. Return the remaining set of clusters (and their trees).

Figure 4.3: The Hierarchical Agglomerative Clustering algorithm.

traditional clustering algorithms do not allow clusters to overlap, while we do not mind if objects appear in multiple clusters — after all, a particular page at the site might belong on several different index pages. For example, a page about an inexpensive, American-made electric guitar could be placed on index pages about "electric guitars", "American-made instruments", "inexpensive guitars" and so on.

Cluster *mining* is a variation on traditional clustering that is appropriate for our task. Rather than partitioning the space of objects, we mine the space for coherent, possibly overlapping clusters of highly similar items. We formalize the cluster mining problem in Figure 4.4. The relationship between traditional clustering and cluster mining is parallel to that between classification and data mining described in [46]. 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.

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_i of D of size less than k such that

- Each c_i is highly cohesive with respect to m (e.g., the average pairwise similarity of objects in c_i exceeds some threshold).

Figure 4.4: The Cluster Mining Problem.

4.3 The PageGather Algorithm

The *PageGather algorithm* uses cluster mining to find collections of related pages at a web site, relying on the visit-coherence assumption (see Section 1.2). 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 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. Combine or discard clusters that are too similar.
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 will discuss each step in turn. In Section 4.4, we will present comparisons among various forms of the algorithm and discuss which works best in practice.

4.3.1 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 requests made to the web server. 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. 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. In fact, this is not necessarily the case. A number of 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 uncommon relative to all of a site's traffic; if necessary, however, more accurate logs can be generated using cookies or visitor-tracking software. A series of page views in a day's log from one visitor, ordered by their time-stamps, corresponds to a single session for that visitor. 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 in most cases; if necessary, however, the series of page views could be divided at significant time gaps.

Implementation. Our goal is to create a web management assistant that will constantly examine the site's access logs and periodically (weekly perhaps) suggest new index pages to the webmaster. Accordingly, we require a means of storing and processing our access data that enables both nightly updates and on-demand processing. Processing the access log into visits is, fortunately, quick and easy. Because the access log is sorted by date stamp, we can easily process it in one pass, assigning hits to user visits in order (see the description of access logs and visits in Section 3.2.2). This process is fast on even a large access log and can easily be performed nightly. We implemented it as a simple Perl script which can be run on the site's access log.

4.3.2 Compute the co-occurrence frequencies between pages and create a similarity matrix.

As discussed previously, we rely on the visit-coherence assumption: pages that tend to co-occur in user visits tend to be conceptually related. We wish to find a measure of co-occurrence, then, most likely to bear out this assumption. Implicit in this assumption is the requirement that not only should the pages occur together often, but also that the presence of one page in a user visit should be a strong predictor of the other. In other words, we consider p_1 and p_2 to have a high co-occurrence frequency if the probability of seeing p_2 *when we have already seen* p_1 is high. Similarly, the probability of seeing p_1 given that we have seen p_2 should be high. In fact, we should care more about these probabilities than about a simple count of how often the pages co-occur. After all, two obscure, unpopular pages could prove to be highly associated, and we would like to recognize that they should be linked together, regardless of the fact that they are rarely downloaded.

Therefore, 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 given that she visits p_2 and $P(p_2|p_1)$, the probability of a visitor visiting p_2 given that she visits p_1. The co-occurrence frequency between p_1 and p_2 is the minimum of these values.

Why take the minimum of these two conditional probabilities? Consider the case where p_1 is an extremely popular page — for example, the front page of the site, that most users visit — and p_2 is relatively obscure. Perhaps p_1 gets 1000 hits a day, while p_2 gets only 10. However, because most visitors come through p_1, about 9 of those 10 visit both pages. Therefore, $P(p_1|p_2)$ will be quite high — 0.9. On the other hand, $P(p_2|p_1)$ will be very low — 0.009. Is this a case where we should consider p_1 and p_2 to have a high co-occurrence frequency? Clearly not — the high value of $P(p_1|p_2)$ is entirely a function of p_1's popularity rather than any particular connection between p_1 and p_2. By taking the minimum of the conditional probabilities as our measure of co-occurrence frequency, we ensure that only pages that are good mutual predictors will get high co-occurrence scores.

Our measure is similar to correlation — a measure of the independence of two variables —

but with two key differences. First, whereas our measure ranges from 1 (the pages are highly associated) to 0 (they never co-occur), correlation ranges from 1 (highly correlated) through 0 (uncorrelated) to -1 (negatively correlated). *Negative correlation* is when the presence of one page tends to imply the absence of the other. In our domain, we have no use for negative correlation, as there is little point in linking together such pages. Second, because correlation is a measure of independence, it factors in the probability of the individual pages in a way we do not. For example, two extremely popular pages which, necessarily, co-occur fairly often would not be considered *correlated*, though both conditional probabilities would be high. In our domain, such a pair of pages should be considered frequently co-occurring, however, as we have no objection to discovering *popular* clusters of pages as well as obscure ones. Another possible measure of co-occurrence frequency would be the ratio of the probability of seeing both pages to the probability of seeing either (the number of times they appear together divided by the number of times either one appears). This measure would have the same desirable properties as our metric.

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 consider two pages already linked if there is a link between them or if there exists another page with links to both pages. This is essentially an ad hoc method to prevent us from wasting time discovering connections we already know about. This approach is basically effective but not perfect. For example, one can easily imagine finding a useful new cluster of pages that contains some pages that are already linked together; our ad hoc method will not prevent this discovery, but can make it more difficult. A preferable approach would leave the co-occurrence frequencies alone at this stage of the algorithm and then have a later to step that examines the set of candidate index pages for ones that duplicate what is already available at the web site. This would have the advantages of not ruling out any possible discoveries early on and allowing a more sophisticated notion of when a candidate index page represents a novel discovery. Such an approach might also allow the discovery of links that should be added to existing site index pages or other such modifications. Implementing such an approach would require an additional level of analysis of the site and comparison of candidate pages to existing index pages, and we did not have an opportunity to explore these issues.

Implementation. This step in the algorithm, like the previous one, should be performed on the data nightly. Unfortunately, it is more time consuming than converting the log into visits. Because we may update the similarity matrix more often than we use it to synthesize index pages, we wish to separate these two steps, creating an intermediate data structure that may be stored on disk and accessed at any time. This also gives the benefit that we may run the second half of the algorithm many times with different settings on the same matrix in order to compare results. The matrix is a large data structure — of size $O(N^2)$ if the site contains N pages. As many of the co-occurrence frequencies may be zero, we store the matrix as a list of the non-zero pairs rather than as a full two-dimensional array. A sparse array is typically stored as an array of linked lists, where the array contains one linked list of nonzero cells in each row; there may also be a second array and set of linked lists for the columns, to allow easier access. When storing a sparse array to file as we must do, however, storing a list of pairs is equivalent. For every page, we record how often the page occurs in user visits. Then for every pair of pages that co-occur, we record how often they occur together. This data is sufficient for us to compute $P(p_1|p_2)$ and $P(p_2|p_1)$ and thus our measure of co-occurrence frequency. Note that whether a list of non-zero pairs is more efficient than a two-dimensional array depends on how sparse the matrix is. If data is accumulated over a long time without being flushed, the matrix will evnetually become full, as most pages will eventually co-occur at least once. In this case, an array representation will probably be more efficient. The non-zero pairs could also be stored in a database, which would allow more efficient processing of massive data sets. We chose not to do this as our experimental data sets were relatively small and manageable. Even so, adding a single day's log from **Music Machines** to the matrix could take up to half an hour, mostly due to the time spent reading and writing the matrix to disk. Scaling this process to larger data sets by improving this data structure is one important area for future work.

4.3.3 Create the graph corresponding to the matrix, and find clusters in the graph.

We observed that the similarity matrix could be viewed as a graph, with nodes representing pages and an edge connecting two nodes when the pages have a high mutual co-occurrence. A graph representation enables us to apply graph algorithms to the task of identifying collections of related pages. However, a graph corresponding to the similarity matrix would be completely (or almost completely) connected. In order to reduce noise, we apply a threshold and remove edges corresponding to low co-occurrence frequency. We find the threshold experimentally by finding a value for which the algorithm produces a sufficient number of clusters that are neither too small nor too large — at least the ten clusters we use in our experiments. In this problem domain, index pages should contain neither too few nor too many links. Too few links makes for an index page that is neither very interesting nor very useful, as it does not lead to much. Too many links makes the page difficult to navigate, as argued in the discussion of user effort in Section 3.2.1. We tried for clusters containing 20-30 links, with an absolute minimum of 5 and an absolute maximum of 100.

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 (see Figure 4.11 for timing data for PageGather and other clustering algorithms). 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

```
Procedure FindCC (co-occurrence matrix M)
    Mark all pages in M "unclustered"
    For each page p in M:
        if p is "unclustered":
            Create a new empty cluster c
            Call CC-Recurse(M, c, p)
Return all clusters c
End FindCC
```

```
Procedure CC-Recurse (matrix M, cluster c, page p)
    Add p to c
    Mark p "clustered"
    For each page q in M:
        if q is "unclustered" and M[p,q] is nonzero:
            Call CC-Recurse(M, c, q)
End CC-Recurse
```

Figure 4.5: The connected component algorithm used by PG_{CC}.

output — we do not want to create a new web page containing hundreds of links.

PG_{CC}'s connected component algorithm is shown in Figure 4.5. The algorithm starts from each page and recursively follows edges in the graph until it has found all pages reachable from the starting page. Those pages form a cluster. The algorithm does this for all pages to find a set of candidate clusters. Clusters under the minimum cluster size or over the maximum are discarded. PG_{Clique}'s clique-finding algorithm is shown in Figures 4.6 and 4.7. The basic method is to start with each page and try to grow all possible maximal cliques from that page. The algorithm iteratively adds neighboring pages to the current cluster and then checks to make sure the cluster is still a clique. It finishes when the clique has reached the maximum clique size, when no more neighbors can be added to form a clique, or when an expansion of the clique would result in a clique that has already been seen. The algorithm does not maintain a list of previously-seen maximal cliques, but explores the nodes in a

Procedure FindClique (co-occurrence matrix M)
For each page p in M:
Create a new empty cluster c
add p to c
Call Clique-Recurse(M, c, p)
Return all clusters c
End FindClique

Figure 4.6: The main procedure of the clique-finding algorithm used by PG_{Clique}.

canonical order that makes it possible to know when a clique has already been considered. cutoff threshold will have to be tuned differently for these two variations. PG_{Clique} is a much "stricter" algorithm than PG_{CC} — it has a stricter requirement for pages to be clustered together and will therefore tend to find fewer, smaller clusters for the same threshold value. When using PG_{Clique}, the threshold should be set lower to allow more edges in the graph, and hence more cliques. The issue of strictness and its impact on results is discussed in Section 4.4.4.

Implementation. The program that performs this step of the algorithm essentially reads in the frequency matrix described above, converts it into a two-dimensional array, and then uses the array to find cliques or connected components. The cutoff threshold is applied to zero out pairs with low co-occurrence frequency, and the array is treated as a graph by maintaining a list of the pages (nodes) and looking up non-zero cells in the array to find the set of edges to other pages. We maintain the matrix as an array in memory for the sake of speed; this approach does not scale well to larger data sets, where an array of size $O(N^2)$ is too large to maintain in memory; the 11,000 pages at the **Homes** site pushed the limits of our Windows NT machine with 64K of memory. Again, using a database would allow the algorithm to be scaled to larger data sets; our dataset was not large enough to require this approach. In order to facilitate comparison, both PG_{Clique} and PG_{CC} as well as the HAC and K-Means algorithms described above are implemented as procedures in this program; the desired algorithm and parameter settings are selected whenever the program

```
Procedure CLIQUE-RECURSE (matrix M, cluster c, page p)
    flag ChildFound = FALSE
    For each page q in M:
        if M[p,q] is nonzero:
            Create a new cluster c_n, a copy of c
            add q to c_n
            if c_n forms a clique and p precedes q:
                Call Clique-Recurse(M, c_n, q)
                flag ChildFound = TRUE
            else if c_n forms a clique
                flag ChildFound = TRUE
                delete c_n
            else
                delete c_n
        if (ChildFound is FALSE OR size of c > max clique size) then
        add c to list of clusters
    else
        delete c
```

Figure 4.7: The recursive procedure of the clique-finding algorithm used by PG_{Clique}.

is run. Fortunately, this step of the algorithm is fast enough to be run on demand at the webmaster's request — less than 30 seconds for both PG_{Clique} and PG_{CC} on both **Music Machines** and **Research**, though **Homes** required three minutes for PG_{Clique}(even HAC, the slowest of the algorithms, never took more than ten minutes). This step of the algorithm could be performed on demand and/or on a regular schedule.

4.3.4 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.

We therefore need a method to evaluate and rank the clusters.

One approach would be to test each cluster on a collection of testing data set aside from the data used to generate the clusters. Each cluster's performance on this data would be evaluated and the clusters ranked accordingly. This approach, however, significantly increases the amount of data and time required for the algorithm to run. We instead use a simpler method; using a training/testing data split or other techniques like cross-validation is one area for future work.

Our own approach is to minimize the time and data required by using information already available in the clusters and co-occurrence matrix. In particular, we evaluate each cluster by computing the average pairwise co-occurrence frequency between all its members. Recall that in step (3) we ignored much of the the co-occurrence frequency information, simply applying a threshold and treating all values above the threshold as equivalent for the purposes of clustering. In this step, we use that co-occurrence information again to rank the clusters.

Implementation. This step is performed just after the clustering described in the previous step, by the same program. The program outputs a list of clusters — each a list of site pages — in ranked order.

4.3.5 Combine or discard clusters that are too similar.

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 judgement, similar.

Implementation. Overlap reduction was implemented as a separate program that can be used to postprocess a ranked list of clusters. This allows us to generate the set of clusters once and then generate the merged and reduced sets for comparison purposes. Both techniques can be computed during a pass through the list of clusters, where each cluster must be compared to all previously seen clusters. In practice, this step takes no more than a second or two even on lists of hundreds of clusters. If the clusters were significantly larger, overlap reduction would take significantly longer, but our problem domain puts a practical limit on the size of the clusters, as described in Section 4.3.3.

4.3.6 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 Figure 1.1. The webmaster is prompted to accept or reject the cluster, to name it, and to remove any links she thinks inappropriate. Links are labeled with the titles of the target pages and ordered alphabetically by those titles. The webmaster is responsible for placing the new page at the site.

4.3.7 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 $O(LlogL)$ time. In step (2), we must process the log and create a matrix of size $O(N^2)$, which requires $O(L + N^2)$ time. Note that we implement our algorithm so that we can create the matrix from a collection of logs and use it repeatedly to generate candidate clusters. 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 $O(L + N^2)$ time.

4.4 Experimental Evaluation

In evaluating the PageGather algorithm, we wish to answer two questions:

1. **Which variant of PageGather is the best?** We compare the performance of several variants of the basic PageGather algorithm.

2. **How does PageGather compare with other techniques in index page synthesis?** We compare the quality of index pages generated by our algorithm with those generated by other algorithms and the human webmaster.

In this chapter, We describe experiments to address each of these questions. First, We discuss our experimental method. We close with discussion of our results.

4.4.1 Experimental Method

Web Sites Used

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

Our first site, **Music Machines**,[1] 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.[2] 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 pages from the web sites of faculty, staff, and students in the computer science department at the University of Washington.[3] **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.[4] **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.

---

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

[2] Data sets are publicly available from the author.

[3] `http://www.cs.washington.edu/people/`

[4] `http://www.cs.washington.edu/research/`

Evaluating Index Page Synthesis

In Section 3.2, We discussed the ideal measure of a site's quality and approximations based on the practical limitations of the data we have available and the kind of task we are trying to accomplish. We argued that true quality is measured in terms of recall and effort, and that we can evaluate site adaptations in terms of their impact and benefit. We now define how we measure these values when evaluating algorithms for index page synthesis.

We wish to evaluate the synthesized index pages using real data from the sites described above. However, in these experiments we do not place the synthesized index pages at the site; instead, we develop a metric that estimates the likely quality of the adaptations based on a set of test data from the site's logs. We take this approach for two reasons. First, in two of our three test sites, we do not have control over the content of the site and so cannot make adaptations. Second, as will be discussed in Section 4.4.4, the index pages generated by PageGather and other purely statistical approaches were too noisy to be placed before actual users. In essence, we estimate the quality of the index pages by simulating their placement at our site, based on a model of how they might be used.

Our measure of quality is essential to estimate the impact and benefit that the index pages *would have had* had they been added to the site. We identify users who would have benefited from the availability of the index pages and compute the likely benefit. As described in Section 3.2.3, we measure the quality of an adaptation to the site as a function of two factors:

- **Impact:** how many people are affected by the change.

- **Benefit:** how much they are affected.

To evaluate the **impact** of an index page, we must estimate the number of users who *would have* visited the index page. We presume that any user whose interests include the topic of the index page would visit the page. Since we assume that a user's interests are expressed in the set of pages she visits, we look for users who visit at least one page that is included on the index page. However, rather than arbitrarily choosing a threshold (one page, five pages, etc.) and deciding that all visitors who see that many pages have been "impacted," we visually represent different degrees of impact in the graphs of our results, as described below. 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.

To evaluate the **benefit** of an index page, we must estimate, for each user who would have visited the index page, how much the page would have helped her. As discussed previously, benefit can be measured in terms of improvements in recall and/or decrease in effort. Improvement in recall is difficult to measure; in order to do so, we would have to count the number of pages that the user *did not* see that she *would have wanted* to see. Since we limit ourselves to only the actual log data, we have little basis for making guesses about unseen pages. Designing a measure of improved recall is one area for future work, and will be discussed in Section 6.2. Instead, we limit ourselves to measuring benefit in terms of decreased effort. For each user in our experiment, we have a set of pages that she visited, some of which appear on the index page we are evaluating. The user's path through the web site represents whatever complex navigation she had to perform in order to find those pages; presumably, if some of those pages were all linked from a single index page, she would have had to exert far less effort. Therefore, we measure decreased effort in terms of the number of pages the user visited that are included in the index page. Because we only wish to compare algorithms, we do not attempt to measure exactly how much effort is saved; we simply count the number of pages that would have been easier to find. This measure does not factor in the possible benefit of these users finding *more* pages linked from the index page.

For example, suppose we are evaluating an index page on the topic of "guitars". The index page contains links to all twenty guitar-related pages at our web site. Suppose one user visits thirty pages at the site, five of which are guitar-related. Because she visits five of the pages linked from the index page, she counts as one person impacted by the index page, and the benefit to her is five.

Comparing Algorithms

We wish to compare the quality of the index pages generated by various algorithms. Each algorithm produces a ranked list of candidate index pages. We take each algorithm's top ten and evaluate each index page with respect to impact and benefit. We graph our results and visually compare the various algorithms.

Our graph is a visual representation of the relationship between impact and benefit for each algorithm (see Figure 4.8 for an example). Each algorithm is represented by a single line, which is the average of its top ten candidate index pages. The X axis shows benefit, in number of pages. The Y axis shows impact, in number of users. A point at $X = 5$ and $Y = 100$ for an algorithm means that 100 visitors were estimated to have a benefit of at least 5. Another point at $X = 7$ and $Y = 30$ means that 30 visitors had a benefit of at least 7. Clearly, this function is monotonically non-increasing, as no more visitors can experience a benefit of $q + 1$ than experienced a benefit of q. This representation is neutral on the question (raised in Section 3.2.3) of whether we prefer high impact/low benefit or low impact/high benefit adaptations, as neither line would dominate the other. However, if one algorithm is above another at all points, we conclude that the former produces more useful index pages. Note that, as our graphs exhibit exponential falloff, we display the impact axis on a logarithmic scale, which tends to downplay differences between algorithms; a small difference in a logarithmic scale translates to a substantial difference on a linear scale.

4.4.2 Variants of PageGather

As described in Section 4.3, the PageGather algorithm has two steps that can be implemented in different ways:

1. We may search the graph for cliques (PG_{Clique}) or connected components (PG_{CC}).

2. We may reduce overlap with either reduction or merging.

These choice points result in four variations of the PageGather algorithm: either PG_{Clique} or PG_{CC} with either reduction or merging. However, PG_{CC} will never produce overlapping clusters; by the definition of a connected component, any two clusters found that share a member will actually be the same cluster. With no overlap, overlap reduction has no effect, and both reduction and merging will produce the same results. Therefore, we have three variants of PageGather: PG_{CC}, PG_{CLIQUE^M}, and PG_{CLIQUE^R}.

In Section 4.3.3, we described how we find the graph threshold experimentally. In our experiments, we did this for all three 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^M} and PG_{CLIQUE^R}, the threshold was 0.05 for Music Machines, 0.13 for Homes, and 0.08 for Research. We must also tune the threshold for both reduction and merging in the overlap elimination stage. We experimentally tune the threshold as described in Section 4.3.5; 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.

Figure 4.8 shows a comparison of these three variants for all three of our test sites. On both **Music Machines** and **Research**, PG_{CC} is unambiguously better than either of the other variants — except for one point on the **Music Machines** graph (PG_{CLIQUE^R} has more impact than PG_{CC} at a benefit of 1), PG_{CC} always dominates the others. In the **Homes** graph, PG_{CC} dominates PG_{CLIQUE^M} and is approximately even with PG_{CLIQUE^R} — only slightly lower in impact at the tail end. Not only is PG_{CC} generally better than either of the PG_{Clique} variants, it is also generally faster, as shown in Figure 4.11. Accordingly, we used PG_{CC} as our standard for PageGather in further experiments.

4.4.3 Index Page Synthesis

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 4.4.1. As explained in Section 4.3.5, we use PG_{CC} — labeled "PageGather" — in all comparisons.

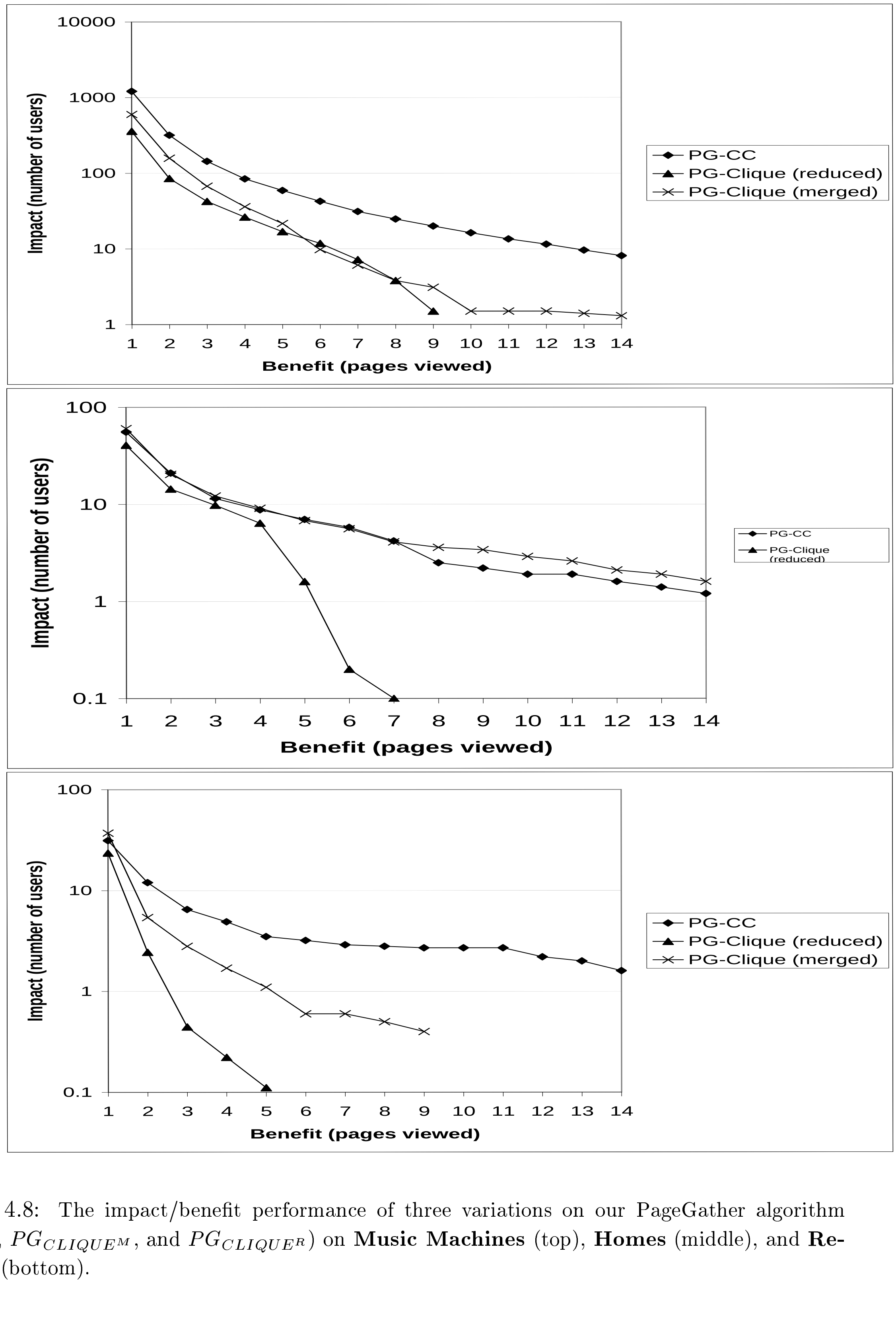

Figure 4.8: The impact/benefit performance of three variations on our PageGather algorithm (PG_{CC}, PG_{CLIQUE^M}, and PG_{CLIQUE^R}) on **Music Machines** (top), **Homes** (middle), and **Research** (bottom).

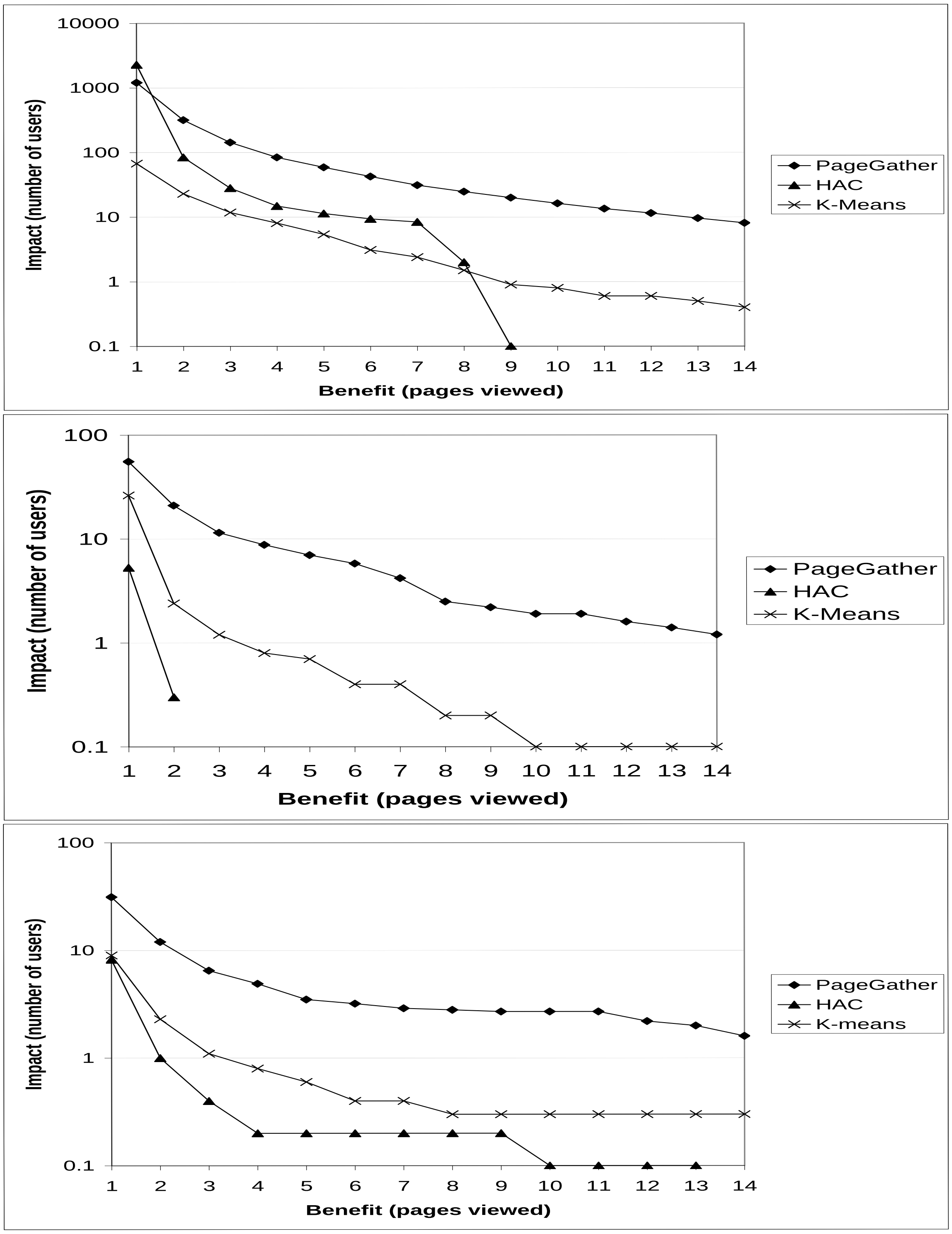

Figure 4.9: The impact/benefit performance of PageGather, HAC, and K-means on **Music Machines** (top), **Homes** (middle), and **Research** (bottom).

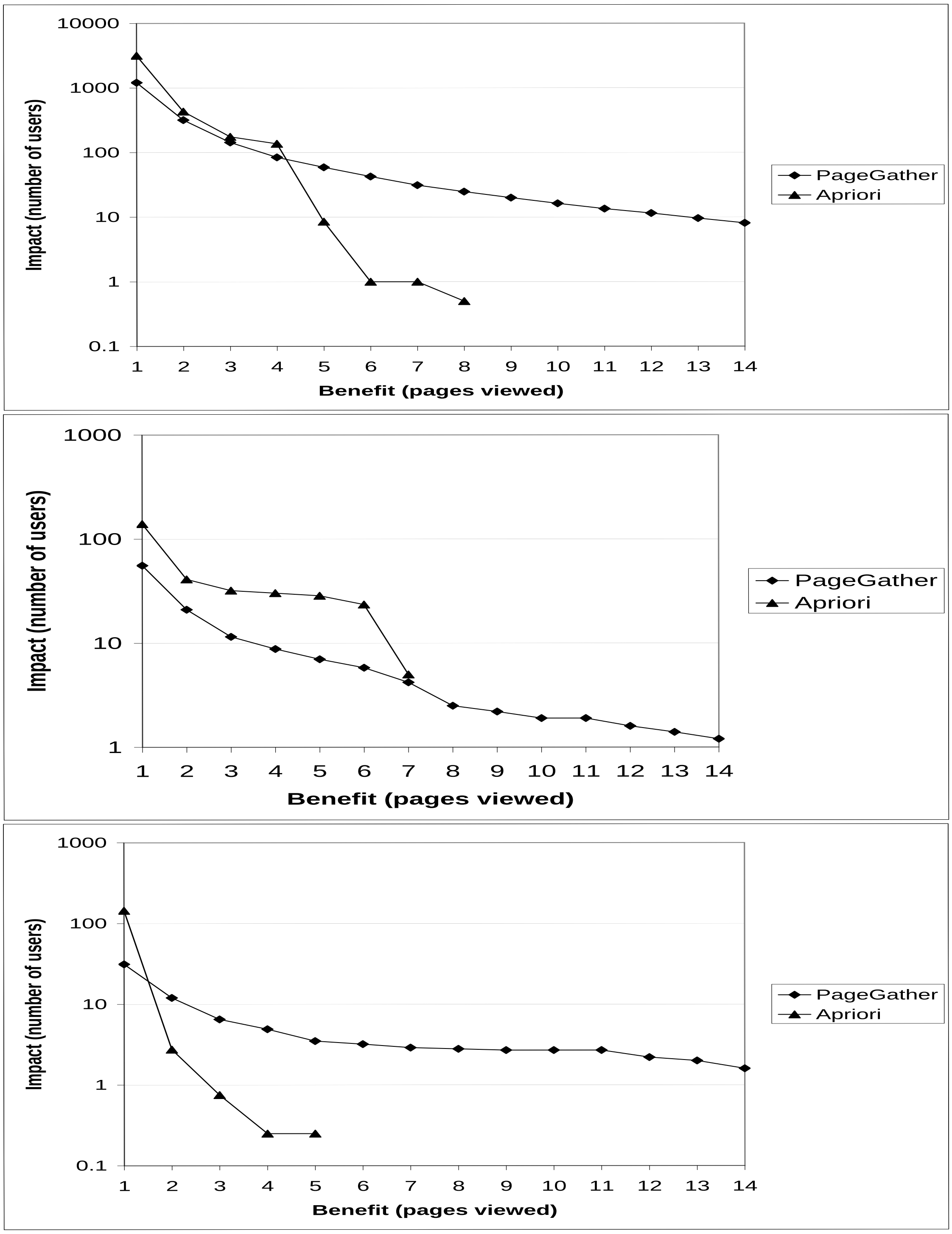

Figure 4.10: The impact/benefit performance of PageGather and Apriori on **Music Machines** (top), **Homes** (middle), and **Research** (bottom).

Clustering Algorithms

In Section 4.1, we discussed traditional statistical clustering. A traditional clustering algorithm can be adapted to perform cluster mining by using co-occurrence as the similarity measure and ranking the clusters the algorithm 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.

HAC and other clustering algorithms have three variants: a single-link version, a complete-link version, and a group-average version. All three use the same pairwise similarity measure but have different ways of defining the similarity between two clusters. In a single-link algorithm, the similarity between two clusters is equal to the similarity between their two *closest* members. In complete-link, the similarity is equal to the similarity between their two *most different* members. In group-average, the similarity between two clusters is the average of all the pairwise similarities between them. We have implemented both single-link and complete-link versions of HAC. In these experiments, we use the complete-link version. As with K-means, 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. We implemented the algorithm this way rather than using a similarity threshold (as described in Figure 4.3) to be able to easily force the algorithm to create a larger number of clusters from which we could choose the most coherent. 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 Figure 4.9, we compare PageGather with HAC and K-Means in all three domains. In all three, PageGather outperforms both clustering algorithms, completely dominating both except for one point (in the **Music Machines** graph, where HAC has a higher im-

	Music Machines	Homes	Research
PG_{Clique}	0:29	3:15	0:05
PG_{CC}	0:01	0:01	0:01
K-Means	0:09	0:05	0:03
HAC	9:36	7:40	2:26

Figure 4.11: Running time (in minutes:seconds) of PG_{Clique}, PG_{CC}, K-Means, and HAC.

pact for a benefit of 1). Of the two clustering algorithms, HAC performs somewhat better, but is significantly slower than both K-Means and PageGather (see Figure 4.11). K-Means and PageGather are comparable in running time, but PageGather produces far better impact/benefit scores.

The Apriori Frequent Set Algorithm

In Section 2.1.2, 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 4.3.5, 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 4.3.5) 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 Figure 4.10, we compare PageGather and Apriori on all three domains. In all three domains, Apriori performs at least as well as PageGather for low benefit values but falls off sharply. This pattern occurs because Apriori tends to find clusters of pages that are *very* tightly related, but such clusters are also very small. Small clusters can only provide so much benefit — because benefit is measured in terms of how many links on the page a user would follow, a cluster of five items can not provide a benefit of more than five. For example, in the **Research** experiments, no Apriori cluster is of size larger than 5, and so its impact/benefit curve stops at 5. Apriori's clusters are tight but small because it is a particularly *strict* algorithm; strictness is discussed in Section 4.4.4.

Apriori 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 4.3.2). 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 low 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.

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. In addition to the basic manufacturer-centric organization at **Music Machines** (described in Section 3.5.3), **Music Machines** contains

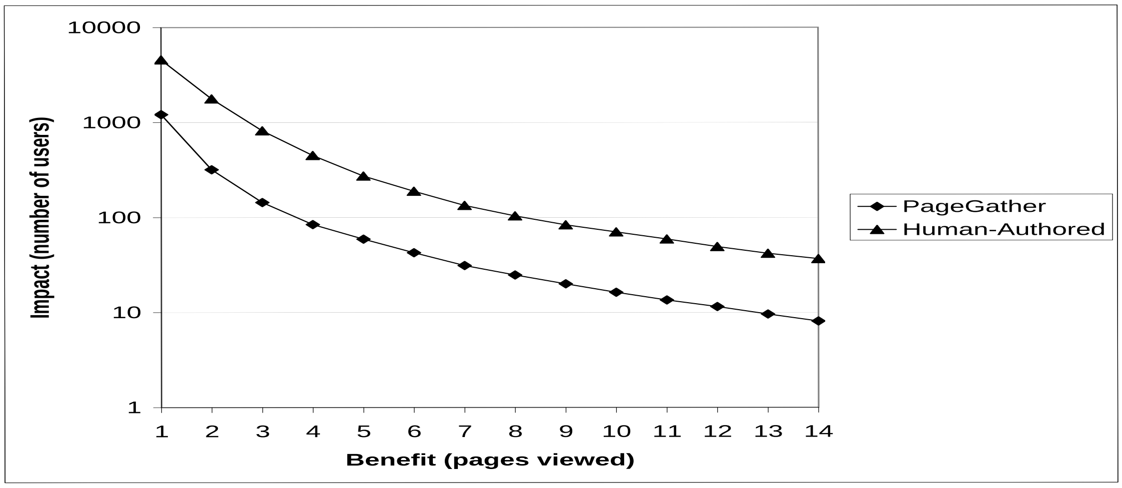

Figure 4.12: The impact/benefit performance of PageGather and clusters derived from human-authored index pages on Music Machines.

a few pages created by the webmaster to provide other views of the data. 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. Figure 4.12 shows the comparison of these clusters to the output of PageGather. We see that the human-authored clusters perform better than PageGather's. The two curves are of similar shape, with PageGather performing substantially worse than the human-authored pages.

4.4.4 Discussion

We briefly discuss two observations drawn from the experiments I have described in this chapter. First, that data mining algorithms can be classified along a spectrum of "strictness," and that an algorithm's degree of strictness is a strong predictor of its strengths and weaknesses. Second, that these statistical algorithms, even the very good ones, do not produce output suitable for our domain.

Strictness

We introduce *strictness*, a property of data mining algorithms, (including clustering algorithms, frequent set algorithms, and any other algorithm intended to produce sets of objects that are presumed to be similar). Generally, an algorithm of this type has a similarity measure which can be applied to any pair of objects to evaluate their degree of similarity. The algorithm also requires a measure of the similarity between two sets of objects (where one or both sets could be singletons), which is usually a function of the individual pairwise similarities between the two sets.

For example, HAC and other clustering algorithms have three variants: a single-link version, a complete-link version, and a group-average version. All three use the same pairwise similarity measure but have different ways of defining the similarity between two clusters. In a single-link algorithm, the similarity between two clusters is equal to the similarity between their two *closest* members. In complete-link, the similarity is equal to the similarity between their two *most different* members. In group-average, the similarity between two clusters is the average of all the pairwise similarities between them. The HAC algorithm continues merging clusters as long as their similarity is above a threshold. In the complete-link variant of HAC, we know that, after merging two clusters, all pairs of objects in the new cluster are above that threshold similarity. However, the single-link variant does not provide this guarantee; we know only that the most similar objects are above the threshold. Single-link clusters, therefore, can be highly irregular — they can be long chains of clusters that were merged only because two points were similar. Complete-link clusters, on the other hand, are more "spherical" — every point is within some distance from every other. Group-average clusters are somewhere in the middle; they are not as chaotic as single-link clusters but do not make the strict guarantee of the complete-link clusters.

The complete-link algorithm, then, is more *strict* than the single-link, in that it makes more guarantees about the similarity of objects in its clusters. We can approximately rank the various data mining algorithms discussed in this chapter — PG_{CC}, PG_{CLIQUE^M}, PG_{CLIQUE^R}, HAC (complete- and single-link), K-Means, and Apriori — according to their strictness, though precise ranking is sometimes difficult when algorithms differ greatly. In order of least to most strict:

1. HAC (single-link) and PG_{CC}
2. K-Means
3. PG_{CLIQUE^R}
4. HAC (complete-link) and PG_{CLIQUE^M}
5. Apriori

The relationship between PG_{Clique} and PG_{CC} is very similar to that between complete-link and single-link HAC. Finding a clique in PageGather's graph corresponds to finding a cluster where all pairwise similarities are above a certain threshold. Similarly, finding a connected component corresponds to creating a cluster where each member is similar to at least one other member (but not necessarily all). Single-link HAC and PG_{CC} are therefore the least strict algorithms. K-Means at least guarantees that all members of a cluster are closer to that cluster's center than to any other cluster's, though no guarantees are made about all pairwise similarities. PG_{CLIQUE^R} is slightly less strict than complete-link HAC and PG_{CLIQUE^M} because of the merging step. In this step, two clusters can be merged due to a high degree of overlap, but this merge does not guarantee that all pairwise similarities are above a threshold. In fact, if they were, the two would not be separate clusters in the first place. PG_{CLIQUE^M}, however, preserves the guarantee because new members are not added to the cliques — clusters are only removed from the ranked list. The clusters that remain are unchanged from their original clique form — the clique guarantee still holds. Finally, Apriori is the most strict of all. Apriori does not use a similarity measure like those used in clustering — instead, the algorithm counts how often objects appear together. Unlike in the clustering algorithms, larger sets are not formed based on a pairwise measure; in order for five objects, say, to be grouped together by Apriori, they must *all five* occur together a given number of times. Apriori, in other words, uses an N-wise similarity measure. This measure is much more strict than that used by any of the clustering algorithms.

Strictness has an apparent impact on the nature of an algorithm's output. In general, the stricter the algorithm, the "tighter" and more "coherent" the clusters it produces. Clusters will contain fewer unexplained outliers that seem to bear no relation to the rest of the contents but will also contain fewer novel and useful discoveries. Furthermore, a stricter algorithm will generally have more trouble finding clusters at all, and so clusters will be smaller and there may be fewer of them. For example, we find that the average pairwise co-occurrence frequencies computed for each cluster as part of the PageGather algorithm (see Section 4.3.4) tend to be higher for clusters found by PG_{Clique} than those found by PG_{CC}. On the other hand, PG_{Clique}'s average cluster size is around 7 while PG_{CC}'s is about 21.

Strict algorithms can also lead to a proliferation of very similar clusters. PG_{Clique} tends to produce many slight variations on the same basic cluster. The reason for this is that there may exist a well-connected set of pages that does not, however, form a clique. Many subsets, however, are cliques, and the algorithm returns these many similar subsets as clusters. If we decrease the co-occurrence threshold, allowing more pages to be connected by edges in our graph representation, one such set of pages may become a clique. However, there will generally always be some borderline sets that exhibit this behavior. We notice similar behavior with Apriori, which produces literally thousands of clusters, many of them almost identical. In fact, it was this tendency that required the overlap elimination step in the PageGather algorithm. The reduction variant simply addresses the problem of the proliferation of similar clusters by getting rid of most of them. Merging, however, addresses more of the difficulties created by strictness by creating larger clusters — and decreasing their strictness in the process.

In all three strict algorithms (complete-link HAC, PG_{Clique}, and Apriori), we find clusters that are generally quite small and, in a sense, too specific. For example, one of PG_{Clique}'s index pages for **Music Machines** was a small set of images of a few specific synthesizers made by one manufacturer. It is a very coherent and highly co-occurring set of pages, but the cluster is small and so specific that it has little impact on most visitors. For this problem, excessive strictness seems to reduce the usefulness of the results. Perhaps a larger collection of web pages, with a couple more orders of magnitude of possible connections would require stricter algorithms; they would have more data in which to find useful clusters, and the less strict algorithms would probably produce too much chaos. Strictness is a spectrum, and it represents a tradeoff between few, coherent clusters at one end and many, looser clusters at the other. What kind of algorithm is appropriate for a particular problem will depend on the amount of data available and the type of results desired. A deeper analysis of strictness and the relationships between different kinds of data mining algorithms could help us better understand the algorithmic tools we have and know which to choose for a particular problem. We believe it a fertile area for future research.

Problems with Statistical Clustering

```
/manufacturers/Boss/DR-55/samples/dr55kit.txt
/manufacturers/Korg/KPR-77/samples/kpr77kit.txt
/manufacturers/Korg/KR-55/samples/kr55kit.txt
/manufacturers/Roland/SH-synths/samples/sh09drum.txt
/manufacturers/Roland/System-100-100m/samples/sys100-drum.txt
/manufacturers/Univox/Micro-Rhythmer-12/samples/univoxbd.txt
/manufacturers/Roland/TR-606/samples/tr606kit.txt
```

Figure 4.13: A sample cluster from **Music Machines**.

Consider Figure 4.13, which shows the contents of a cluster found by PG_{CLIQUE^R} using data from **Music Machines**. This cluster is fairly typical of clusters found by PageGather — it has a general topic ("drum machine samples") but includes some pages that do not fit the topic, and does not include everything from the site that does fit the topic. This cluster represents a useful discovery: some number of visitors to the site are interested in the topic "drum machine samples."

However, this cluster cannot really be turned into an index page. What would the index page be called? Who would visit it? How would they benefit? If the page is called "drum machine samples," users will tend to be confused that it contains things that are not drum machines. Furthermore, they will probably assume that the page has links to *all* the drum machine samples at **Music Machines**. In other words, users expect an index page to be both:

- **Pure** — containing no inappropriate links, and
- **Complete** — containing *all* links that accord with the topic.

Before we turn the cluster in Figure 4.9 into an index page, we should remove the synthesizers and add all the other drum machines.

Statistical algorithms cannot accomplish this task, however. They have no provision for considering the conceptual structure of the object space and ensuring that their clusters correspond to meaningful topics. In fact, our problem as defined in Figure 3.4 does not include sufficient information to guarantee pure and complete index pages. In the next chapter, We redefine our problem and present conceptual cluster mining, a cluster mining algorithm for generating pure and complete clusters.

Chapter 5

CONCEPTUAL CLUSTER MINING

In the previous chapter, we presented PageGather, a statistical cluster mining algorithm for index page synthesis. PageGather is an algorithm for deciding what the contents of an index page should be. We argued, however, that PageGather is not a complete solution, as it cannot guarantee that the contents correspond to a coherent topic. In this chapter, we present IndexFinder, a *conceptual* cluster mining algorithm for producing pure and complete index pages. Just as statistical cluster mining is a variation on statistical clustering, conceptual cluster mining is a variation of *conceptual clustering.* We will describe conceptual clustering, present the IndexFinder algorithm, and describe an experimental evaluation of IndexFinder.

5.1 Index Page Synthesis: Redefinition

In Section 4.4.4, we argued that for automatically synthesized index pages to appeal to human users, they must be *pure* and *complete* with respect to comprehensible topics, where we define:

- **Pure** — containing no inappropriate links, and

- **Complete** — containing *all* links that accord with the topic.

In accordance with this observation, we refine the definition of the index page synthesis problem, first defined in Figure 3.4. The new definition, shown in Figure 5.1, adds the requirement that each index page be pure and complete with respect to a topic. Later in this chapter, we will more formally define what a topic is and how to ensure purity and completeness.

Given:

- A web site S (a set of HTML and other documents connected by hyperlinks).
- An access log L (a record of page requests made to the site).
- A set T of possible topics of interest, where each element of T is a subset of S.

Output all index pages P, where P consists of links to a set of pages in S such that:

- Pages in P frequently co-occur in user visits in L (for some definition of "frequently").
- $P = T_i$ for some $T_i \in T$

Figure 5.1: The Index Page Synthesis Problem, redefined.

5.2 Conceptual Cluster Mining

In Chapter 4, we presented cluster mining, a variant of traditional clustering. Similarly, we now introduce *conceptual cluster mining*, a variant of traditional conceptual clustering (as described in Section 2.2.1): 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* (see Figure 5.2). In refining the index page synthesis problem, we added the requirement that each index page correspond to a topic from among a set of possible topics of interest. This set need not be provided explicitly; in Figure 5.2, the set of possible topics is defined implicitly by a description language for concepts. In the case of **Music Machines**, our conceptual language consists of attributes describing the musical instruments and data such as the type of instrument, the price of the instrument, and where the instrument is made. A concept is a conjunction of these attributes, such as "cheap keyboards" or "pictures of American-made guitars". A page belongs to a concept if its attributes match those specified by the concept.

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

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_i of D of size less than k such that

- Each c_i is highly cohesive with respect to m (e.g., the average pairwise similarity of objects in c_i exceeds some threshold).
- c_i corresponds to a concept expressible in L, and includes exactly those objects belonging to that concept.

Figure 5.2: The Conceptual Cluster Mining Problem.

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 might we devise a tractable algorithm for this problem? I'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.

5.3 The SCML Algorithm Schema

The SCML Algorithm Schema $[\Omega, \Gamma]$

1. Run statistical cluster mining algorithm Ω 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 Γ.
   (d) Find the concept $v = \Gamma(c, D - c)$.
   (e) Find the set of objects c_v which is the extension of v.
3. Return all the sets c_v found.

Figure 5.3: The SCML algorithm schema.

Our key insight was that the members of a statistical cluster can be viewed as (possibly noisy) 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[1] algorithm schema (Figure 5.3), 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.

[1]Schema for Clustering with Machine Learning.

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. SCML is an algorithm *schema* because different algorithms may be used for the statistical cluster mining step and the concept-finding step.

5.3.1 Instantiations of SCML

```
Function GREEDY(positives,negatives,maxlength)
        rule = ""    length = 0
        do (
            bestscore = 0    length++
            for each a in attributes not already in rule
                for each value of a
                    p = positives matching rule + a=v
                    n = negatives matching rule + a=v
                    score = SCORE(p,n,rule,a,v)
                    if (score ¿ bestscore)
                      bestscore = score
                      besta = a    bestv = v
                    endif
                next v
            next a
            rule = rule + besta=bestv
            until ( length ¿= maxlength OR bestscore == 0 )
        return rule
```

Figure 5.4: The greedy concept learning algorithm used in PG_{POS} and PG_{MDL}.

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

```
Function SIMPLE-SCORE(pos,neg,rule,a,v)
         return pos + 1/neg
Function MDL-SCORE(pos,neg,rule,a,v)
         return mdl(rule) - mdl(rule+a=v)
```

Figure 5.5: The scoring functions used in the greedy concept learning algorithm used in PG_{POS} and PG_{MDL}.

problem. We then compare these three instantiations experimentally.

We instantiate the SCML schema using PageGather for Ω. We use three different algorithms for Γ. As a baseline, we wish use a simple algorithm that performs a greedy search from general to specific, constructing a conjunctive rule, as this approach seems well-tailored to our problem. The algorithm is based on GREEDY-3 [32] and iteratively builds a conjunctive rule, choosing a conjunct that maximizes a scoring function at each iteration (see Figures 5.4 and 5.5).

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 [40]. 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. The description length of a concept is a function of the number of bits required to represent the rule that specifies which objects are in the concept and the number of bits required

```
Procedure IREP(Pos,Neg)
begin
  Ruleset = ∅
  While Pos ≠ ∅ do
        /* grow and prune a new rule */
        split(Pos,Neg) into (GrowPos,GrowNeg)
          and (PrunePos,PruneNeg)
        Rule = GrowRule(GrowPos,GrowNeg)
        Rule = PruneRule(Rule,PrunePos,PruneNeg)
        ifthe error rate of Rule on
         (PrunePos,PruneNeg) exceeds 50% then
           return Ruleset
        else
           add Rule to Ruleset
           remove examples covered by Rule from (Pos,Neg)
        endif
  endwhile
  return Ruleset
```

Figure 5.6: The IREP algorithm, excerpted from [13].

to represent any exceptions to the rule. Adding a new conjunct to a rule increases the number of bits required to represent it, but could make the rule more accurate, leaving fewer exceptions to describe. In our PG_{MDL} algorithm, whenever we consider adding a new conjunct to our classification rule, we compare the description length (in bits) of the current concept to that of the new concept. Only if adding the conjunct decreases the description length do we extend the rule. Because the MDL measure takes into account the tradeoff between rule accuracy and rule length, we do not need a maximum rule length as we did with PG_{POS}.

The third algorithm is PG_{RIP}, which uses RIPPERk [13] for the concept learning algorithm Γ. RIPPERk 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 C4.5rules in both speed and performance. RIPPERk was chosen over a decision-oriented algorithm because RIPPERk learns sets of rules — which correspond nicely to the single concepts we wish to find — rather than entire decision trees. RIPPERk is based on the IREP algorithm (see Figure 5.6), with improvements to the rule-value metric, the stopping condition, and rule optimization. RIPPERk builds an ordered list of rules to be applied to examples for classification; we use only the first rule RIPPERk finds to describe our clusters.

5.4 Experimental Evaluation

5.4.1 Experimental Method

Attribute	Values
instrument-type	the type of instrument
form	what physical form the instrument takes
sonic	type of sound generation used
sonic2	type of synthesis for electronic instruments
effect-type	type of effect, for effects units
midi	whether it uses Musical Instrument Digital Interface
price	approximate cost of the instrument
age	whether the instrument is vintage or modern
filetype	the type of data the file represents
size	size of the manufacturing company
status	whether the manufacturer is still in business
location	where the instrument is produced

Figure 5.7: Attributes used in descriptions of documents at **Music Machines**.

Attribute	Values
instrument-type	synth, drum, sampler, effect, mixer, sequencer, controller, piano, other
form	keyboard, module, modular, console, other
sonic	analogue, digital, hybrid, acoustic, electric, other
sonic2	poly, mono, modular, additive, fm, rompler, modeling, sampler, other
effect-type	reverb, delay, modulation, multi, vocoder, other
midi	yes, no
price	cheap, mid-priced, expensive
age	vintage, modern, other
filetype	index, info, samples, images, mods, software, schematics, specs, other
size	small, medium, large
status	defunct, current
location	america, japan, europe, other

Figure 5.8: Attributes and values used in descriptions of documents at **Music Machines**.

The basic *simulation* method described in Chapter 4.4.1 of using impact and benefit to evaluate our synthesized index pages can be applied to the output of IndexFinder and other conceptual algorithms as well. 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 — the algorithms cannot function without such descriptions. Providing such information about a site requires a great deal of work and intimate knowledge of the site; accordingly, we have performed such experiments only on **Music Machines** — of which the author is the creator and webmaster.

Every document at **Music Machines** was labeled with values for a set of attributes. Attributes are described in Figure 5.7 and all values are listed in Figure 5.8. Not all attributes apply to every file. For example, **Music Machines** contains some files relating to specific instruments and some to specific manufacturers; the instrument-specific attributes do not apply to manufacturers (though the manufacturer-specific attributes can be applied to instruments, since every instrument has a manufacturer). Similarly, the "effect-type" attribute applies only to instruments of type "effect". Not every of **Music Machines**'s several thousand pages hand to be individually tagged. In general, one instrument will have a number of associated files, and the general instrument attributes need be entered only once for the instrument, changing only the filetype for different files.

We perform two kinds of experiments with conceptual algorithms. The first is equivalent to the simulation experiments described in Section 4.4.1, with the addition of conceptual descriptions as input to the algorithms. A limitation of these experiments, however, is that they are simulations — we compute only an estimate of how users might have benefited from various index pages, not a measure of actual benefit.

Therefore, for a second comparison between conceptual algorithms, we place automatically generated index pages at the Music Machines site and observe how people actually use them. For this *deployment* experiment, 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 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.

5.4.2 Variants of IndexFinder

We compare the three SCML instantiations described above using our simulation metric; Figure 5.9 shows the performance of PG_{RIP}, PG_{POS}, and PG_{MDL}. All three algorithms

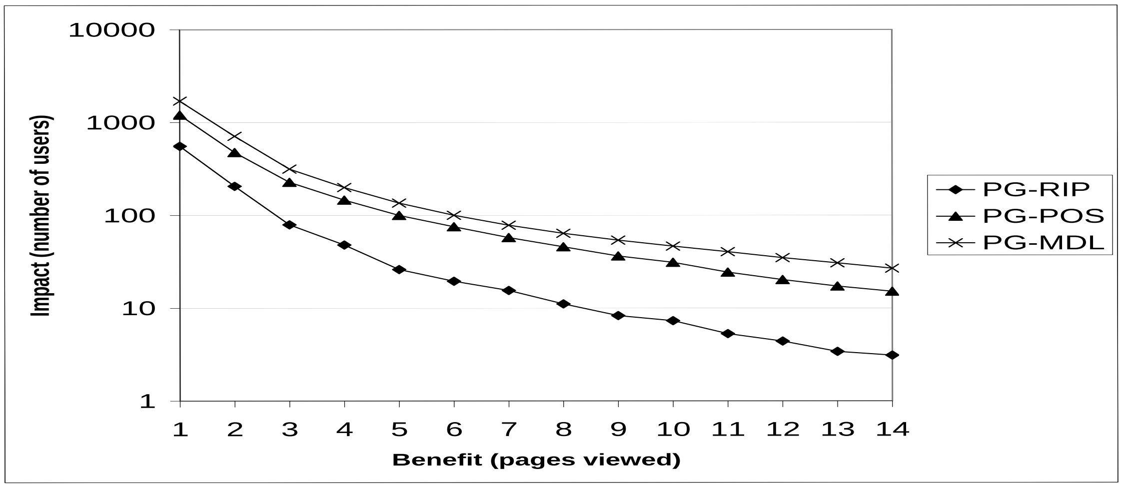

Figure 5.9: The impact/benefit performance of PG_{RIP}, PG_{POS}, and PG_{MDL} on Music Machines.

show curves of the same basic shape. PG_{MDL} and PG_{POS} are both clearly better than PG_{RIP}. Although PG_{MDL} and PG_{POS} appear to be close, PG_{MDL} dominates at all points, and the logarithmic scale of the graph de-emphasizes differences between curves.

Figure 5.10 shows the performance of index pages generated by the three instantiations of SCML when deployed at the **Music Machines** site. The graph very closely resembles Figure 5.9 in both shape and the relationship between the three curves, though the numbers are an order of magnitude larger in the simulation. In both experiments, PG_{MDL} dominates the other instantiations of SCML. In further experiments, we use PG_{MDL} as our representative version of IndexFinder.

5.4.3 Index Page Synthesis

In this section, we evaluate IndexFinder, comparing it to PageGather, COBWEB, and clusters derived from human-authored pages at a web site. We use the impact/benefit measure described in Section 4.4.1 as well as placing automatically generated index pages on the Music Machines site as described in Section 5.4.1.

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

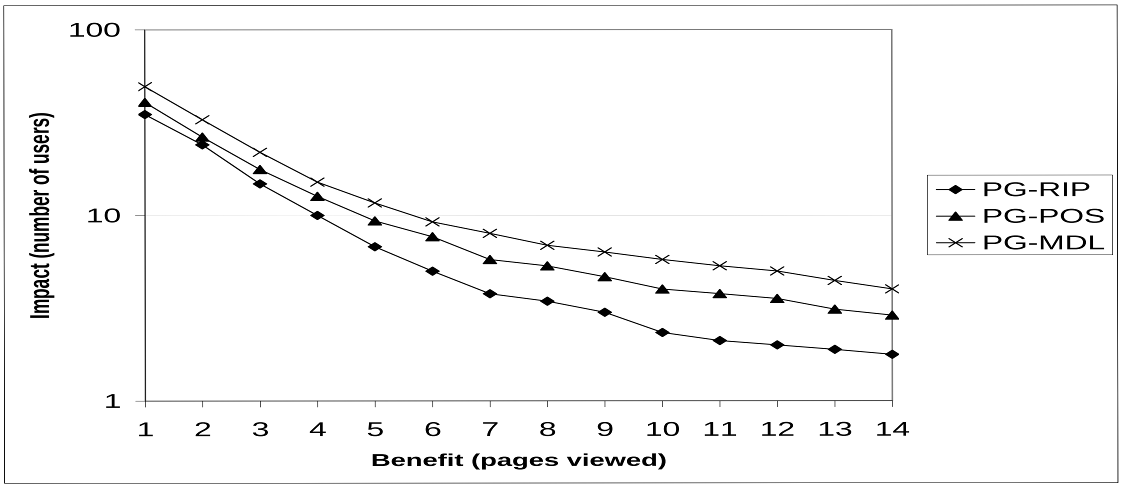

Figure 5.10: The impact/benefit performance of index pages generated by PG_{RIP}, PG_{POS}, and PG_{MDL} and actually placed at the Music Machines site.

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. We allow clusters to overlap or one to subsume another because the overlapping clusters may still both be of interest to users. For example, users might be interested in both the classes of "samples of keyboards" and "samples of cheap keyboards".

In Figure 5.11, we compare IndexFinder to PageGather, COBWEB, and human-authored index pages using our simulation metric. There is a clear ranking of the curves, from the human-authored pages at the top, to IndexFinder, PageGather, and COBWEB, respectively. Note that IndexFinder is relatively close to (and seems to be converging with) the human-authored pages, while it has substantially higher performance than COBWEB.

In Figure 5.12, we compare IndexFinder, COBWEB, and human-authored pages by placing index pages at the site. We do not include PageGather in this experiment because, as discussed in Section 4.4.4, the clusters generated by PageGather's statistical method do not easily lend themselves to being named and presented to users. Clusters produced by the conceptual approaches are named according to the rules they generate to describe the clusters. For example, if the algorithm finds a candidate index page with the conceptual

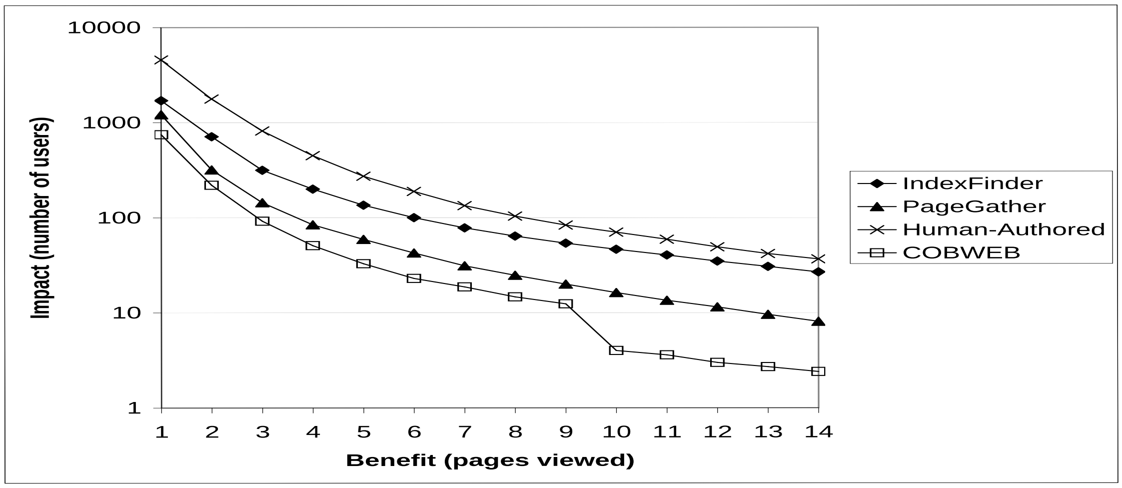

Figure 5.11: The impact/benefit performance of IndexFinder, PageGather, COBWEB, and clusters derived from human-authored index pages on Music Machines.

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.

As in the simulation experiment, IndexFinder outperforms COBWEB but does not do as well as the human-authored pages. Furthermore, as seen in the experiments described in Section 5.4.2, the simulation and deployment experiments produce similar graphs, of similar shapes, though the actual numbers differ significantly.

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. Figure 5.13 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.

5.4.4 Discussion

In Section 4.4.4, we argued that purely statistical approaches were inadequate for index page synthesis because users expect conceptual coherence in index pages. In this chapter, I have redefined the index page synthesis problem to account for this requirement and presented

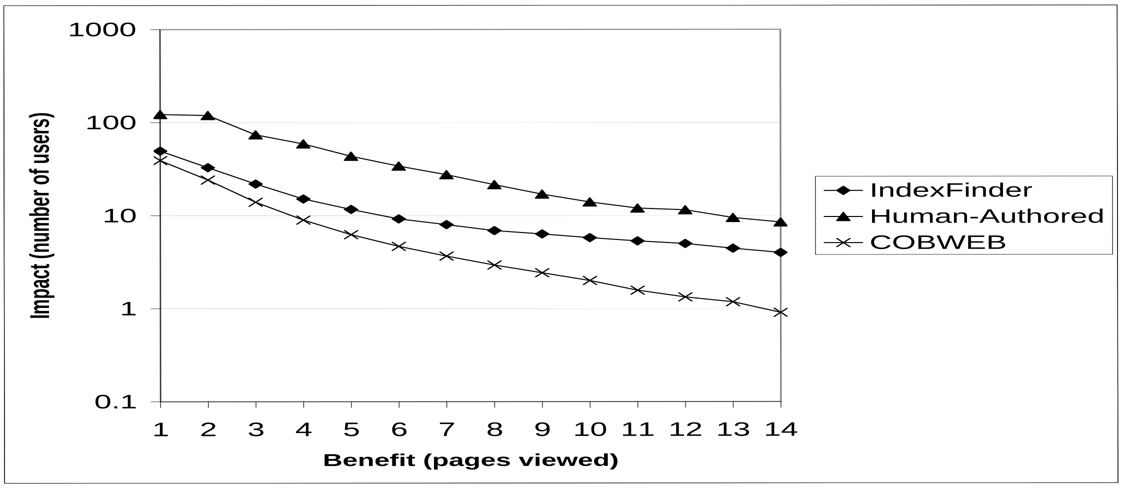

Figure 5.12: The impact/benefit performance of index pages generated by IndexFinder and COBWEB and clusters derived from human-authored index pages when actually placed at the Music Machines site.

IndexFinder, an algorithm to generate pure and complete index pages.

The advantage of generating index pages that are conceptually pure and complete comes at a cost: it is time-consuming to generate the page descriptions needed for this approach. As discussed above, each page (or group of related pages) at **Music Machines** was hand-described with respect to twelve attributes. Sites may have thousands or millions of pages and hundreds of descriptive attributes; hand-labeling would be prohibitive. On the other hand, large sites are increasingly stored in and generated from large databases, automated by systems like STRUDEL [15], and tagged with XML [23]. Pages at such sites often already have conceptual descriptions and are ready for conceptual clustering. Research in automatically describing or classifying pages (such as [21]) would also make this problem more manageable.

The cost of preparing the web site for conceptual clustering aside, we see significant advantages to this approach. Because PageGather tries to find the best statistical clusters, while IndexFinder imposes the additional requirements of conceptual purity and completeness, we might expect that IndexFinder would show worse performance according to our metric, as the price of addressing the problems of statistical noise. However, not only are

	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

Figure 5.13: Conceptual descriptions of the top five clusters found by IndexFinder and COBWEB.

IndexFinder's index pages more coherent, they also perform better according to our numeric metric — in our comparison, IndexFinder dominated PageGather at all points, by a substantial margin.

In Section 1.2, 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 not only does a better job finding the contents of index pages, it also guarantees purity and completeness with respect to a topic and can generate the titles of synthesized pages automatically.

Chapter 6

CONCLUSION

In Chapter 1, we argued that, as web sites increase in size and complexity, they must become adaptive to the needs of their users, improving their organization and presentation by learning from the way people use them. We have described one approach to web site adaptivity, index page synthesis. The work we have presented contributes both practical tools for web site adaptivity and deeper understanding upon which future research can build. In this chapter, we review the practical and theoretical contributions of this research and conclude with a discussion of future directions.

6.1 Contributions

Problem Definition. Web sites are a relatively new development in our society; research in web site adaptivity is even more recent and still in many ways inchoate. This research has contributed to the defining of this problem space by classifying related work and describing important design tradeoffs. These design issues include:

- **Customization vs. Transformation.** There are a spectrum of possible approaches to site adaptivity between pure customization and pure transformation. To date, most research projects and implemented solutions have taken the customizing approach. We have highlighted the tradeoffs between these extremes, and shown examples of effective transformation approaches.

- **Non-Destructive Adaptation.** Especially while the technology is young, guaranteeing that a web site will be safe from any unpredictable adaptations is vitally important. Even a reliable adaptive system, however, must work with the site's human designer to preserve her intentions. Of course, an adaptive site should not be entirely hamstrung in the interest of safety, but an adaptive site should err on the side of caution. We have provided several examples of minimally destructive adaptations, and the IndexFinder algorithm creates effective new pages with minimal effect on the existing pages.

- **Statistical vs. Conceptual Data Mining.** Statistical data mining algorithms have the advantage of processing massive amounts of data with minimal input from human beings; conceptual algorithms have the advantage of producing coherent, understandable output suitable for presentation to human users. We have discussed the implications of both approaches, and produced a hybrid algorithm that uses the advantages of both.

Data Mining Algorithms. Data Mining algorithms have evolved to address a number of common problems: partitioning a set of objects or documents into clusters, extracting association rules from market basket data, constructing a conceptual hierarchy over a set of objects, and others. The distinct character of the web domain, and of index page synthesis in particular, demands similar but not identical kinds of algorithms. In this dissertation, we have described cluster mining and conceptual cluster mining, two ways of adapting traditional data mining approaches to the web domain. We have both applied existing algorithms to these problems and presented new algorithms well suited for them, and compared their performance. PageGather is a cluster mining algorithm specifically designed for our application, but applicable to other problems where small clusters of closely-related objects must be found among a larger collection of data. In such problems, PageGather is faster and produces better clusters than adapted clustering algorithms. IndexFinder is a conceptual cluster mining algorithm that builds on PageGather to find clusters of similar objects that are pure and complete with respect to an easily expressible concept. Like PageGather, IndexFinder performs better on this problem than other data mining algorithms.

User Interface Issues. A particular feature of problems in the web domain is that they usually involve interaction with human users — the pages produced by an adaptive web site must always remain usable and understandable to the site's users. This fact has had a fundamental effect on the development of the IndexFinder algorithm — the requirement of purity and completeness in IndexFinder's output index pages is motivated by the need of users to have index pages with discernable topics and descriptive titles and to know that all (and only) relevant links are available on those pages. This requirement of comprehensible output exists, of course, in data mining and machine learning problems as well, but is often an afterthought; by making it an integral part of the algorithm, we believe, the results can be improved (as in the experimental results in Chapter 5) and used more effectively in real applications. Our contribution has been not in furthering the state of user interface design, but in bringing these issues into the heart of our research. Our results have shown not only that it is possible (and arguably necessary) to design with an eye toward human comprehensibility, but that such an approach produces better results.

6.2 Future Work

No piece of research is ever truly finished. We now describe a number of directions for future work, in three categories: improvements to the work I've described, extensions to this research, and more general directions for research.

6.2.1 Improvements to PageGather and IndexFinder

To track user visits across the site, we use the server logs, treating each IP number as a single user. All page hits from that IP number are considered a single user session. This approach has several problems in accurately tracking users: (1) multiple users may access the site from the same IP number (e.g., AOL users share a common gateway); (2) a single user may have multiple IP numbers (e.g., the common gateway may be dynamically distributed over multiple machines); and (3) pages may be cached for individual users or at gateways. We circumvented the caching problem by setting HTML "no cache" headers on all pages at **Music Machines**; however, this is not a desirable general-purpose solution. Otherwise, we treated any mistakes as allowable noise in the system. A more accurate approach would be to give each visitor to the site a cookie and log all page accesses with that cookie information. Using cookies adds complexity to the site, but the overhead is minimal and many sites use cookies already.

As discussed, we also assumed that every page hit for a particular user on a single day represented a single visit, and considered all pages in that visit to have co-occurred. However, two sessions separated by several hours should probably be considered separately. Partitioning the day's visits at significant time gaps would address this issue. Furthermore, a visitor to a site may pursue a variety of goals over the course of a visit; we may not wish to consider two pages that are separated by dozens of intervening pages to have co-occurred in the same way that adjacent pages have. This issue could be addressed by defining a limited window (pages must be within K hits of each other to have co-occurred) or to discount the co-occurrence according to the number of intervening pages.

6.2.2 Extensions to PageGather and IndexFinder

In our definition of the page synthesis problem, we touch on the importance of synthesizing *new* index pages — the system should not suggest new pages that largely duplicate existing ones. The PageGather algorithm, however, uses an ad hoc method to decrease the likelihood of generating duplicate pages: currently, PageGather discounts the co-occurrence frequency between pages that already appear together at the site. This approach could rule out useful pages; for example, two guitars may appear together on a "guitars" page but should still both be included on a "vintage instruments" page. This method also fails to prevent duplicate pages; for example, a "guitars" page may already exist at the site that includes all guitars, but the system could suggest a new page of "guitar images" — because the images are not directly linked together, IndexFinder does not consider them to already appear together. In a sense, this new page duplicates the topic covered by the existing page. One way to address this issue might be to describe not only individual pages in a conceptual language (see Figure 5.2), but existing index pages as well. Then, the topics of new pages could be compared against existing pages for duplication.

Currently in IndexFinder, in order for a set of pages to be considered a candidate index page, they must co-occur often across all users. If some set of pages (e.g., vintages basses) co-occurs among some small subset of users (e.g., bassists), it may be washed out in the general population. If we could discover such specific clusters, we could make those index pages available as well, perhaps customizing parts of the site to classes of users. The impact of such adaptations could be evaluated in terms of the size of the user class to which they apply, as well as how much benefit they provide. One way to do this would be to classify users into categories and find clusters that occur in each class. Users should be permitted to fall into multiple categories, both because they may have multiple interests and because excessive partitioning would reduce the amount of data available for learning. The evaluation function for candidate clusters could also be altered to take into account a cluster's benefit for a subset of users. Adding an index page of great use to 1% of the site's users will probably not significantly encumber the remaining 99%; however, this kind of approach could easily lead to a balkanization of the site and an increase in user confusion.

Perhaps the biggest open question about this research is whether it will scale well to significantly larger sites. **Music Machines** is large enough to provide interesting data, but sites such as Yahoo! and Amazon.com are orders of magnitude larger. Larger sites raise scaling issues in at least two ways. First, the algorithms as implemented probably do not scale well by several orders of magnitude. Limitations of space and time would probably make them impractical. Second, even if the computation could be scaled easily, we must keep a human in the loop, and the interaction between the human and the computer must not grow significantly more complex. Whereas IndexFinder might find a few dozen clusters at **Music Machines**, it would presumably found thousands at a site the size of Amazon.com. The appropriate direction for this research may be to apply the lessons of massive data mining to the PageGather and IndexFinder algorithms and then explore the delicate human/computer interface.

6.2.3 Other Research Directions

We have discussed the customization/transformation spectrum. Most existing research, including what we have described in this dissertation, as focused on techniques for one or the other. We believe, however, that both kinds of approaches have advantages and that hybrid approaches represent a good balance of these advantages. A fruitful direction for future research lies in exploring techniques for customizing to groups of users rather than individuals. Some research, such as the WebWatcher and much of the work in collaborative filtering, has already explored this direction — practical developments in adaptive web sites will probably come from such approaches.

Perhaps one of the most interesting and broadly applicable directions to extend the research I've described is further exploration of the relationships among various kinds of algorithms for finding sets of similar objects: clustering, learning, cluster mining, frequent set mining, etc. One dimension along which to compare these types of algorithms is that of *strictness*, discussed in Section 4.4.4. Strictness is a measure of how freely an algorithm groups items together as "similar"; a strict algorithm will tend to produce smaller, more tightly-packed clusters than a less strict one, all things being equal. The strictness continuum reflects a tradeoff between coherent but small and hard-to-find clusters at one end and chaotic but large and easier-to-find clusters at the other. A deeper exploration of this topic would attempt a (partial) ordering of algorithms by strictness, a taxonomy of types of algorithms by strictness criteria, and a theoretical and experimental evaluation of how different parts of the taxonomy perform on different kinds of problems. This work would have far-reaching implications for data mining, machine learning, web mining, and related fields.

As discussed in Section 3.6, this research has focused on users pursuing information-gathering tasks on the web. We do not believe, however, that this work is limited to that domain. Adaptivity can improve many aspects of a site aside from its information organization. Adaptivity can improve accessibility, whether to accomodate technological limitations (as in [6]) or user disabilities. Getting the most information in the least time is not the only measure of a successful web site; both advertisers seeking "eyeballs" and bored surfers seeking entertainment value the opposite. The techniques we have described for analyzing user access information for common patterns could be applied to these kinds of problems as well, perhaps by altering the quality measure.

6.3 Final Word

The World Wide Web is a relatively recent phenomenon in our history — as massive, distributed, heterogeneous library of human knowledge and trivia, a complex and uneven interface to the world's largest knowledge base. It is an entity of astounding complexity, and we are only beginning to explore how to make it accessible and understandable, even adaptable to the needs of the people who use it. We have described one such exploration: making web sites easier to use by making them adapt to the way people use them.

The web domain has certain characteristics that guide research:

- The availability of massive amounts of data.
- The need to find small nuggets of interest without classifying all data.
- The need to produce human-understandable results.

These conditions occur in many Web-related research and design problems, as well as in other areas in artificial intelligence, data mining, and machine learning. The approaches we have described here — cluster mining and hybrid statistical/conceptual learning — are well suited to many such problems.

BIBLIOGRAPHY

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

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo. Fast discovery of association rules. In U. Fayyad, G. Piatetsky-Shapiro, O. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. MIT Press, Cambridge, MA, 1996.

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

[4] Rakesh Agrawal and Ramakrishnan Srikant. Mining Sequential Patterns. In *Proceedings of the International Conference on Data Engineering (ICDE)*, Taipei, Taiwan, March 1995.

[5] C. Anderson, P. Domingos, and D. Weld. Adaptive web navigation for wireless devices. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2001. Submitted.

[6] C. Anderson, P. Domingos, and D. Weld. Personalizing Web Sites for Mobile Users. In *Proceedings of the Tenth Int. WWW Conference*, 2001.

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

[8] M. Balabanovic and Y. Shoham. Fab: Content-based, Collaborative Recommendation, 1997.

[9] J. Borges and M. Levene. Data Mining of User Navigation Patterns. In *Proc. of Workshop on Web Usage Analysis and User Profiling (WEBKDD), 5th Int. Conf. Knowledge Discovery and Data Mining*, 1999.

[10] I. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White. Visualization of Navigation Patterns on a Web Site Using Model-Based Clustering. In *Proc. 6th Int. Conf. Knowledge Discovery and Data Mining*, 2000.

[11] P. Cheeseman and J. Stutz. Bayesian Classification (AutoClass): Theory and Results. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, Cambride, MA, 1996.

[12] M. Chen, J. Park, and P. Yu. Data mining for path traversal patterns in a web environment. In *Proc. of the 16th International Conference on Distributed Computing Systems*, 1996.

[13] W. Cohen. Fast effective rule induction. In *Proc. 12th Int. Conf. Machine Learning*, 1995.

[14] D. Cutting, D. Karger, J. Pedersen, and J. Tukey. Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections. In *15th Annual Int'l SIGIR92*, 1992.

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

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

[17] D. Fisher. Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning*, 2:139–172, 1987.

[18] D. Fisher. Iterative Optimization and Simplification of Hierarchical Clusterings. *Journal of Artificial Intelligence Research*, 4, 1996.

[19] N. Good, J. Schafer, J. Konstan, A. Borchers, B. Sarwar, J. Herlocker, and J. Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 1999.

[20] S. Hanson and M. Bauer. Conceptual Clustering, Categorization, and Polymorphy. *Machine Learning*, 3:343–372, 1989.

[21] C. Jenkins, M. Jackson, P. Burden, and J. Wallis. Automatic RDF Metadata Generation for Resource Discovery. In *Proceedings of the Eighth Int. WWW Conference*, 1999.

[22] T. Joachims, D. Freitag, and T. Mitchell. WebWatcher: A Tour Guide for the World Wide Web. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 770–775, 1997.

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

[24] H. Lieberman. Letizia: An Agent That Assists Web Browsing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 924–929, 1995.

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

[26] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, Berkeley, CA, 1967. University of California Press.

[27] H. Mannila and H. Toivonen. Discovering Generalized Event Episodes Using Minimal Occurrences. In *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining*, pages 146–151, 1996.

[28] R. Michalski and R. Stepp. *Learning from Observation: Conceptual Clustering*, pages 331–363. Morgan Kaufman Publishers, 1983.

[29] B. Mirkin. Concept Learning and feature Selection Based on Square-Error Clustering. *Machine Learning*, 35:25–39, 1999.

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

[31] D. Mladenic. Personal Webwatcher: Design and Implementation. Technical Report IJS-DP-7472, Carnegie Mellon University, School of Computer Science, 1996.

[32] G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–100, 1990.

[33] M. Pazzani and D. Billsus. Learning and Revising User Profiles: The Identification of Interesting Web Sites. *Machine Learning*, 27:313–331, 1997.

[34] M. Pazzani and D. Billsus. Adaptive Web Site Agents. In *Proc. Third Intl. Conf. Autonomous Agents*, 1999.

[35] M. Pazzani and D. Billsus. Evaluating Adaptive Web site Agents. In *Workshop on Recommender Systems Algorithms and Evaluation, 22nd International Conference on Research and Development in Information Retrieval*, 1999.

[36] M. Pazzani, J. Muramatsu, and D. Billsus. Syskill and Webert: Identifying Interesting Web Sites. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 54–61, 1996.

[37] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu. Mining Access Patterns Efficiently from Web Logs. In *Proc. 2000 Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD'00)*, 2000.

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

[39] J. R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81–106, 1986.

[40] J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.

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

[42] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, pages 175–186, 1994.

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

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

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

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

[47] U. Shardanand and P. Maes. Social Information Filtering: Algorithms for Automating "Word of Mouth". In *Conference on Human Factors in Computing Systems – CHI '95*, 1995.

[48] M. Spiliopoulou. Web usage mining for site evaluation: Making a site better fit its users. *Special Section of the Communications of ACM on "Personalization Technologies with Data Mining"*, 43(8):127–134, Aug 2000.

[49] M. Spiliopoulou, L. Faulstich, and K. Winkler. A Data Miner analyzing the Navigational Behaviour of Web Users. In *Proc. of the Workshop on Machine Learning in User Modelling of the ACAI'99 Int. Conf.*, July 1999.

[50] M. Spiliopoulou, C. Pohle, and L. Faulstich. Improving the effectiveness of a web site with web usage mining. In *Advances in Web Usage Mining and User Profiling: Proceedings of the WEBKDD'99 Workshop*, pages 139–159. Springer Verlag, 2000.

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

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

[53] A. Wexelblat and P. Maes. Footprints: History-Rich Web Browsing. In *Proc. Conf. Computer-Assisted Information Retrieval (RIAO)*, pages 75–84, 1997.

[54] P. Willet. Recent Trends in Hierarchical Document Clustering: a Critical Review. *Information Processing and Management*, 24:577–97, 1988.

[55] T. Yan, H. Jacobsen, H. Garcia-Molina, and U. Dayal. From User Access Patterns to Dynamic Hypertext Linking. In *Proceedings of the Fifth Int. WWW Conference*, Paris, France, 1996.

[56] O. Zaiane, M. Xin, and J. Han. Discovering Web Access Patterns and Trends by Applying OLAP and Data Mining Technology on Web Logs. In *Proc. of IEEE Conference on Advances in Digital Libraries*, 1998.

Appendix A

VITA

Mike Perkowitz

Ph.D.	The University of Washington	March, 2001
M.S.	The University of Washington	June, 1995
Sc.B.	Brown University	May, 1993