A Machine Learning Approach to Web Personalization

Corin R. Anderson

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

Doctor of Philosophy

University of Washington

2002

Program Authorized to Offer Degree: Department of Computer Science & Engineering

University of Washington
Graduate School

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

Corin R. Anderson

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.

Co-Chairs of Supervisory Committee:

Daniel S. Weld

Pedro Domingos

Reading Committee:

Daniel S. Weld

Pedro Domingos

Oren Etzioni

Date:

University of Washington

Abstract

A Machine Learning Approach to Web Personalization

by Corin R. Anderson

Co-Chairs of Supervisory Committee:

Professor Daniel S. Weld
Computer Science & Engineering

Assistant Professor Pedro Domingos
Computer Science & Engineering

Most web sites today are designed one-size-fits-all: all visitors see the exact same pages regardless of interests, previous interactions, or, frequently, even browsing client (desktop PC or wireless PDA). But one size often does not fit all. Instead of presenting the same content, the web experience should be dynamic and personalized, adapting to visitors' preferences as evinced in previous interactions.

This thesis proposes a framework for personalizing the web experience. Within our PROTEUS framework, we view personalization as a two-step process of first modeling users, and then improving the site given the model. We frame this problem as a machine learning task: the goal is to predict users' web navigation given their previous behavior. We explore several means of personalization, including improving the wireless web and building personalized, dynamic portals, and concentrate on one in particular—automatically adding shortcuts to likely navigation destinations.

A challenge in modeling web navigation is that training data for an entire site may be plentiful, but sparse for any individual page. This difficulty can be overcome, however, by noting that most large web sites have a rich underlying relational structure that can be exploited for generalization: pages can belong to different types (*e.g.*, pages about laptop computers versus pages about printers), with each type described by a different set of attributes (*e.g.*, size of display versus printing speed).

We leverage this structure by developing relational Markov models (RMMs), a novel extension to Markov models. States in an RMM belong to relations and are described by variables over hierarchically structured domains. Based on these hierarchies, the RMM defines sets of related states, learns transition probabilities between these sets, and uses shrinkage to estimate transitions between individual pages. This thesis presents RMMs in detail and provides results showing that they outperform traditional Markov models for predicting web navigation by a substantial margin.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This thesis, nay, my pursuit of a Ph.D., would not have been possible without the support, understanding, and encouragement of many of my friends and family. My advisors, both past and present, have shown me what good research is, and helped steer me along that path. Thanks, Oren Etzioni, for giving me my first start at AI research and a chance to improve the Web; and thanks, Alon Halevy, for my first taste of being co-advised and the opportunity to watch two masters at their art. Thanks to Pedro Domingos for an endless supply of insight, inspiration, and explanation of even the most difficult and thorny subjects. And, of course, thanks to Dan Weld, for advising me through many diverse topics, and whose advice, wit, and keen ability of observation I will remember long after I graduate.

I've been fortunate to have become friends with many others along the road I've taken. Special thanks to Tessa Lau and Steve Wolfman, who always lent an ear and had thoughtful comments about anything on my mind. Thanks also to the AI cadre: AnHai Doan, Julie Goldberg, Dave Hsu, Geoff Hulten, Cody Kwok, Don Patterson, Matt Richardson, and Sarah Schwarm, and to the many others whom I've had the pleasure of association: Zack Ives, Michael Noth, Rachel Pottinger, Ka Yee Yeung, and Doug Zongker.

Last but certainly not least, I could not have finished my degree without the love and support of my family. I cannot express how lucky I am to be related to such a great bunch; thanks, so much, to everyone. Thanks, Casey, for distracting me with Age of Empires and MarioKart when I needed it, and for graduating from the UW with me again. Thanks, Mom, for editing all my papers, and for keeping me company on those too-early commutes. Thanks, Dad, for many late-night rides home and for so many homemade sandwiches. I love you all.

Chapter 1

INTRODUCTION

Information access has become a ubiquitous part of our daily lives. We consult the web for everything from local news, to technical references, to answers to specific questions, such as "Where's the seminar I need to be at in 5 minutes?" *Navigation* is the art of accessing information on the web—following links and browsing through content. Unfortunately, in today's web, this art is a rather dissatisfying one. It is dissatisfying because, as we navigate, we are constantly bombarded with content and links that are irrelevant to the task at hand. Traversing this sea of distractions is frequently annoying and, at times, so frustrating that we give up on our information quest altogether.

Following in the spirit of Perkowitz and Etzioni's challenge to the AI community [80], we propose building *adaptive web sites* that *personalize* the web experience for each visitor. A personalized web experience is one that has been customized to an individual visitor or group of visitors (as opposed to a transformation intended for the entire web audience). Through personalization, we can improve the navigation on a web site by, for example, highlighting content and links of interest, hiding those that are irrelevant, and even providing new links in the site to the user's likely web destinations. We are particularly interested in automated approaches (as opposed to ones that demand the user's explicit input), and thus frame the problem as a machine learning one. The goal is to predict a user's actions on the Web (or on a single web site), based on observations of the current visit, other users, and data describing the sites viewed. In this chapter, we further motivate this problem, provide several scenarios in which personalization is beneficial, and outline our general approach.

1.1 Motivation

Most web sites today are designed with a one-size-fits-all philosophy: the site designer determines the needs of the visitors and builds the content accordingly. However, one size frequently does not

fit all. Visitors may use the site in ways different from the designer's expectations and for which the site's presentation (pages) or navigational structure (links) are not well-suited. Visitors' browsers may be unable to display the content as the designer intended, because of screen resolution or bandwidth constraints (*e.g.*, the visitor is browsing from a wireless PDA or cell phone). Even the same visitor may have different interests on different occasions, perhaps based on content viewed at another site (for instance, after viewing a news story about a merger between two companies, a visitor may be more interested in the stock quotes for those companies than for the usual portfolio).

Personalization can improve the web experience by adapting:

- **The presentation of a single page**, by rearranging regions of the page to put the most important material near the top (to increase its *salience*);
- **The content on each page**, by adding data that is highly relevant, or eliding data that is irrelevant;
- **The navigational structure of a site**, by adding links between previously non-linked pages that help lead visitors to their web destinations;
- **The navigation between or aggregation of several sites**, by building portal sites combining content and links from many frequently-visited sites.

Personalization can improve a wide range of web experiences, and we highlight three such scenarios here.

1.1.1 Scenario 1: Information-seeking browsing

Suppose that every Tuesday a visitor, Craig, looks for the title and abstract of the day's colloquium for the University of Washington Computer Science Department. Craig starts at the main entry page (`www.cs.washington.edu`), follows the link to the "Events and Talks" page, follows the link to the "Colloquia" page, scrolls through the list of colloquia for the current quarter, and finally follows the link for the current day's colloquium. To find this information again the next week, he must follow these same steps—drilling down through several pages, scrolling content, and hunting

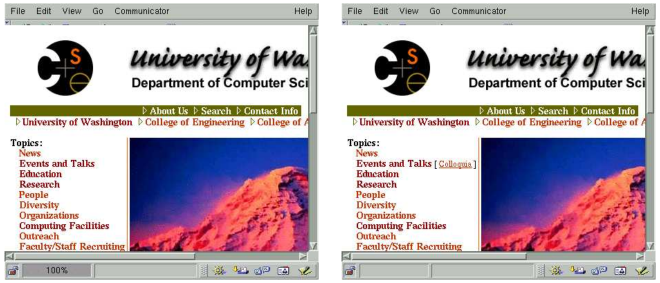

Figure 1.1: **Shortcut link to colloquia.** A personalized link to the Colloquia page is added because it is a frequent destination.

for the right link. Each of these steps takes time and is an opportunity for error, and completing the entire sequence correctly each week is unnecessarily time-consuming.

In contrast, compare the same interaction on a personalized site. In the first week of the term, Craig follows the same links as before. But by the following week, the site can immediately suggest a link directly to the Colloquia page, because it has observed Craig's previous navigation (see Figure 1.1). The link is even placed near the "Events and Talks" link, as that appears to be where Craig starts his quest for this information. With this shortcut, instead of following three links and scrolling extensively to reach his destination, Craig follows just two links, saving a great deal of time and effort.

1.1.2 Scenario 2: Wireless web browsing

One of the fastest growing communities of web users is that of *mobile* visitors—people who browse the Web with wireless PDAs, cell phones, and pagers. Nearly all cell phones sold today are "web-ready" and authorities predict that the number of wireless Internet devices will outnumber desktop computers in the near future [86]. Despite this trend, however, few web sites today cater to mobile visitors, instead optimizing their content exclusively for desktop clients. Unfortunately, mobile devices are not as capable as their desktop counterparts, being limited by small screens, low-bandwidth networks and slower processors.

In this second scenario, suppose Cathy is on a commuter train and wants to read a summary of the day's Mariners baseball game. At her desk, she'd visit the team site (`www.mariners.org`) with her PC browser, but, when commuting, she has only her wireless PDA. Cathy enters the URL by hand in her PDA and reaches the first page. Her PDA's browser does not support cascading style sheets [64] or JavaScript [42], and few of the embedded images load immediately, so she has some difficulty orienting herself on the page. Cathy scrolls through several screens to find the link for the current game, and clicks to follow it. The PDA's network connection is only 19.2 Kbps, so nearly 30 seconds pass before the new page is downloaded. Unfortunately, even after the page is loaded completely, Cathy cannot find the box score on the page, because it is buried among superfluous content. She finally gives up in frustration, deciding instead to wait until she gets home to catch the game highlights.

Most of the difficulty in this scenario is caused by the small screen size, slow network, and limited capabilities of the PDA. Several companies [10, 89, 56, 37] propose to address these difficulties by *transcoding* the original content into a simpler format for the PDA. Replacing complex HTML with simpler code will improve the visual appearance of each page, but will not reduce how much user interaction is needed to find the target content. A personalized mobile web experience, however, could reduce these actions. The personalized site would observe a user both on the wireless device and the desktop computer to build a model of interests and navigation. When the user requests a page from the wireless device, the personalizations would adapt to the cost model of the device, for example, by displaying only text, and aggregating content from many pages into one HTML document (because scrolling on a PDA is less costly than following a link). In this scenario, personalization is not simply about saving a link here and there—it is about bringing content and visitors together before the visitor gives up in frustration.

1.1.3 Scenario 3: Personal web portal

This third scenario highlights how personalization can improve the everyday web experiences of a typical software developer, whom we will call Casey. Casey might read a web-based newspaper as soon as he gets to work, and then spend a few hours on software development, with intermittent consultation of online programming documentation. Following a break at noon for lunch and to read comics on the Web, Casey returns to programming, might take a mid-afternoon break to check news and a few more comics, and finally consult online transit information shortly before leaving at 5:30 P.M. Such stereotypical patterns of web access are common. However, despite the regularity with which users view content, few mechanisms exist to assist with these routine tasks. Lists of bookmarks must be authored and maintained manually by users and are presented in a cumbersome hierarchical menu. Links and content on manually personalized portals, such as MyMSN [70] or MyYahoo! [95], are more easily navigable, but still must be chosen and managed by users in an explicit manner.

Given an automatically personalized portal, Casey would direct his web browsing through the portal's proxy, allowing it to record and model his online behavior. Whenever Casey opened a browser window or returned to the "Home" page, the portal would display a personalized, dynamic page, including links and content of likely interest to him. Moreover, this portal could depend on the context of the browsing session, including the time of day, the topic of recent browsing, the other applications currently running on Casey's computer, *etc.* This personalized portal does not adapt each individual page requested, but provides a view of the information Casey is likely to seek in the current browsing session. The personalized portal would save Casey effort in finding the content he wants, when he wants it.

1.2 Our two-step approach to personalization

We view personalization as a two-step process. In the first step, a web personalization system (or *personalizer*) builds a model of the user, including aspects that model the user's interest in content and navigation behavior. In the second step, the personalizer adapts the web experience, transforming content or navigation to maximize the value of the site given the user model. Next, we describe these steps in more detail.

1.2.1 User modeling

The primary goal of user modeling is to enable prediction of a user's actions on a personalized web site, and thus to determine which adaptations are useful for the visitor. There is a rich literature describing user modeling in general and the interaction between it and adaptation in specific (see,

for example, the overview of user modeling systems by Kobsa [60]). In our work, we model two aspects of a user: interest in content and navigation behavior. Our model of a user's interest in content relates to the textual content on each page—the words displayed. We model this aspect using a traditional information retrieval approach to measure similarity between the set of words of interest to a specific individual and the set of words appearing in a document. A user's navigational behavior is modeled as a Markov chain (*i.e.*, the user is an agent transiting probabilisticly through a state-based world), conditioning on recent navigation history.

1.2.2 Adapting content

To generate a personalized web experience for a visitor, we use heuristic search through the space of sites[1]. This approach is is very similar to that presented by Perkowitz [79], although our models of a site and a visitor are more sophisticated; we present them in Section 2. We are also more concerned with *personalizing* content for an *individual* visitor, as opposed to making adaptations generally useful for a site's entire web audience, although our approach is relevant to both targets.

Each state in the search space conceptually describes a web site—the navigational structure, the content on each page, and how each page is graphically presented. The representation of the site may rely on the HTML of the pages directly, or, for sites built atop databases, the representation may use the queries and HTML templates defining the content. The search operators personalize various elements of the site, as suggested in Section 1.1. The value of each state (*i.e.*, personalized site) is the expected utility the visitor receives by browsing that site. Intuitively, expected utility is the sum of the utilities of each page on the site, discounted by the difficulty in navigating to each page. The expected utility depends on the model of the visitor. The search for the personalized site continues until an optimal site is found, or until computational resources are exhausted.

1.3 Summary of the thesis

The primary questions we address in this thesis are:

[1] We use the term "site" to mean either a traditional web site or a portal site, the latter of which may loosely combine content from many, disparate web sites. While the content of such portal sites transcends a single web *server*, it still consists of pages and links, and referring to it as a site simplifies our discussion

- How can automated personalization improve the web experience?
- How can automated personalization scale to adapt content for each visitor at each page impression?

PROTEUS. Our first implemented personalizer, PROTEUS, aims to address the first question. The PROTEUS personalizer embodies our PROTEUS architecture for web personalization, which we describe further in Chapter 3. The PROTEUS system personalizes web browsing for visitors using wireless PDAs (such as Palm VIIs) at many web sites, adapting each site in turn. The system models users by their textual content interests and the sequence of navigation actions they perform, emphasizing the content interests and attempting to capture intra-page navigation (*i.e.*, scrolling). The PROTEUS implementation uses a heuristic search algorithm, hill-climbing in the space of personalized sites, and supports three search operators: elide content from a page, reorder elements on a page, and add a shortcut link to a page. Eliding content and reordering elements on a page improve the presentation of an individual page, while adding shortcut links improves the navigation through the site.

We ran a small user study of PROTEUS and found that personalization did improve web browsing, but we also found three weaknesses. First, adaptations made to the site that the user does not expect can be disorienting and not beneficial. For example, eliding large blocks of uninteresting content may seem useful (if the content is uninteresting, the user will not want to see it), but may also inadvertently remove visual landmarks that the user relied on for navigation. Second, naïve search through the space of personalized sites is computationally expensive, too expensive for acceptable server-side run-time performance. Third, although we collected a great deal of data, about many pages and many visitors, much of it was unused by PROTEUS in personalizing the site because the system considered improvements for each visitor in isolation of other users. Consequently, the learned user models were frequently unreliable, despite our having "more" data.

MINPATH. To address these weaknesses, we focused our efforts on a single personalization, adding shortcut links, and developed the MINPATH algorithm, which finds personalized shortcut links efficiently. Adding shortcut links to a page is frequently useful for visitors, helping them reach their destinations more quickly, and rarely decreases the usability of a page. Our work with

MINPATH seeks to answer the second primary question in the thesis: how can personalization be made efficient?

The MINPATH user model simplifies the content interest—in fact, it assumes all pages have the same intrinsic value—but treats navigation with more precision. In addition, MINPATH employs model-based clustering to group similarly-behaving visitors together (following the technique used in WebCANVAS [24]) and build models for these groups based on a larger volume of training data (*i.e.*, based on the training data from all the cluster's members).

MINPATH adds shortcuts to each requested page p by computing the *expected savings* that a shortcut from p would offer, as the product of the navigation effort the link would save and the probability the user wants to reach the shortcut destination. Intuitively, MINPATH examines every possible shortcut from p in a site and selects the m shortcuts with the greatest expected savings. Of course, MINPATH does not *actually* consider every possible shortcut, but instead uses a bounded graph traversal in the web site, starting at p, computing the expected savings for all pages visited. In traversing the web site, MINPATH is effectively emulating the actions of visitors, following the probabilistic model learned for the visitor's cluster. The traversal can be very fast, depending on the bound selected, and thus MINPATH can suggest shortcut links in a fraction of a second. Experiments with MINPATH found that it can identify shortcuts that would save visitors up to 40% of their navigation effort.

Relational Markov models. Although MINPATH avoids most of the weaknesses of PROTEUS, a problem persists: how to predict behavior on seldom- or never-before-visited pages. In moving from PROTEUS to MINPATH, we train user models with data from more individuals. Thus, if a particular visitor has not viewed a page before, but another (or many) other visitors in the same cluster have, then MINPATH can make a reliable prediction about navigation. But a more frequent case is one where few if any visitors in the same cluster will have viewed a page before. New pages are added to sites constantly, e-commerce and other sites produce potentially an unlimited number of new pages from database queries, and the Zipf-like distribution [51] of web page visitation all point toward a highly skewed distribution of training data—data is plentiful about some pages but very sparse about most others. As a solution to this problem, we extend Markov models to take advantage of relational structure, thus producing relational Markov models (RMMs). Relational Markov models allow states in the model (pages in a site) to belong to a relation and be described by a number of parameters, each parameter value being drawn from a hierarchically structured domain. Web sites frequently have a great deal of relational structure already available, in the form of a database schema or human-authored site schema describing the content. For example, an e-commerce site's pages naturally are divided into: main entry page, product description pages, pages describing groups of products (*e.g.*, by manufacturer), shopping cart pages, *etc.* The hierarchy of values of parameters affords a means of generating a hierarchy of states, and we use this state hierarchy to perform generalization from specific states to higher abstractions of states. For example, we can compute the probability of transiting from a *specific* music CD page to a page about the artist (*e.g.*, a page selling "Stunt" to a page about the Barenaked Ladies) by estimating that probability directly (with perhaps sparse data) and shrinking it toward the probability of a more general action (following the link to the artist page from *any* music CD page).

In preliminary experiments incorporating relational Markov models into MINPATH, we found that MINPATH can save users 50% to 80% more links than when using traditional Markov models. Further, MINPATH is able to suggest useful shortcut links on pages that neither the visitor nor anyone else has ever viewed before.

MONTAGE. Our final system for web personalization returns to the problem of improving the web experience as a whole, re-addressing our first thesis question (how automatic personalization can be made useful). In the MONTAGE system, we automatically generate personalized, dynamic portals of web content, based on the navigation behavior of each individual visitor. The target audience for MONTAGE is desktop users (as opposed to wireless PDA users for PROTEUS), and MONTAGE produces an aggregation of content—a *montage*—in the form of either a text- and links-only display or a graphically-rich amalgam of content and links. Also unlike PROTEUS, MONTAGE provides a personalized view of content from many sites, all at once, instead of only one site at a time.

MONTAGE uses the individual-user model introduced by PROTEUS, but records the *context* of each web request: the time and date of the request, the topic of recent browsing, perhaps the other applications the user has been using lately, *etc.* A web user views a montage as the starting page in the browser, or whenever the browser returns "home." Each time the user views the montage, MONTAGE captures the user's current context and computes the conditional probability of the user's interests and navigation given the context. Unlike PROTEUS and MINPATH, which have the user's

currently- and recently-requested pages as conditions for the user model, MONTAGE must rely only on the general context of browsing (because a request for a montage is the first request in a multi-site browsing session). In a fielded user study of the system, participants reacted favorably to personalizing content based on their past interactions and their current browsing context, and suggested a number of improvements to the usability of such a system.

1.4 Contributions of this thesis

This thesis makes the following contributions:

1. **Precise statement of web personalization problem.** We outline the space of web personalization systems and precisely state what web personalization means. Our statement extends and generalizes Perkowitz's statement of the adaptive web site problem [79, 80, 82].

2. **Framework for personalization as search.** We follow a similar approach to Perkowitz [79], but contribute:

    - A richer model of a web site; and
    - A principled model of web site utility for a visitor, based on value of content and probabilistic navigation.

3. **Efficient approach to finding shortcut links (the MINPATH algorithm).** Using a simplified but precise user model, we find high-quality shortcut links efficiently. We evaluate many probabilistic models for web navigation, including first and second order Markov models, clusters of models, and several hybrid models.

4. **Development of relational Markov models.** We provide a formal definition of what relational Markov models are, describe where they are applicable, and relate them to other probabilistic models. We show in experiments that relational Markov models outperform traditional Markov models for predicting web navigation.

5. **Evaluation of several systems (PROTEUS, MINPATH, and MONTAGE).** We build and evaluate three systems for personalizing web interactions, and conduct user studies for two of them.

1.5 Outline of this thesis

In Chapters 2 and 3, we refine the web personalization problem and detail our general approach. Chapter 4 presents PROTEUS, our personalizer that improves web browsing from wireless PDAs. We describe our MINPATH algorithm for finding shortcut links in Chapter 5. Chapter 6 develops our work on relational Markov models and uses RMMs in an experiment with MINPATH. In Chapter 7, we present our work on the MONTAGE personalized web portal system, including a user study. We conclude the thesis with a discussion of related and future work in Chapters 8 and 9 and summarize in Chapter 10. Appendixes A and B provide additional detail about our experiments and Appendix C describes many current web log mining techniques, including those we used in our work.

Chapter 2

WEB PERSONALIZATION

In this chapter we discuss the web personalization problem. We begin with a brief comparison of web personalization and adaptive hypermedia, and then describe the range of possibilities for personalizing the web experience—the "space of personalizations." We present formal models of a web site and of web navigation, and formally state the web personalization problem.

2.1 Web personalization and adaptive hypermedia

Adaptive hypermedia can be viewed as a generalization of adaptive or personalized web systems. The field of adaptive hypermedia has been researched since long before the advent of the Web, although it received a considerable influx of attention shortly thereafter. The goals of adaptive hypermedia are the same as those of adaptive and personalized web work: to "build a model of the goals, preferences, and knowledge of each user, and use this model throughout the interaction with the user, in order to adapt to the needs of that user" [19]. The particular techniques employed by adaptive hypermedia systems span a wide range, although they typically focus more on rule-based approaches (*i.e.*, the system automatically builds a user model, but a human designer specifies what action should be taken given the model). Brusilovsky has written several excellent surveys of the adaptive hypermedia field [16, 19], and we compare our approach to specific systems in Section 8.1.

2.2 The space of web personalization

Just how "personalized" a web experience is depends on many variables: whether it is personalized for a single user or for large groups of users; whether entire pages are adapted or only navigation between them changes; *etc.* Traditional web sites, delivering the same content to every visitor, occupy one point in this many-dimensional space. More modern approaches, such as customizable portal sites or sites offering visitors recommendations based on past browsing, fill other voids. We describe the dimensions of this space, and place these genres of web experience in this space, below.

2.2.1 *Scope of target audience*

The target audience of a personalized experience is the group of visitors who will see the exact same content as each other. Traditional web sites deliver the same content regardless of the visitor's identity—their target is the whole population of the Web. Personal portal sites, such as MyYahoo! [95] and MyMSN [70], allow users to build a personalized view of their content—the target here is the individual visitor. A middle ground between these extremes places visitors into one of a small number of archetypal groups (*e.g.*, a university may have different site views for current students, prospective students, and faculty).

Adapting content for the entire Web audience *en masse* has the advantage that the agent can spend more resources producing a high-quality personalization—the expenditure is amortized over a large audience. In comparison, personalizing for each visitor requires efficient (and consequently less sophisticated) algorithms, as the agent must adapt the content frequently. Perkowitz and Etzioni [82] refer to adaptation targeting a large audience (*e.g.*, the whole Web audience) as a *transformation*, and adaptation for individual visitors as *customization*.

2.2.2 *Scope of personalization*

We view the web experience as a hierarchical composition of many sub-experiences: a browsing session visiting many sites; many site-sessions at individual web sites; and individual page views within each site-session[1]. Each level of this hierarchy can be personalized and the nature of the adaptations depends on the level of decomposition.

A single page view can be personalized by altering how the content is displayed, or by suggesting navigation links to likely destinations. A site-session allows an agent to predict the set of pages the visitor will view and improve the navigation between them. Given many site-sessions over a period of time, the agent can also predict the content interests for each visit and both show more interesting content and avoid displaying redundant content over time. Finally, a personalizer adapting entire

[1]We refer to both site-session and browsing session simply as a session when its use is clear from context.

browsing sessions across many sites can leverage the commonalities between sites, aggregating similar content and providing navigation between sites (effectively creating personalized portals).

2.2.3 *Location of user modeling*

Collecting information for the user model can happen at one or many locations. At the browsing client, the agent can collect detailed information about the complete interaction between user and web content: the pages viewed, how long each page is viewed (often called the “dwell time”), the navigation between pages, the navigation on a single page (*e.g.*, scrolling through a page), whether a user records a bookmark for the page, *etc.* Unfortunately, at the browsing client, the personalizer has access to only the current user's model, and not the models of other users. Web users are reluctant to share with others such detailed information about their own browsing, so the personalizer must base its adaptations on only the current-user model.

At the opposite end of the spectrum from the client, an individual web server can collect statistics about accesses to all of its pages by all the users on the Web. The server knows exactly when each page was requested, and everyone who viewed it. However, the server cannot observe these visitors' behavior at other sites, nor can it observe how the visitors actually interacted with each page of content (*i.e.*, how much they scrolled through it, if it was displayed in the browser window for long, *etc.*)[2]. In addition, information collected by the server may be obscured by third-party agents acting on behalf of one or more visitors (*e.g.*, a web proxy cache [51] effectively masks many users' accesses to a web page as a single access).

In between these extremes, an intermediate proxy can observe many users' behaviors browsing many web sites, with the same detail as the web server. That is, an intermediate proxy will know that its many users all view the same set of web pages, but will not know, for instance, how much the users have scrolled through the pages, or for how much time the content held the attention of the user. A model built by a proxy trades off the completeness of the model—knowing all the browsing for an entire population—with the detail of the model—knowing the minutiae of the interaction (scrolling, dwell times, *etc.*).

[2]There are some “tricks of the trade” that allow a web server to garner more detailed information about how its content is used by visitors; see Appendix C for more details.

2.2.4 Location of personalizer

Likewise, the personalizer itself may operate at a number of different locations. Three locations naturally arise—at the client, at the server, and at an intermediate proxy—with implications for what personalizations are possible with each choice. At the client or an intermediate proxy, personalization over any scope of time is possible, from individual page to many sessions across many sites. At a web server, personalization may occur only for each individual site—spanning sites is not possible. There also is a corresponding trade-off in location of personalization and the data available for modeling users, as discussed earlier (*i.e.*, a client-side personalizer usually cannot access server-side user models).

2.2.5 Visitor's goal of web interaction

To improve the web experience, we must have some idea of what motivates each visitor to interact with a web site or web page. On one extreme is idle browsing—"surfing." The visitor is simply viewing content that satisfies his or her interests, be they short- or long-term. For this motivation, personalizing the site by adding content that generally interests the visitor would be useful. The time the user spends viewing pages is not as important, so minimizing navigation time is only a secondary concern. Of course, helping the user in finding interesting content is useful.

On the other extreme are users seeking a particular bit of information, such as the location of a class at a university or the cost of an airline ticket on a travel site. We call this behavior goal-directed browsing, and each session of such browsing an "information sortie."[3] Information sorties are frequent in mobile web browsing (from wireless PDAs or cell phones). The visitor's behavior is single-minded and not interested in diversions, even for other pages that match long-term interests. Appropriate personalizations for information sorties would reduce visual clutter by eliding uninteresting content, and directly link to the information sought, by predicting the user's current goal. Scenario Two (Chapter 1) gave an example information sortie for the box score of a baseball game (and the consequences of not personalizing the experience).

[3]Thanks to Adam Carlson for suggesting this term.

2.2.6 Model of web site

A dimension of more practical concern is how the web site is conceptually modeled by the personalizer. A common representation is as the directed graph of web pages and hypertext links often called a *site graph* or *web graph*. A weakness of the site graph is that presentation of content on pages and the conceptual relation between elements of content is conflated—both are represented as the same HTML. Instead, we can represent the site's data selection, navigation, and presentation separately, as suggested by systems such as STRUDEL [40] or TIRAMISU [7]. With this decomposition, we can personalize each aspect of the site independently; for example, by adapting the queries that select what data to display, or by changing a single generic HTML template that consequently adapts many pages in the same, consistent manner. This representation of the content selection and navigation is called the *site schema*.

Another model, related to the site graph, augments the nodes in the model with the relational structure that they share. For example, many pages at an e-commerce site are product description pages and are described by the same set of attributes (product name, manufacturer, *etc.*); still other pages are the main entry page, shopping cart pages, and indexes of products (by manufacturer, by product type, *etc.*). This relational information allows the personalizer to make generalizations about how the site is used, and to apply adaptations observed for specific pages (*e.g.*, a particular product page) to other, related pages (other product pages).

2.2.7 Personalization agent

Although we have referred to a personalization agent in this section, the act of personalization can be performed by a human webmaster, a human web user, an automated process, or any combination thereof. A human webmaster likely has the most accurate conception of the content available, but is expert in only one site. Also, the human webmaster is a bottleneck for performance—the webmaster cannot personalize more than a few score pages per day.

The human visitor, on the other hand, has the most accurate conception of his or her interests, both at a single site and across many sites. The user is best positioned to adapt content to maximize its value, but is likely not an expert in the content at any site. Moreover, users tend to be loath to expend any effort to customize their own views of the Web, even if it could directly benefit them (*e.g.*, in a survey we conducted for our MONTAGE work [6], three-quarters of respondents organized their bookmarks less than once per month).

The automated personalization approach removes the burden of adaptation from both the webmaster and the visitor. The automated approach can adapt content per visitor, per page impression, and can leverage models of many visitors and many sources of content. Unfortunately, the automated approach is only as accurate as the models it can learn, and the inference of visitors' true interests from their observed behavior is necessarily imperfect. A mixed initiative approach would perhaps be the best of all worlds: personalizations would be generated automatically, with the user or web designer optionally providing guidance to clarify preferences or incorrectly modeled details.

2.2.8 Summary

Table 2.1 summarizes the dimensions of web personalization. Traditional web sites, that offer a static view of content for all visitors, occupy one point in this space: the target audience is the whole web, the scope is many site-sessions, "personalization" happens at the server side, and changes are made by the human web designer. Sites such as MyYahoo! target individual visitors, over the course, of many site-sessions, offering server side personalization, but must be maintained by the web visitor.

This thesis explores two particular points that are somewhat distant from traditional web sites. One point is a client-based automated personalizer, for which we developed and experimented with PROTEUS and MONTAGE[4]. PROTEUS adapts content for goal-directed site-sessions and MONTAGE improves information browsing across browsing-sessions. Both use the site graph model and target individual users.

The other point is a server-side personalizer, for which we designed the MINPATH algorithm and developed relational Markov models. The server-side implementation allows the personalizer to take advantage of the greater amounts of data available there—describing users and the site itself. The purpose of MINPATH is to improve goal-directed browsing for individuals and groups of related users, and we experiment with representing the site as a site graph and as a site graph with relational

[4]These personalizers are actually implemented as proxies for convenience, but take no advantage of other-user models when adapting content.

Table 2.1: **Space of web personalization.**

Dimension	Range of values
Scope of target audience	Whole Web Group of related users Individual user
Scope (in time) of personalization	Web session Many site-sessions Single site-session Single page view
Location of user modeling	Client Intermediate proxy Server
Location of personalizer	Client Intermediate proxy Server
Visitor's goal of web interaction	Browsing for long-term interests Browsing for short-term interests Goal-directed browsing ("information sortie")
Model of web site	Directed graph of pages and links (site graph) Separate data selection, navigation, and content presentation (site schema) Directed graph of pages and links, and relational structure of pages
Personalization agent	Human webmaster Human web visitor Automated Mixed-initiative

information.

Other points in this space have been explored either as deployed applications or in the research community. Personalized portal sites, such as MyYahoo! [95] or MyMSN [70], are user-driven (each visitor must build his or her own personalized portal) and limit the adaptations to a set of webmaster-defined modifications. Perkowitz and Etzioni's adaptive web sites [82] use an automated approach that transforms individual web sites (*i.e.*, personalization and modeling occur at the site) for the entire Web audience. Their adaptive web sites anticipate users have short-term interests in a particular genre of pages, and thus they build "index pages"—pages of links to pages relating to a common concept. Web agents such as Letizia [65] or WebWatcher [57] personalize pages for individual visitors by adopting a site graph model and augmenting the page representation with additional features, such as text keywords. User modeling and personalization happens either at the client site (Letizia) or the server site (WebWatcher). We review a wider selection of related work in more detail in Chapter 8.

2.3 A formal model of web personalization

In this section we present a precise statement of the web personalization problem. As we mentioned earlier, personalizing a web site is very similar to building an adaptive web site [82] or an adaptive hyperspace [17], but specifically targets the Web and adapts to individual users or classes of users; we highlight other differences in our discussion. We begin by defining several terms.

2.3.1 Definitions

We model a web site as a directed graph whose nodes are pages and whose directed arcs are hypertext links. More precisely, a web site W is a triple $(\mathcal{P}, \mathcal{L}, \Pi)$, where $\mathcal{P}$ is a set of pages, $\mathcal{L}$ is a set of hyperlinks, and Π is a set of partitions of the pages in $\mathcal{P}$. A page $p \in \mathcal{P}$ is a hierarchical aggregation of content that appears in a web browser given a particular URL (Uniform Resource Locator). The hierarchy of web content is meant to capture the visual aggregation of content on the display. For example, a page is often composed of a navigation bar along the top or left side, and a primary content region in the center; the primary content region may be divided by horizontal lines to create smaller regions of content; *etc.* In practice, this hierarchy typically follows the tree of HTML tags in the source of the page. Page p is thus represented as the root of this hierarchy, and is a *content node*. A content node c is a pair (C, B) where C is a sequence of children $\langle c_1, \ldots, c_k \rangle$ of c, and B is a behavior that c imparts on its children. The elements of C may be plain text, embedded objects (such as images), or (recursively) content nodes. The behavior B is the action that affects the human-viewable content. For example, if c were a "`<strong>`" node, then B would render its children in boldface; or if c were an "`<a>`" node, then B would render its children as a hypertext link.

A link $l \in \mathcal{L}$ is a triple (p_s, p_d, a) where p_s is the source page (*i.e.*, the page on which the link

appears), p_d is the destination page[5], and a is the anchor text. When the anchor text is unimportant or clear from context, we may abbreviate $l = (p_s, p_d, a)$ as $p_s \rightarrow p_d$.

A partition $\Upsilon \in \Pi$ expresses some set of relations about the pages in W. For example, a partition may express the directory relation by segregating pages by their "top-level" directory (*e.g.*, `/homes/corin/research/` and `/homes/weld/index.html` would be in the group `/homes` and `/education/course-webs.html` would be in the group `/education`). Alternatively, a partition may express the conceptual type of each page (*e.g.*, personal home pages versus course web pages versus research group pages). Υ may also be hierarchical: for example, a hierarchical organization of the pages according to several levels of directories. Many such relational structures may be relevant to W, so Π is often a set with many elements.

A visitor is any agent that requests pages from a web site. We are particularly interested in modeling human visitors, although we note that a substantial fraction of web requests is made by robots[6] (for example, during February 2002, as much as 20% of the requests to `www.cs.washington.edu` were made by robots). A visitor v is represented as a pair $(\mathcal{H}, \mathcal{D})$ whose elements are, respectively, the visitor's history of page requests and the visitor's demographics. A single request r_j is a tuple (p_s, p_d, ϕ) containing the source page p_s[7], destination page p_d, and context ϕ of the request. The context embodies the time of the request, the device used to make the request, the topic of recent browsing, *etc.* When clear in our discussion, we will use simply r_i to refer to the page p_d requested in r_i. The demographics $\mathcal{D}$ is an n-tuple of values that embody all the available information about the visitor external to the web experience: the visitor's age, gender, city of residence, annual income bracket, *etc.*

By themselves, requests in $\mathcal{H}$ are little more than a morass of low-level system activity traces, but they can be refined into structure more useful for personalization. In particular, see Appendix C for common practices in cleaning and processing these logs. The end result of this process is a set of *sessions* or *trails*. A session is sequence of page requests made by a single visitor that is coherent in time. That is, each subsequent request is made within some fixed time window of the previous

---

[5] p_d may also be the distinguished page external which represents any page outside the current web site.

[6] A *robot* is a software agent that requests web pages automatically. Most common are web *spiders* that crawl the Web collecting pages for indexes such as Google [53].

[7] The source p_s may also be the distinguished page external, which represents any page outside the current web site.

request. A trail is a session that is also coherent in space: there exists some link between each pair of subsequent pages. More precisely, if we denote the time of request r_i as $\mathsf{time}(r_i)$, then a sequence $S = \langle r_0, r_1, \ldots, r_n \rangle$ is a session if and only if:

$$\forall\, i, 0 \leq i < n, \mathsf{time}(r_i) \leq \mathsf{time}(r_{i+1}) \leq \mathsf{time}(r_i) + \textit{timeout}$$

and S is a trail if and only if S is a session and

$$\forall\, i, 0 \leq i < n, r_i \rightarrow r_{i+1} \text{exists}$$

The *length* of a session or trail is n, the number of links followed. From the perspective of the personalizer watching a visitor's behavior midway through a session or trail, only a *prefix*, $\langle r_0, \ldots, r_i \rangle$ has been observed. The session or trail *suffix*, $\langle r_{i+1}, \ldots, r_n \rangle$, can only be hypothesized.

In summary we have:

$W =$	$(\mathcal{P}, \mathcal{L}, \Pi)$	A web site is a directed graph of pages and links with structure describing the pages
$l =$	(p_s, p_d, a)	A link has a source, destination, and anchor
$p_i =$	c_i	A page is its root content node
$c_i =$	$(\langle c_{i1}, \ldots, c_{ik} \rangle, B)$	A content node is a sequence of children and node behavior; or
$c_i =$	*text*	A content node is plain text; or
$c_i =$	*object*	A content node is an embedded object
$V =$	$(\mathcal{H}, \mathcal{D})$	A visitor is a history of requests and set of demographics
$\mathcal{H} =$	$\langle r_0, \ldots, r_n \rangle$	A history of requests is a sequence
$r_i =$	(p_s, p_d, ϕ)	A request has an originating page p_s, destination page p_d, and context ϕ

2.3.2 Evaluation

To find the best personalized site, we require an *evaluation function* $\mathcal{F}(W, p, \phi, v) \mapsto \mathcal{R}$ that measures the quality of a personalized site W for a particular visitor v and requested page p given the

Given:
- A web site $W = (\mathcal{P}, \mathcal{L}, \Pi)$,
- A set of visitors $V = \{v_0, \ldots, v_m\}$,
- A distinguished visitor $v_i = (\mathcal{H}, \mathcal{D}) \in V$,
- An entry page $p \in \mathcal{P}$,
- An evaluation function $\mathcal{F}(W, p, v)$;

Output:
- A web site $W' = (\mathcal{P}', \mathcal{L}', \Pi')$ that maximizes $\mathcal{F}(W', p, v)$.

Figure 2.1: **Web personalization problem statement.**

current browsing context ϕ. The quality of a site is a complex function of many things: the visitor's desire to find relevant information quickly; the cognitive cost imposed on the visitor by presenting a view of the site that is potentially different from the visitor's expectation; the site administrator's desire to deliver content efficiently; the content owner's desire to sell products or influence the visitor; *etc.* In our work, we concentrate on only a subset of these aspects, namely, the value of the site from the point of view of the visitor. In particular, we will not directly model the cognitive load required when viewing a page, but instead will encode the trade-off between predictability and improvements to the interface in the bias of our algorithms.

2.3.3 Problem statement

We can now state precisely the web personalization problem; the definition appears in Figure 2.1.

We call an agent that generates a personalized web site a web site personalizer. A web site personalizer may run on the server itself, on the visitor's browsing client, or at an intermediate proxy. Note that the web personalization problem differs from the adaptive web site problem defined by Perkowitz and Etzioni [82] by optimizing the content for a distinguished visitor v_i, based on context ϕ and using a richer model of web site and evaluation. In the remainder of this dissertation we will explore several forms of web site personalization in detail. The next chapter presents our general approach, followed by chapters discussing our implemented systems.

Chapter 3

PERSONALIZATION AS SEARCH

In this chapter, we present our PROTEUS framework for personalizing web sites. We frame the task as a search problem, similar to that proposed by Perkowitz and Etzioni [79, 82]. Within this framework, a personalizer follows the two-step approach outlined in Section 1.2, separating user modeling and adaptation. In essence, our web site personalizer performs a search through the space of possible web sites. The initial state is the original web site of unadapted pages. The state is transformed by any of a number of adaptation functions, that can create pages, remove pages, add links between pages, *etc.* The value of the current state (*i.e.*, the value of the web site) is measured as the expected utility of the site for the current visitor and browsing context. The search continues either until no better state can be found, or until computational resources (*e.g.*, time) are exhausted.

3.1 State representation

Each state in our search space $\mathcal{S}$ is an entire web site, W; thus, $\mathcal{S} = \{W_0, W_1, \ldots\}$. Although an actual implemented system (such as ours, as we discuss later) may choose to adapt only a single page at a time, we model the entire web site to allow adaptations to be made anywhere in the site.

3.2 State evaluation

We estimate the quality of the personalized web site as the expected utility of the site from the point of view of the requested page: $\mathcal{F}(W, p, \phi, v) = E[U_{v,\phi}(p)]$. Intuitively, the expected utility is the sum of the utility the visitor receives by browsing each page in the site, discounted by the difficulty of reaching each page. For example, following a link at the top of the current page may not be difficult, but reaching a page many links away will require scrolling (to find the links) and waiting for intermediate pages to download, which can be particularly lengthy over a wireless network. The graph nature of a web site naturally suggests a recursive traversal to evaluate the site, starting from

the current page. At each step in the traversal (*i.e.*, at each page in the site), we define the utility of the page as the sum of its *intrinsic* utility—the utility of the page, in isolation—and the *extrinsic* utility—the utility of the linked pages[1]. We make these concepts more precise below.

3.2.1 Web site model for evaluation

We transform the search state model slightly to make evaluation easier. We observe that, while adaptation requires detailed internal structure of each web page, an evaluation function that is based on a human visitor's view should not be exposed to this internal structure, but should see only a linear sequence of text and links. Thus, instead of using the tree-based model, we use only the *leaves* of the tree in their left-to-right order.

The leaves of a page p_i and their ordering impose a linearization of the content, which we subsequently decompose into a sequence of "screens" $\langle s_{i0}, \ldots, s_{im} \rangle$, each of which represents the web content that can be seen in one window of the visitor's browser. A single screen s_{ij} is composed of web content (*i.e.*, the text and graphics that are displayed), which we denote as T_{ij}, and a set of links $l_{ij1}, \ldots, l_{ijk}$ that appear on the screen. In summary:

p_i	$= \langle s_{i0}, \ldots, s_{im} \rangle$	A page is a sequence of screens
s_{ij}	$= (T_{ij}, \{l_{ij1}, \ldots, l_{ijk}\})$	Each screen contains web content and links

3.2.2 Expected utility

Because our goal is to maximize *expected* utility, we must be able to calculate the expectation that the visitor will view any given piece of content in the site. To this end, we explicitly model the visitor's navigation through the site. We consider two alternative but related models of visitors' action. In both, we assume the visitor has only a fixed set of *navigation actions* at his or her disposal. Specifically, if the visitor is at screen s_{ij}, then the set of available actions is[2]:

$$A = \{a_{\mathsf{scroll}}, a_{l_{ij1}}, \ldots, a_{l_{ijk}}\}$$

[1] In many ways, the intrinsic and extrinsic utilities are analogous to the authority and hub weights, respectively, from Kleinberg's work [59].

[2] The visitor can also simply stop browsing at the site. We model this action by creating a special stop page in W to which every other page links. When a visitor stops browsing, he or she implicitly follows the link to the stop page.

That is, the visitor may: scroll down to the next screen (assuming that s_{ij} is not the last screen of the page); or follow any link that appears on the screen.

The two models differ in whether they assume the actions are mutually exclusive. In the first model, which we call the *session model*, actions are not mutually exclusive—the visitor may perform a number of actions at any screen (*e.g.*, follow three different links (each time returning to this screen with the browser's "back" button) and scroll to the next screen of content). Thus, in this model, we maintain independent probabilities that the visitor will take each action, and these probabilities will generally not sum to one. This model is appropriate when the personalizer will adapt the site infrequently (*e.g.*, once per day, or once per session), or when the sequence of previously-visited pages is unimportant.

In the second model, which we call the *trail model*, the actions *are* mutually exclusive—the visitor may take only one action. Further, this model makes the simplifying assumption that all the content on a page displays in one screen; thus, this model ignores scroll actions (*i.e.*, it considers only navigation actions). The trail model is appropriate if the personalizer adapts the content after each request—that is, if the personalizer assumes the visitor will not take multiple actions without giving the system another opportunity to adapt the site or current page. This model effectively calculates a probability for each possible trail of page requests the visitor may take (given the recently-observed trail prefix), and is thus best suited for personalizers that require or can take advantage of this sequence information.

Given a model of navigation (either the session model or the trail model), we can formulate the expected utility of the site. Recall from Section 2.3.3 that the evaluation function $\mathcal{F}$ takes as input the web site W, the page requested p, the browsing context ϕ, and the current visitor model v, and calculates the quality of the site. If we let $\hat{W}$ and $\hat{p}$ be the personalized versions of W and the requested page p and $U_{V,\phi}(\hat{p})$ be the utility of $\hat{p}$ for visitor v, then:

$$\mathcal{F}(\hat{W}, \hat{p}, \phi, v) = E[U_{V,\phi}(\hat{p})]$$

In other words, as we observed earlier, the evaluation of $\hat{W}$ is the product of a recursive traversal through the site, and this equation states that $\hat{p}$ is the root of that recursion. Similarly, because only the first screen of $\hat{p}$ is initially visible to the visitor, we calculate the expected utility of $\hat{p}$ (or any p_i,

in fact) as the expected utility of its first screen:

$$E[U_{V,\phi}(p_i)] = E[U_{V,\phi}(s_{i0})]$$

The expected utility of a single screen s_{ij} is the sum of its expected intrinsic and extrinsic utilities:

$$E[U_{V,\phi}(s_{ij})] = E[IU_{V,\phi}(s_{ij})] + E[EU_{V,\phi}(s_{ij})]$$

The intrinsic utility of a screen measures how useful the screen's content is toward fulfilling the visitor's information goal, in isolation of the rest of the web site. Typically, the intrinsic utility will depend on the visitor model—the past history and demographics. A more detailed description of intrinsic utility depends on particular assumptions regarding visitor interests and goals; see Chapter 4 for a discussion of the method we used in PROTEUS and Section 7.4 for our approach in MONTAGE.

The extrinsic utility, on the other hand, measures the value of a screen's connections to the rest of the web site. As we noted earlier, the visitor may reach other parts of the web site by taking navigation actions from the current screen. Associated with each of these actions is a probability that the action will be taken, denoted by $P_V(\text{action}|\phi)$. In addition, actions may impose a cost to the visitor (*i.e.*, a negative utility); these costs are γ_s and γ_l for scrolling and following a link, respectively. These costs subtract directly from the expected utility of the action, and represent the cost to the visitor (in time, money, *etc.*) of taking the action. In summary, if we let $\mathsf{dest}(l_{ijk})$ be the destination page of link l_{ijk}, then the extrinsic utility of screen s_{ij} is a sum weighted by probabilities:

$$\begin{aligned} E[EU_{V,\phi}(s_{ij})] \quad = \quad & P_V(\mathsf{scroll}|\phi)(E[U_{V,\phi}(s_{i,j+1})] - \gamma_s) + \\ & \sum_k [P_V(l_{ijk}|\phi)(E[U_{V,\phi}(\mathsf{dest}(l_{ijk}))] - \gamma_l)] \end{aligned} \tag{3.1}$$

We can see that Equation 3.1 introduces the recursive component of the evaluation function, by referencing the utility of other pages and screens. The recursion is halted when the expected utility of a screen or page is less than the cost of reaching that content (*i.e.*, when $E[U_{V,\phi}(s_{i,j+1})] < \gamma_s$ or $E[U_{V,\phi}(\mathsf{dest}(l_{ijk}))] < \gamma_l$).

The equations given above and a formula for intrinsic utility completely determine the utility of an adapted page $\hat{p}$. However, the equations given, evaluated verbatim, would not be computationally tractable—they call for a screen-by-screen decomposition of potentially every page in the entire web site. For small sites, this fine-grained analysis may be possible, but many sites have hundreds if not thousands of static web pages, as well as potentially limitless dynamic web pages—far too many to examine individually. Fortunately, the evaluation can be made computationally much simpler with the aid of a few assumptions and simplifications. We describe those conditions and how to take advantage of them as we present our implemented systems.

3.3 Search control

Any state-space search method is, in principle, applicable to our problem, but we found that a simple hill-climbing search control method worked well (see Table 3.1). Throughout the search, the personalizer maintains the best-known state so far, *BestState*, and a current search seed *SearchSeed*. At each iteration in the search, the personalizer produces all states one "step" away from the search seed by applying the search operators (described below), and replaces the search seed with the best of these new states. The search stops after a fixed number of iterations and the personalizer returns the best state found.

3.4 Search operators

The search operators make individual adaptations to the web site. For example, an operator may create a new page, or add a link between existing pages, or embed a block of content from one page into another. These adaptations may affect any part of the site (*i.e.*, a single page or link, or many pages), but in our work we concentrate on operators that influence only the requested page p. The specific search operators we examine are detailed as we present each system in this dissertation.

Table 3.1: **Search control.** The Search function takes a web site, visitor model, browsing context, and requested page as input and returns the personalized site $\hat{W}$ that maximizes $\mathcal{F}(\hat{W}, \hat{p}, \phi, v)$.

Inputs:

W	*Unadapted web site*
v	*Visitor for whom to personalize*
ϕ	*Context of current browsing session*
p	*Next page requested by* v

```
Search(W, v, φ, p)
  BestState ← W
  BestValue ← F(W, p, φ, v)
  SearchSeed ← W
  For K iterations:
    S ← Generate new states from SearchSeed
    s ← State in S with greatest F(s, p, φ, v)
    SearchSeed ← s
    If value of s > BestValue
      BestState ← s
      BestValue ← F(s, p, φ, v)
  Return BestState
```

Chapter 4

THE PROTEUS ARCHITECTURE

This chapter presents our first implemented system for web site personalization: PROTEUS[1]. PROTEUS follows the search framework described in the previous chapter, although we describe a few implementation-specific issues here. The goal of PROTEUS is to improve the mobile web browsing experience by personalizing content. Visitors browse the Web from mobile clients, such as wireless Palm VIIs, and PROTEUS acts as an intermediary between the visitor and the Web. PROTEUS treats the entire web as a single "site," in the sense that PROTEUS builds a single model based on visitors' behavior at all sites viewed. We evaluated PROTEUS with a small field study and present those findings in Section 4.5.

PROTEUS follows the architecture presented in Chapter 3, and we present here the details of the system left unspecified by the high-level architecture. PROTEUS employs the search control described earlier, and uses expected utility (with a session navigation model) to evaluate each state. Figure 4.1 shows the architecture of our implementation. We next introduce the search operators, followed by details of how we calculate the intrinsic utility of content for our visitors.

4.1 Search operators

PROTEUS employs three search operators to adapt content on the site: elide-content, swap-siblings, and add-shortcut. These operators manipulate the content on the requested page, either eliding or rearranging content, or adding navigation links. We limit the elision and rearranging adaptations to operate only on select regions of the page called *content blocks*, to ensure the operation does not change the meaning of the content. The next section describes these content blocks, and we follow by giving more detail about each adaptation.

[1]Much of the material presented in this chapter was published in the 10th International Conference on the World Wide Web [4].

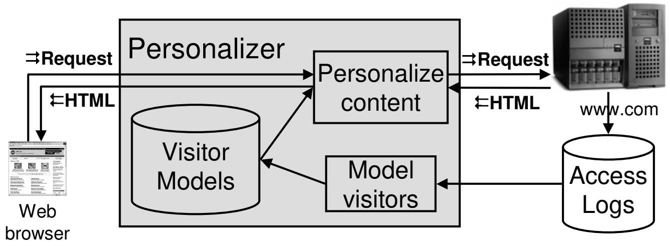

Figure 4.1: **PROTEUS architecture.**

4.1.1 Content blocks

A *content block* corresponds to a conceptually coherent region of content on the page that an adaptation may manipulate. For instance, content in a `<div>` tag can be manipulated, as can entire lists (`<ul>`, `<ol>`), but a single emphasized word (`<em>`) cannot. In addition, blocks can be nested. Figure 4.2 shows a typical decomposition into these content blocks, for Yahoo!'s finance portal (`finance.yahoo.com`). PROTEUS uses the following heuristics to identify content blocks:

- A paragraph of text. Regions of text in the HTML file that contain only small spans of markup (*e.g.*, links, emphasized words, *etc.*) are aggregated into paragraphs even if no `<p>` tag existed in the original document.

- An HTML header (`<h1>`, ..., `<h6>`) and all the content following it until the next header of the same "level" or higher. For example, an `<h2>`, all the content following it, including perhaps `<h3>` headers, until the next `<h2>` or `<h1>`. This block also is limited in scope by the header tag's parent node in the HTML parse tree.

- An HTML table (`<table>`).

- A single row in a table (`<tr>`)[2].

[2]Note that, although the *contents* of a table row or cell may be adapted, we never remove the row itself, to ensure that the table remains formatted correctly.

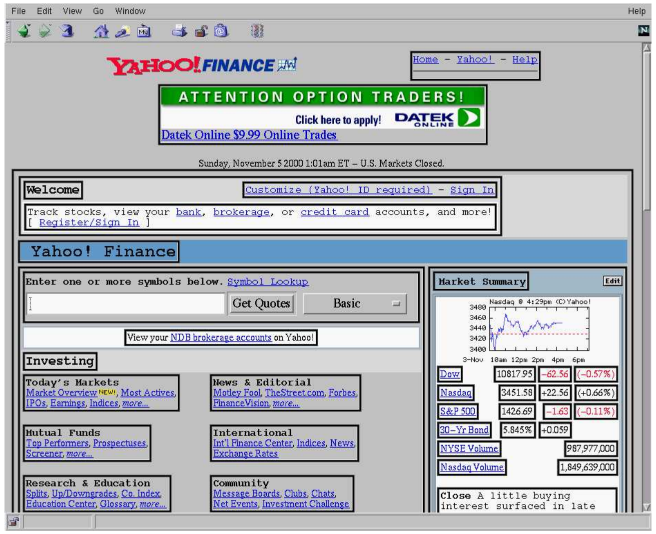

Figure 4.2: **Content blocks.** Solid black borders outline blocks of content that may be manipulated by search operators. Note that these blocks may be nested.

- A single cell in a table (<td>).
- A list (<ul>, <ol>, or <dl>).
- A list item (<li> in ordered or unordered lists, or both the <dt> and <dd> from a definition list).
- A division (<div>).
- A form (<form>).

4.1.2 elide-content *operator*

The elide-content operator replaces a subtree of $\hat{p}$ with a link to the original content in a fashion similar to Digestor [13] (Figure 4.3). Content elision trades off screen space and visual complexity of $\hat{p}$ for the cost the visitor may incur to view the elided content (*i.e.*, to follow the link). Note that elided content is still available—the visitor can always reach the original content by following the replacing link. The anchor of the newly created link is derived from the first several words of the elided content; PROTEUS limits the anchor length to three words of the first HTML tag in the block. This choice of anchor is useful when the content block starts with an HTML header tag (hence, the anchor text would be the header text), but can perform poorly if, for example, the block contains an unrelated advertisement near the top.

4.1.3 swap-siblings *operator*

The second operator, swap-siblings, permutes two subtree siblings in $\hat{p}$'s hierarchy. The result of swap-siblings is to swap the positions of the subtrees' content in the rendered page: if c_l is the left-sibling of c_r, then swapping c_l and c_r will place the content of c_r above c_l in the browser window. Like elide-content, swap-siblings is allowed to swap only preidentified content blocks. In addition, swap-siblings may not swap any blocks that are implicitly ordered, such as ordered lists (<ol>) or text (<p>). Figure 4.4 shows an example of swap-siblings.

4.1.4 add-shortcut *operator*

The final operator, add-shortcut, creates a new link from $\hat{p}$ to some other page in W (Figure 4.5). Because W may have an arbitrarily large number of pages, add-shortcut does not consider *all* new links from $\hat{p}$. Instead, add-shortcut considers only those pages that can be "reached" by following at most k links from (the original version of) $\hat{p}$. Thus, add-shortcut may create a link from $\hat{p}$ to another page that effectively "shortcuts" a longer path of links. For instance, if the visitor previously followed the path $p \rightarrow p_a \rightarrow p_b \rightarrow p_c \rightarrow p_d$, add-shortcut may create a link directly from $\hat{p}$ to p_d. Furthermore, PROTEUS places the new link $\hat{p} \rightarrow p_d$ next to the link $\hat{p} \rightarrow p_a$, based on the assumption that, if the visitor previously reached p_d by first going through p_a, and the visitor wants to find p_d again, the visitor will look toward the link to p_a first. Of course, PROTEUS knows to place

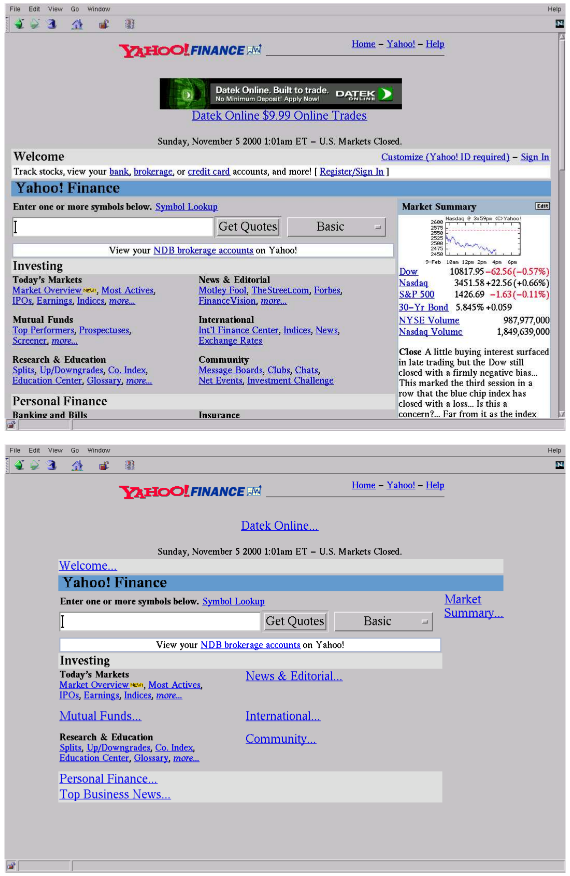

Figure 4.3: **Elided content.** At the top is an unadapted web page. Beneath a number of blocks of content have been elided and replaced with hypertext links.

Figure 4.4: **Swap content.** The top two articles on the unmodified Slashdot web page (top) have been interchanged (bottom).

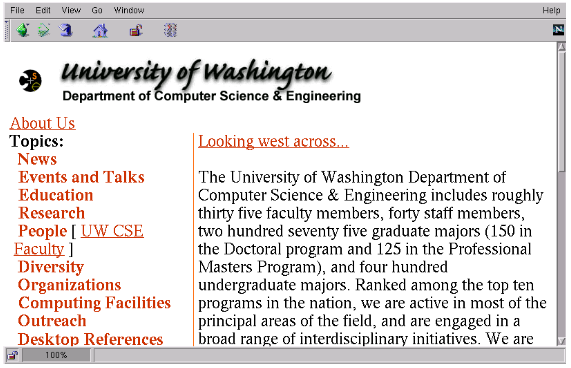

Figure 4.5: **Shortcut links.** PROTEUS has created a new shortcut link on the this page: "UW CSE Faculty". The link is placed near the most common starting point of the path it shortens.

$\hat{p} \rightarrow p_d$ near $\hat{p} \rightarrow p_a$ only if the visitor actually took the path $p \rightarrow p_a \rightarrow \cdots \rightarrow p_d$ before. If the visitor has not established this path before, then PROTEUS places the link based on the paths *other* visitors at the site have taken. That is, if there are many paths from $\hat{p}$ to p_d, PROTEUS will choose the most popular path and place the shortcut link near the first step along that path. The anchor text of the link is chosen as the destination's `<title>` (if it exists), the first `<h1>` (if it exists), or the URL.

4.2 Web site evaluation

As we stated in the previous section, we evaluate the search states by calculating the expected utility of the web site for the visitor (PROTEUS ignores the visitor browsing context). The previous section described the framework for this evaluation, but left some details to the actual implementation of the system. Below, we describe these choices in PROTEUS.

4.2.1 Probabilities and costs of navigation actions

PROTEUS employs the session navigation model, as described in Section 3.2.2. PROTEUS estimates the probabilities that the visitor takes each of the actions available at screen s_{ij} by measuring the frequency with which the visitor took the action in the past. For example, the probability that the visitor follows a link $p_s \rightarrow p_d$ is the quotient of the number of sessions in which the visitor viewed p_d sometime after p_s divided by the number of sessions in which the visitor viewed p_s (*i.e.*, the probability of the link $p_s \rightarrow p_d$ is simply the probability the visitor will reach p_d sometime after p_s).

The probability for scrolling is derived empirically and is held constant. PROTEUS presently uses a probability of 0.85 that the visitor will scroll to the next screen, although this number can be determined by collecting more information from the visitor's browser (for instance, dividing up every conceptual page of content into HTML pages that fit exactly within one screen, and counting how often the visitor requests each additional screenful of content; such a technique is used by the Daily Learner [15]).

Actions impose a cost on the visitor. Through empirical evaluation, we set the cost of scrolling, γ_s in Equation 3.1, at 0.01 and the cost of following a link, γ_l, at 0.05. These values tended to work acceptably in practice, although we found that our results were largely insensitive to their exact values. These costs subtract directly from the expected utility of the distal page, which ranged in value from 1.0 or 2.0 all the way up to over 100.0, with the discrimination between high-quality and low-quality pages at different levels for different users. In practice, the dominant factor in the expected utility equation is the product of probabilities of taking chains of actions. That is, for all but the most probable links the visitor would follow, the contribution of a remote page to expected utility is already vanishingly small, irrespective of the cost of following the link. The situation is similar for scrolling through the screens of the current page.

4.2.2 Intrinsic utility

Our implementation measures the intrinsic utility of a screen as a weighted sum of two terms, which relate to how the screen's content matches the visitor's previously viewed content, and how frequently the visitor viewed this screen. If T_{ij} is the viewable content on screen s_{ij}, then:

$$IU_V(s_{ij}) = \omega_{sim} \cdot sim_V(T_{ij}) + \omega_{freq} \cdot freq_V(s_{ij}) \tag{4.1}$$

$sim_V(T_{ij})$ is the similarity between T_{ij} and a model representing the visitor's web content interest. In PROTEUS, we are concerned only with the textual content of the page, and ignore any graphical elements. We model the visitor and the requested content as *word vectors*, vectors in an n-dimensional space, with each word that appears on any web page as a different dimension. We omit words from a fixed set of common "stop words," such as 'I', 'you', 'the', 'an', *etc.* We scale both vectors according to a TFIDF scheme [87], which weighs words proportionally to how often they appear in the document (in this case, the visitor or document model), and inversely proportionally to how often the words appear in *any* document. We explain this approach in more detail next.

Let $w_{T_{ij}}$ be the word vector for content T_{ij}, and consider a particular word k. The value of $w_{T_{ij}}(k)$ is the number of times word k appears anywhere in T_{ij}, divided by the number of *other pages* on which word k appears. For instance, if the word "Microsoft" appeared eight times in T_{ij}, but appeared in a total of 119 documents, then $w_{T_{ij}}(\text{"Microsoft"}) = 8/119$. The numerator is called the "term frequency" and the denominator is called the "document frequency."

Let w_V be the visitor's word vector, and again consider a word k. Instead of simply summing the number of times the visitor has encountered the word k before, we compute a weighted sum in the following manner. Any words on a page at the end of a browsing session receive a high weight; if the visitor stopped browsing after that page, then that page, more likely than not, answered the visitor's needs.[3] On earlier pages in a browsing session, any words near the *links* selected receive a moderate weight, while the other words on the page receive little or no weight. We assign moderate weight to link anchor words because these words caught the visitor's eye while browsing; these words must have some significance for the visitor. The term frequency for a word k is thus the weighted sum of all the occurrences of word k in the visitor's history. The document frequencies,

[3]Of course, a visitor may end a browsing session without ever satisfying his or her information goal, and, worse, this model would reinforce whatever dead-end the visitor stopped at. However, our experience indicates that, for a wide range of information-seeking goals, the last page in a session most often *does* satisfy the visitor's task, and does contain pertinent content. An interesting line of future work would be to improve how the content model is populated from the access logs.

by which the entries in w_V are divided, are the same as for $w_{T_{ij}}$: the number of web pages on which word k appears.

The similarity between T_{ij} and V is thus finally computed as the dot product of the word vectors normalized by their lengths:

$$sim_V(T_{ij}) = \frac{w_{T_{ij}} \cdot w_V}{||w_{T_{ij}}|| \cdot ||w_V||}$$

The $freq_V(s_{ij})$ term in Equation 4.1 measures the utility of s_{ij} by the number of visits to its parent *page* p_i. Intuitively, each visit to p_i expresses an incremental unit of value given to the *page*, and we distribute this value uniformly among the screens. Thus, $freq_V(s_{ij})$ is calculated by counting the number of visits to p_i, and dividing by the number of screens into which it decomposes. We balanced the weights ω_{sim} and ω_{freq} in Equation 4.1 to trade off each additional page view by the visitor for roughly 1% of text similarity. As with the action cost values, this choice of value seems to work well in practice, but this value can be determined empirically.

4.3 Heuristic optimizations

In the last chapter, we noted that calculating expected utility using the given equations verbatim would be computationally intractable—the equations imply a need to recur through the entire web site, evaluating each screen of content. Fortunately, we can make this evaluation more tractable with two heuristics. The first heuristic is to assume that the cost of scrolling a page of text is much smaller than that of following a link ($\gamma_s \ll \gamma_l$). In this case, the cost of viewing, say, the second screen on a distant page is dominated by the cost of reaching that page ($\gamma_l + \gamma_s \approx \gamma_l$). Thus, we may treat all pages but $\hat{p}$ as single-screen pages and can ignore the recursion due to scrolling on these pages.

The second heuristic places a bound on the number of pages considered when evaluating $\hat{p}$. We implemented a priority queue into which pages p_i and screens s_{ij} are placed, ordered by the maximum probability that the visitor will reach the respective content. For example, the first screen on $\hat{p}$ has probability 1.0, the second screen probability 0.85, and a link very frequently followed from the second screen (say, in nine sessions out of ten) has probability $0.85 \times 0.9 = 0.765$. We can tune exactly how much computation is spent evaluating $\hat{p}$ by setting a threshold on the probability—any content that has a lower probability of being viewed will simply be ignored in our computation. This threshold gives us a direct "knob" that we can turn to trade off performance for accuracy: the lower the threshold, the more accurate the evaluation, at the expense of recurring through more of the site.

4.4 Performance

PROTEUS is written in Python and uses a MySQL database server. PROTEUS runs as a proxy through which mobile clients connect. To adapt a remote web site, PROTEUS retrieves the requested page and manipulates it locally, retrieving additional pages from the remote site on demand. Our system can generate each alternative personalization in 10 milliseconds and produces an average of 40 adaptations per search iteration. Evaluating each personalization requires roughly 500 milliseconds using a probability threshold of 0.001. In our experiments we ran PROTEUS for 20 iterations of search for each page, which typically produced a personalized page in four to seven minutes.

With these rates, PROTEUS can successfully adapt a web site offline in a reasonable amount of time, but is not yet fast enough to produce personalized content at "click-time." However, two simple enhancements can substantially increase the speed of our system. First, the single most-expensive operation is retrieving content from the remote host. Our system caches as much information locally as it can, but ensuring that the local information is accurate, and collecting new content from remote hosts, consumes a great deal of time. If PROTEUS were installed on the remote web site directly, this overhead would be eliminated. Second, although Python is an excellent prototyping language, its performance in certain areas, such as string manipulation, is less than ideal. Our later systems, such as MINPATH (Chapter 5), are written in C++ and perform more than two orders of magnitude faster. We are confident that an equivalent speed-up is possible for PROTEUS.

4.5 Evaluation

In this section we present the results of an experiment that provides insight into the effectiveness of our system. In the experiment, we track ten test subjects' browsing habits on their desktop workstations and then compare how effectively these subjects can use a suite of personalized and

non-personalized web sites on a wireless Palm Connected Organizer. We measure visitor effort in terms of both time to attain the goal and the amount of navigation (number of scrolling actions and links followed) the visitor must take. In the following subsections, we present our method of collecting data, details of our experimental setup, and our results and analysis.

4.5.1 Test subjects

Nine of our ten user study participants were graduate students at the University of Washington; the other was a faculty. All participants had extensive experience on the Web (all were computer scientists), but only three had used handheld computers extensively before. None of the participants had browsed the web from a mobile device, and few users were familiar with the web sites we examined in our study.

4.5.2 Data collection

A key element in our experiment is a body of detailed access logs for the test subjects' browsing behavior. To produce these logs, we instrumented every subject's desktop browser to make all its requests through a proxy, and to not cache any web content. Thus, pages the subjects viewed and links followed were recorded in the logs[4]. We then asked the test subjects to perform a suite of information-seeking tasks ("information sorties") that we provided each day. The tasks dictate a starting page and we directed the subjects to attain their goals by browsing exclusively at the given site. The complete list of questions used in our experiment appears in Appendix A. We use two example questions for illustration here:

- "Find the current stock price for MSFT, starting at `finance.yahoo.com`". We varied the particular stock ticker symbol among a number of computer-related technology stocks (MSFT, YHOO, AMZN, *etc.*).
- "Find the make and model of the editor's choice digital camera at `cnet.com`". The variable is the consumer electronics device, which we selected from among several choices.

[4]The only page views missing from the logs were pages visited using the browser's Forward and Back buttons. However, because every new page request included the URL of the referring document, we could reconstruct the entire sequence of page requests, with the only exception of loops of Forwards and Backs.

The tasks in the seed suite were drawn randomly from a distribution of parametric questions (*i.e.*, the tasks contain variables that permit many similar but not identical questions) and represent a coherent model of visitor interest (*i.e.*, the goals were typical of a visitor whose interests did not change substantially over time).

4.5.3 Evaluation with a mobile device

In our experiment, we asked the test subjects to browse the web with a wireless Palm Connected Organizer[5]. The subjects were given a suite of information-seeking goals to achieve, drawn from the same distribution as the goals during the original seeded browsing phase. Note that about half of the goals in this test phase were identical to the goals from the seeded-browsing phase (although not all of our participants answered all the goals during the training phase). This duplication is acceptable in our experiments, because visitors frequently *will* have the same goals on a number of occasions, for example, when the answer to a question changes with time (*e.g.*, "What is the stock price of MSFT *today*?"). During this experiment, we measured the number of navigation actions taken and amount of time required to meet each information-seeking goal.

We asked test subjects to answer the suite of questions twice: once, on the unmodified web site, and once on a *personalized* version of the target site. We reversed the order of unmodified and personalized site on half the test subjects, to reduce any participant training effects. We personalized the target site for each visitor by allowing PROTEUS to first build a model of that visitor, based on the subject's past seeded browsing data (their desktop, broadband browsing data), and then to create adapted pages for the test suite. We did not personalize every page of every web site because of the sheer volume of the task—all the sites in our study contained dynamic web content and potentially an unlimited number of pages. Because our current implementation is not yet fast enough to adapt a single page in real-time, we personalized the sites before the subjects performed their tests. We chose the specific pages for PROTEUS to adapt by using our human judgment of where the subjects would likely visit during the test. Note that we did not influenced the *personalization* at all—we simply selected the subset of pages that PROTEUS will personalize, purely for the sake of

[5]We connected a Palm VII to a Ricochet wireless modem for network connectivity and used EudoraWeb [89] as the web browser on the Palm.

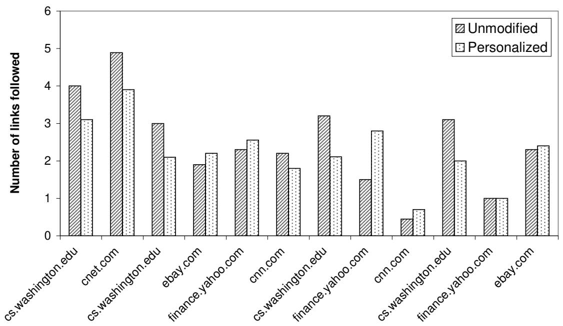

Figure 4.6: **Links followed.** Pairs of bars represent questions in the test suite and are ordered as they were presented in the experiment. Each pair is labeled with the web site used in the information sortie, and the heights show average number of links followed.

efficiency. On average, PROTEUS personalized 21 pages for each subject, which represented 38% of the subjects' page views during the experiment. Because PROTEUS personalized only a subset of the actual pages viewed, our results present a lower bound on the benefit the visitor would receive and tend to understate the full impact personalization would have for a mobile visitor.

A priori, we anticipated two results from this experiment. First, we anticipated that, to reach their information goals, the subjects would require fewer navigation actions and less time on the personalized site than on the equivalent unmodified site. Second, we anticipated that subjects' behavior would become more efficient as they repeatedly navigated the web sites on the Palm device. To mitigate this "subject-training" effect in our results, we alternated for each visitor which version of the web sites we presented first—personalized or unmodified. Thus, both versions of the web sites received equal advantage in our experiment as a whole.

Figures 4.6, 4.7, and 4.8 compare links followed, scrolling actions required, and time spent to attain each goal on the personalized versus unmodified web sites. Along the y-axis is the amount of effort—time in seconds or number of links or scrolling actions—while along the x-axis is the location of each goal listed chronologically. Results for the unmodified sites appear as the left, darker column while results for the personalized sites are given in the right, lighter column. These

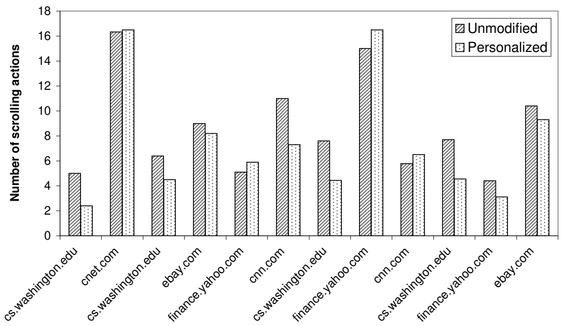

Figure 4.7: **Scrolling actions required.** Pairs of bars represent questions in the test suite and are ordered as they were presented in the experiment. Each pair is labeled with the web site used in the sortie, and the heights show average number of scrolling actions needed to complete the task.

graphs show that, for many of the sites, PROTEUS's personalizations appear quite useful: PROTEUS's addition of shortcut links and elision of unnecessary content reduced both the time required and the amount of visitor navigation at the sites. However, the study also illustrates a number of weaknesses in our implementation. These are not fundamental flaws in our approach, but rather implementation issues that must be addressed by a successful personalizer:

Overly aggressive content elision. For a number of personalized pages, particularly those for the `cnet.com` and `finance.yahoo.com` domains, PROTEUS elided blocks of content that contained the links for which the visitor was looking, thereby requiring more effort to attain the information goal. PROTEUS's visitor model incorporates both content and structural preferences, but it is clear that the weights of these preferences must be tuned carefully. As an extension to PROTEUS, one could incorporate a *confidence* model that predicts how accurate each component of the model will be on each page. For example, on a page containing predominantly links, the confidence in the structural model component is much greater than for the content (*i.e.*, word-based) component. That is, the personalizations on a page of links should depend more strongly on the probability the visitor will follow each link, and less strongly on the textual content of each anchor word. Additionally, the confidence in the structural component depends on the number of times the

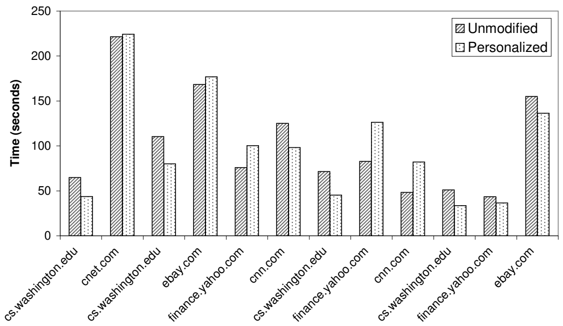

Figure 4.8: **Time required.** Pairs of bars represent questions in the test suite and are ordered as they were presented in the experiment. Each pair is labeled with the web site used in the sortie, and the heights show average time in seconds to complete the task.

visitor has viewed the page. If the visitor views the page frequently, and follows the same two links each time, then the personalizer has much higher confidence that those links are very important elements of the page.

Another side effect of overly aggressive elision was that visual cues used by visitors to orient themselves were removed by PROTEUS. Without these cues, visitors had difficulty navigating within each page. The cues themselves are apparently "not interesting" as far as the user model would predict, but clearly do have some use for the visitors.

Inconspicuous elision links. We designed our method of eliding content explicitly to tolerate incorrect elision of content by PROTEUS: PROTEUS creates a link in the place of the removed content. However, the subjects in our study often could not find the link to the elided content, usually for one of two reasons. First, the link anchor text for elided content was taken from the first few words of the removed content. For the examples in Figure 4.3 the anchors are intuitive (*e.g.*, "Mutual Funds..."). However, when the elided content block contained, for instance, an advertisement at the top, PROTEUS selected the text from the advertisement as the link anchor—clearly, an unintuitive choice. Unfortunately, automatically finding a brief summary of a block of text, much less a page of content, is an open research problem [68]. A second reason elision links were difficult to find was that their visual appearance was no different from the other links on the page. Thus, the visitors could not tell when a link led to elided content, or when the link simply led to another page. A simple solution to this problem is to add a unique annotation to elision links, or find some other approach to make elision links more salient.

Over-estimated navigation probabilities. On graphically intense pages, such as `cnn.com` or `cnet.com`, the visitor can easily find his or her desired link when using the desktop browser. However, on the mobile device, such a page typically appears as a morass of links and text, often indistinguishable from one another, and the visitor has great difficulty in locating the link of interest. Unfortunately, PROTEUS calculates the probability that the visitor follows a link simply as how often the visitor has followed the link before while browsing on the *desktop*, when the visitor is unencumbered by the display constraints of the mobile device. Thus, PROTEUS tends to overestimate the link probabilities and, instead of adapting the page to reduce the visual complexity, will search for (what it views as) more useful personalizations, such as adding new shortcut links. In another line of future work, one could expand the link probability estimate to take link *salience* into account, which will discount the value of visually complex pages and encourage PROTEUS to create simpler pages. An alternative approach would be to explicitly model the differences in navigation behavior between desktop and mobile users, and apply this function to the observed desktop model to create an approximate mobile model.

4.6 Summary

This chapter described our initial attempts to personalize web sites, in this case for mobile visitors. Our experiment provides evidence that search-based personalization is an effective means of retargeting existing content to mobile clients. However, our implemented system displayed some weaknesses, in particular slow run-time performance and possible detrimental changes to the site. In the next chapters, we address both these concerns, by concentrating on the add-shortcut adaptation. Adding shortcut links rarely degrades the web experience and frequently improves it. In the next chapter we describe an algorithm that identifies useful shortcut destinations very efficiently, and subsequently we refine this approach to take advantage of the relational structure inherent in most web sites.

Chapter 5

THE MINPATH ALGORITHM

In this chapter, we concentrate on a single personalization, add-shortcut, that had a high impact in our PROTEUS experiment. We develop an algorithm that finds shortcuts efficiently, and in particular leverages much more of the web server log data to build more reliable navigation models. In the next section we present our MINPATH algorithm, followed by an evaluation of several instantiations of MINPATH on data from our institution's web site (`www.cs.washington.edu`).[1]

5.1 Finding shortcuts with MINPATH

We assume the visitor is on an information sortie—looking for the page at the site that will answer an information need. Our goal, then, is to help the user reach that target page quickly; we do so by providing shortcut links to visitors to help shorten long trails. Our system adds shortcut links to every page the visitor requests. Ideally, the shortcuts suggested will help the visitor reach the destination of the trail with as few links as possible. We state the shortcut link selection problem precisely as:

- **Given:** a visitor V, a trail prefix $\langle p_0, \ldots, p_i \rangle$, and a maximum number of shortcuts m;

- **Output:** a list of shortcuts links ($p_i \rightarrow q_1$, a_1, c_1), ..., ($p_i \rightarrow q_m$, a_m, c_m), where a_j is the anchor of link $p_i \rightarrow q_j$, the link is placed as content node c_j, and the list *minimizes* the number of links the visitor will follow between p_i and the visitor's destination.

Each link selected includes three parts: the link destination, the anchor shown to the user, and the location on the source page to place the link. In this thesis, we concentrate on only the first of these challenges, and we evaluate our approach using a simulated user that ignores the anchor text and

[1] Much of this chapter appeared in the 17th International Joint Conference on Artificial Intelligence [3].

location. However, a reasonable strawman approach could use the destination page's <title>, if it exists, or URL as the anchor, and place the link in a fixed-position region on each source page.

The choice of m, the number of shortcuts to add per page, is an important one, and is given as an input to the problem. Large values of m, for instance, equal to the number of pages on the site, would allow a personalizer to add a shortcut to every page. Then the user could "easily" navigate to any page on the site with a single link. Of course, such a personalized page would be unusable: the page would be unmanageably large, from the standpoint of rendering the HTML, and particularly from the cognitive standpoint of the visitor understanding the content. Instead, we anticipate that the human web site designer will choose m to be small, for example, five or fewer. With a small value of m, and by placing the shortcuts in a consistent position on each page in the site, the designer can minimize the added cognitive cost of displaying dynamic content on each page. Thus, in our work, we assume the site designer has made such choices, and we do not explicitly model the cognitive cost in adding shortcuts.

The last page in the trail prefix, p_i, is the page the visitor has requested most recently, and the page on which the shortcuts are placed. We calculate the *savings* that a single shortcut $p_i \rightarrow q$ offers as the number of links the visitor can avoid by following that shortcut. If we know the entire trail $T = \langle p_0, \ldots, p_i, \ldots, p_n \rangle$, then the number of links saved by $p_i \rightarrow q$ is:

$$\begin{cases} j - i - 1 & \text{if } q = p_j \text{ for some } i < j \leq n \\ 0 & \text{otherwise} \end{cases}$$

That is, if the shortcut leads to a page further along the trail, then the savings is the number of links skipped (we subtract one because the visitor must still follow a link—the shortcut link). If the shortcut leads elsewhere, then it offers no savings.

5.1.1 *The* MINPATH *algorithm*

At a high level, our approach follows the search framework of Chapter 3, but optimizes for adding shortcuts. First, we make the assumption that the intrinsic utility of each page is equal (thus, the expected utility of each page depends solely on the extrinsic utility—the page's connections with the rest of the web site). Second, instead of searching through web-state-space, our algorithm computes the value of each relevant shortcut link in only one invocation. This choice has an effect on the optimality of MINPATH, as we discuss in Section 5.1.2. MINPATH employs the trail model of user

navigation (Section 3.2.2).

If one had knowledge of the complete trail $\langle p_0, \ldots, p_i, \ldots p_n \rangle$, selecting the best shortcut destination at any page p_i would be easy: simply make it $p_i \rightarrow p_n$[2]. Of course, at runtime, a visitor has viewed only a trail prefix, and the adaptive web site must infer the remaining pages. Our approach relies on a model of the visitor's behavior to compute a probability for every possible trail suffix $\langle q_{i+1}, \ldots, q_n \rangle$ on the site. Intuitively, these suffixes are all the possible subtrails originating from p_i. Given a suffix and its probability, we assign an *expected savings* to the shortcut $p_i \rightarrow q_j$ to each q_j in the suffix as the product of the probability of the suffix and the number of links saved by the shortcut. Note that a particular shortcut $p_i \rightarrow q_j$ may appear in many trail suffixes (*i.e.*, many trail suffixes may pass through the same page q_j), and so the expected savings of a shortcut is the sum of the savings of the shortcut for all suffixes.

A brief example will elucidate these ideas. Suppose that a visitor has requested the trail prefix $\langle A, B, C \rangle$ and we wish to find shortcuts to add to page C. Suppose that our model of the visitor indicates there are exactly two sequences of pages the visitor may complete the trail with: $\langle D, E, F, G, H \rangle$, with a probability of 0.6, and $\langle I, J, H, K \rangle$ with a probability of 0.4. The expected savings from the shortcut $C \rightarrow E$ would be $0.6 \times 1 = 0.6$, because the trail with page E occurs with probability 0.6 and the shortcut saves only one link. The expected savings for shortcut $C \rightarrow H$ includes a contribution from both suffixes: $0.6 \times 4 + 0.4 \times 2 = 2.4 + 0.8 = 3.2$. Of course, in practice, there are far many more than two trail suffixes to consider, and these suffixes may even be subsequences of each other (*e.g.*, $\langle I, J \rangle$, $\langle I, J, H \rangle$, $\langle I, J, H, K \rangle$, *etc.*).

The MINPATH algorithm is shown in Table 5.1. The ExpectedSavings function constructs the trail suffixes by traversing the directed graph induced by the web site's link structure. Starting at the page last requested by the visitor, p_i, ExpectedSavings computes the probability of following each link and recursively traverses the graph until the probability of viewing a page falls below a threshold, or a depth bound is exceeded. The savings at each page (*CurrentSavings*) is the product of the probability, P_s, of reaching that page along suffix T_s and the number of links saved, $l - 1$. The MinPath function collates the results and returns the best m shortcuts. The next section describes

[2]We leave to future work the task of generating an anchor for the link and selecting a position for it on the originating page.

how we obtain the model required by MINPATH.

5.1.2 *Analysis of* MINPATH

Returning the best m shortcuts is optimal only in a restricted sense. Specifically, MINPATH assumes that the expected savings of shortcut $p \rightarrow q$ is independent of other shortcuts that may be added to page p. Based on this assumption, MINPATH makes a greedy choice when building its set of m links to add to p: MINPATH takes the m links that, individually, have high expected savings. This assumption is likely violated in practice; for example, two shortcuts with high expected savings may both lead along a common trail suffix, and so the expected savings of both is not the sum of each individually. However, it is convenient for computation, as MINPATH needs to calculate the expected savings for each page in the site only once. The complexity of this computation (ExpectedSavings in Table 5.1) is $O(b^d)$ where b is the average number of links on each page (*i.e.*, the branching factor) and d is the length of trail suffixes considered. We choose small values for d to ensure this computation is fast.

It is desirable to relax the assumption that shortcuts are independent, because doing so would allow MINPATH to make better use of the limited number of shortcuts added per page. One proposal is to follow a greedy heuristic, selecting, at first, the single best shortcut $p \rightarrow q$, but then re-calculating the expected savings for all other shortcuts *given* the existence of $p \rightarrow q$. Unfortunately, this proposal is not provably optimal—the optimal set of *two* shortcuts may be the second- and third-place links when ordered by expected savings. However, this approach would produce sets of shortcuts that perform no worse, and perhaps much better, than our current MINPATH algorithm. One important subtlety is how the existence of previously selected shortcuts affects navigation throughout the site. For example, suppose that page x has two links, $x \rightarrow y$ and $x \rightarrow z$, and the shortcut $p \rightarrow q$ bypasses the trail $p \rightarrow x \rightarrow y \rightarrow q$. Given the existence of $p \rightarrow q$, the probability of navigation at x may be effected, because the shortcut reduces the fraction of traffic at x that follows $x \rightarrow y$. In addition, if q links to z, then the shortcut $p \rightarrow z$ would have lower expected savings, because the expected trail to z has become shorter (one link, now, from p to q, instead of three). We leave as future work exploring this extension, although it should be largely a matter of careful bookkeeping (*i.e.*, recording the probability and length of each trail suffix, and adjusting the probabilities and savings

Table 5.1: **MINPATH algorithm.**

Inputs:

T	*Observed trail prefix* $\langle p_0, \ldots, p_i \rangle$
p_i	*Most recent page requested*
V	*Visitor identity*
m	*Number of shortcuts to return*

MinPath(T, p_i, V, m)
  $S \leftarrow$ ExpectedSavings$(p_i, T, V, \langle\rangle, 1.0, 0, \{\})$
  Sort S by expected page savings
  Return the best m shortcuts in S

Inputs:

p	*Current page in recursive traversal*
T	*Trail prefix (observed page requests)*
V	*Visitor identity*
T_s	*Trail suffix (hypothesized pages in traversal)*
P_s	*Probability of suffix* T_s
l	*Length of suffix* T_s
S	*Set of shortcut destinations and their savings*

ExpectedSavings$(p, T, V, T_s, P_s, l, S)$
  If ($l \geq$ depth bound) or ($P_s \leq$ probability threshold)
    Return S
  If ($l \leq 1$)
    CurrentSavings $\leftarrow 0$
  Else
    CurrentSavings $\leftarrow P_s \times (l-1)$
  If $p \notin S$
    Add p to S with Savings(p) = *CurrentSavings*
  Else
    Savings$(p) \leftarrow$ Savings(p) + *CurrentSavings*
  Trail $\leftarrow$ concatenate T and T_s
  For each link $p \rightarrow q$
    $P_q \leftarrow$ probability of following $p \rightarrow q$ given *Trail* and V
    $T_q \leftarrow$ concatenate T_s and $\langle q \rangle$
    $S \leftarrow$ ExpectedSavings$(q, T, V, T_q, P_q, l+1, S)$
  Return S

given the proposed shortcuts).

A provably optimal solution may require examining all possible sets of m shortcuts on the site[3]. There are $\binom{n}{m}$ such sets on a site with n pages, and for each set, MINPATH must compute the expected number of links the visitor will follow given the set of shortcuts. The number of these sets grows roughly n^m, and the computation for each requires $O(b^d)$ time. Thus, the optimal solution may require computation time greatly in excess of MINPATH or our other proposed, but non-optimal, approach.

5.2 Predictive models

The key element to MINPATH's success is the predictive model of web usage; in this section, we describe the models we have implemented and evaluated. The probabilistic model MINPATH uses must predict the next web page request p_i given a trail prefix $\langle p_0, \ldots, p_{i-1}\rangle$ and the visitor's identity V (the identity can lead to information about past behavior at the site, demographics, *etc.*): $P(p_i = q|\langle p_0, \ldots, p_{i-1}\rangle, V)$. Of course, a model may condition this probability on only part or even none of the available data; we explore these and other variations in this section. To simplify our discussion, we define a "sink" page that visitors (implicitly) request when they end their browsing trails. Thus, the probability $P(p_i = p_{sink}|\langle p_0, \ldots, p_{i-1}\rangle, V)$ is the probability that the visitor will request no further pages in this trail. Finally, note that the models are learned offline, prior to their use by MINPATH. To answer our second thesis question in the affirmative, that is, to enable dynamic personalizations per page impression, only the evaluation of the model must run in real time.

5.2.1 Unconditional model

The simplest model of web usage predicts the next page request p_i without conditioning on any information. We learn this model by measuring the proportion of requests for each page q on the site during the training period[4]:

[3] In the worst case, all possible sets of m links must be examined. However, more typically, many sets may be eliminated because the sum of the expected savings of their shortcuts, independently, is less than that of some current-best set of shortcuts.

[4] More precisely, throughout our implementation we use MAP estimates with Dirichlet priors [54], setting m to 4.0.

$$P(p_i = q) = \frac{\text{number of times } q \text{ requested}}{\text{total number of page requests}}$$

We assume the visitor can view a page only if it is linked from the current page. Thus MINPATH forces the probabilities of pages not linked from the current page to be zero and renormalizes the probabilities of the available links. If the current page is p_{i-1}, then MINPATH calculates:

$$P(p_i = q|p_{i-1}) = \begin{cases} \frac{P(p_i=q)}{\sum_{q'} P(p_i=q')} & \text{if } p_{i-1} \rightarrow q \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

where the q' are all the pages to which p_{i-1} links.

Despite having large volumes of training data, we cannot build a model that predicts each and every page—many pages are requested too infrequently to reliably estimate their probability (although we address this concern directly in Chapter 6). For example, during September 2000, the UW CSE web site (`www.cs.washington.edu`) received 129,000 page requests covering 8,000 unique URLs, but no requests for the remaining 232,000 web pages. More generally, Glassman has observed that web page requests follow a Zipf-like distribution [51], in which a very small fraction of the pages receive a very large fraction of the traffic. So, instead of modeling navigation between individual pages, we group pages together to increase their aggregate usage counts, and replace page requests by their corresponding group label (much in the spirit of work by Zukerman *et al.* [98]). Specifically, we use the hierarchy that the URL directory structure imposes as a hierarchical clustering of the pages, and select only the most specific nodes (the ones closest to the leaves) that account for some minimum amount of traffic, or usage, on the site. (This selection corresponds to a partition in the set Π used to model the web site.) The pages below each node share a common URL prefix, or *stem*, that we use as the label of the node. By varying the minimum usage threshold, we select more or fewer nodes; in Section 5.3, we report how MINPATH's performance is correspondingly affected.

Figure 5.1 illustrates this idea on a portion of the web site we used to evaluate MINPATH. In the example, we suppose that 1,000 web requests are made in the `/education` hierarchy of the site—pages whose URLs begin with `/education`. If the threshold were set at 15% (150 requests), then the selected nodes would be cse142, cse143, courses, course-webs.html, and education;

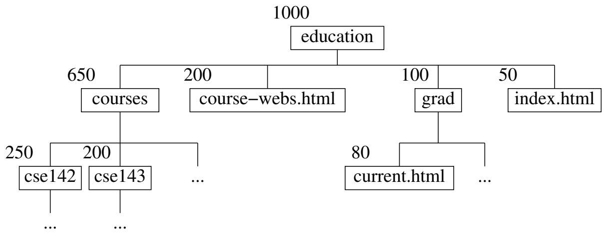

Figure 5.1: **Aggregating web usage at URL stems.** The URLs are grouped according to their directory structure. Associated with each node is a count of how often any of the URLs below that node were requested.

each of these nodes received at least 150 requests[5]. Note that this process is not simply a cut through the tree; both leaf nodes and interior nodes are selected. Note, also, that some of the selected nodes are actual web pages while others are directories that contained many pages and subdirectories. The stem at each node is the prefix common to all URLs below the node, and is formed by concatenating the labels from the root to that node. For example, the stem for the cse142 node is `/education/courses/cse142/`.

5.2.2 Naïve Bayes mixture model

The unconditional model assumes all trails on the site are similar—that a single model is sufficient to accurately capture their behavior. Common intuition suggests this assumption is false—different visitors produce different trails, and even the same visitor may follow different trails during separate visits. As an alternative, we hypothesize that each trail belongs to one (or a distribution) of K different *clusters*, each described by a separate model. We can thus compute the probability of requesting page q by conditioning on the cluster identity C_k:

[5]Note, however, that usage at higher nodes *excludes* lower nodes that are selected: courses is selected because its usage, exclusive of cse142 and cse143, is still greater than 150.

$$P(p_i = q|\langle p_0, \ldots, p_{i-1}\rangle) = \sum_{k=1}^{K} P(p_i = q|C_k)P(C_k|\langle p_0, \ldots, p_{i-1}\rangle) \qquad (5.1)$$

The result is a *mixture model* that combines the probability estimates $P(p_i = q|C_k)$ of the K different models according to a distribution over the models. By Bayes' theorem, $P(C_k|\langle p_0, \ldots, p_{i-1}\rangle) \propto P(C_k)P(\langle p_0, \ldots, p_{i-1}\rangle|C_k)$. To calculate $P(\langle p_0, \ldots, p_{i-1}\rangle|C_k)$, we make the Naïve Bayes assumption that page requests in the trail are independent given the cluster, thus: $P(\langle p_0, \ldots, p_{i-1}\rangle|C_k) = \prod_{j=0\ldots i-1} P(p_j|C_k)$. The resulting model is a *Naïve Bayes mixture model* (similar to those used in AUTOCLASS [26]) for which we learn the model parameters $P(p_i = q|C_k)$ and the cluster assignment probabilities $P(C_k)$ using the EM algorithm [34].

The mixture model uses the probabilities $P(C_k|\langle p_0, \ldots, p_{i-1}\rangle)$ as a "soft" assignment of the trail to the cluster—each cluster C_k contributes fractionally to the sum in Equation 5.1. Alternatively, we may use a "hard" assignment of the trail to the most probable cluster, C_*. We explore both of these possibilities in Section 5.3. The value of K may be fixed in advance, or found using holdout data. For each value of K, we compute the likelihood of the holdout data given the previously learned model, and choose the K that maximizes the holdout likelihood.

An additional piece of information useful when selecting the cluster assignment is the visitor's identity, which we can incorporate by conditioning Equation 5.1 on V. If we assume that page requests are independent of the visitor given the cluster, then the only change to the right side of Equation 5.1 is that $P(C_k)$ becomes $P(C_k|V)$. Unlike an individual trail, a visitor's behavior may not be well represented by any single model in the mixture because the same visitor will behave differently when seeking different information goals[6]. Thus, we represent a visitor as a mixture of models, and estimate the $P(C_k|V)$ as the proportion of the visitor's history that is predicted by C_k. Specifically, let $H = \{T_1, \ldots, T_h\}$ be the set of h trails the visitor has produced on the site previous to the current visit, and $P(T_i|C_k)$ the probability that cluster C_k produced trail T_i; then

$$P(C_k|V) = \sum_{i=1}^{h} P(T_i|C_k)/h$$

[6]This assumption depends on how the cluster models are learned. We could either cluster individual trails, or cluster *visitors* but train the models on the visitors' *sets* of past browsing trails. We have experimented with both approaches and found the former yielded the best results.

5.2.3 Markov models

Both the unconditional and Naïve Bayes mixture models ignore a key piece of information from the web accesses: the sequential nature of the page trails. A *first-order Markov* model, on the other hand, incorporates this information by conditioning the probability of the next page on the current page: $P(p_i = q|p_{i-1})$. The Markov model is trained by counting the transitions from pages p_{i-1} to p_i in the training data, and by counting how often each page appears as the initial request in a trail. As before, we replaced the URLs of page requests with URL stems to increase the volume of relevant training data. The need for this transformation is even greater for the Markov model because it has quadratically more probability values to estimate than the unconditional model, and the events (the links $p_{i-1} \rightarrow p_i$) are more rare.

We experimented with first- and second-order Markov models ($P(p_i = q|p_{i-1})$ and $P(p_i = q|p_{i-1}, p_{i-2})$), and with using a mixture of Markov models [24]. We use the same EM-based method to build these mixtures as we did to learn the Naïve Bayes mixture model.

5.2.4 Positional and Markov/Positional models

In addition to conditioning the probability on the last requested page, we also consider conditioning on the ordinal position of the request in the visitor's trail: $P(p_i = q|i)$ or $P(p_i = q|i, p_{i-1})$. Effectively, this model is equivalent to training a separate model (either unconditional or Markov) for each position in the trail (although, for practical purposes, we treat all positions after some limit L as the same position). Visual inspection of the training trails led us to hypothesize that these models may better predict behavior, although conditioning on the additional information increases the amount of training data necessary to properly fit the model.

5.3 Results

We evaluate MINPATH's performance on usage at our home institution's web site based on data from September 2000 and February 2002. The September 2000 data produced a training set of 35,212 trails (approximately 20 days of web usage) and a test set of 2,500 trails (approximately 1.5 days of usage); the time period from which the test trails were drawn occurred strictly after the training period. During the training and testing periods, 11,981 unique pages were requested from the total

population of 243,119 unique URLs at the site. The February 2002 data produced 154,724 training trails in the first three weeks of the month and 2,500 test trails chosen uniformly from the trails in the last week. We consider only those trails with link length at least two, because shorter trails cannot be improved. We set MINPATH's link depth bound to 8 and probability threshold to 10^{-5}; in all our experiments the probability threshold proved to be the tighter constraint.

We measure MINPATH's performance by the number of links a visitor must follow to reach the end of the trail. We estimate visitor behavior when provided shortcuts by making two simplifying assumptions. First, we assume that, when presented with one or more shortcuts that lead to destinations along the visitor's trail, the visitor will select the shortcut that leads farthest along the trail (*i.e.*, the visitor greedily selects the apparently best shortcut). Second, when no shortcuts lead to pages in the visitor's trail, the visitor will follow the next link in the trail (*i.e.*, the visitor will not erroneously follow a shortcut). Note, finally, that MINPATH places shortcuts on each page the visitor requests, and so the visitor may follow multiple shortcut links along a single trail.

Without shortcuts, the average length of trails in the September 2000 test set is 3.42 links and 3.33 for the February 2002 test set. Given an oracle that could predict the exact destination of the visitor's current trail, MINPATH could reduce the trail to exactly one link. The difference between the unmodified length (3.42 or 3.33) and one link is the range of savings MINPATH can offer web visitors.

We first explore the relationship between the minimum URL usage threshold and the performance of MINPATH. Using the February 2002 data, we compare thresholds between 5% (which produces 10 URL stems) and 0.005% (which produces 3,216 stems) and all pages visited more than once (minimum, yielding 21,140 stems) when learning a mixture of first-order Markov models. Figure 5.2 shows these results. The general trend is that finer-grained models (*i.e.*, lower usage threshold and thus more URL stems) lead to improved MINPATH performance, although the differences in performance between thresholds less than 0.05% are very small. In the September 2000 experiments we adopted a 0.025% threshold; for the February 2002 data, we chose 0.010%[7].

We next compared MINPATH's performance when using a variety of models (see Figure 5.3). The first column shows the number of links followed in the unmodified site (from the September

[7] We chose the lower threshold for the February data because we had more training data available.

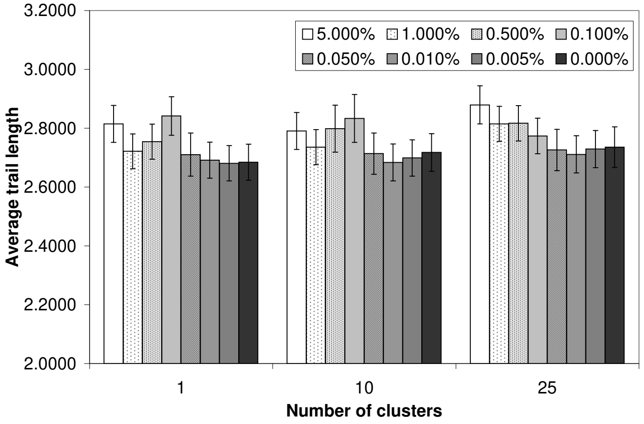

Figure 5.2: **Varying threshold of URL stem usage.** All series depict a first-order Markov model and adding three shortcuts per page. The series range from 0.0% minimum usage threshold (21,140 stems) to 5% (10 stems). The series are grouped by the number of clusters in the mixture. Error bars denote 95% confidence intervals. Results based on February 2002 data set.

2000 data). In the second and third sets of columns, MINPATH uses, respectively, an unconditional and Markov model and produces 1, 3, or 5 shortcuts. In the last two sets, MINPATH uses mixture models of either 10 or 25 clusters, and selects the distribution of the models in the mixtures based on only the current trail prefix (ignoring past visitor behavior). These graphs demonstrate first that MINPATH does reduce the number of links visitors must follow: when using a mixture of Markov models and suggesting just three shortcuts, MINPATH saves 0.97 links, which is 40% of the possible savings. Second, we see that the Markov model, by conditioning on the sequence information, outperforms the unconditional model substantially—three shortcuts suggested with the Markov model are better than five shortcuts found with the unconditional model. Third, these results indicate that mixture models provide a slight advantage over the corresponding single model (for example, 2.72 for the Naïve Bayes mixture model versus 2.75 for the unconditional model). We computed the average of the difference in trail length between the single model and the mixture

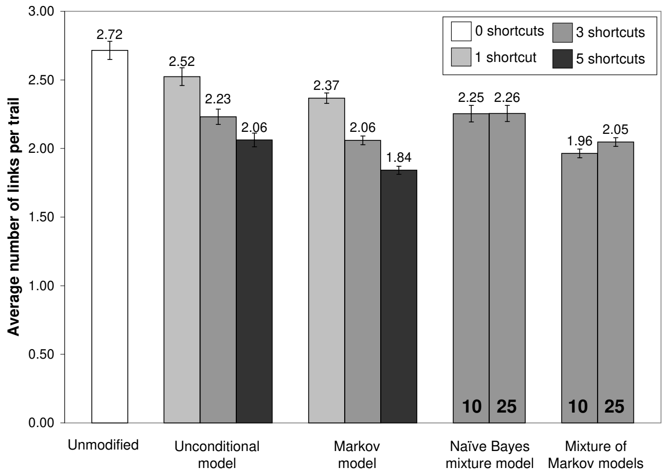

Figure 5.3: **MINPATH's performance (September 2000).** Each column shows the average number of links followed in a trail. The mixture model columns are annotated with the number of clusters. All error-bars denote 95% confidence intervals.

model for each test trail, and found the gains are significant at the 5% level (*i.e.*, the 95% confidence interval of the mean of the difference between the models lies completely above zero). Finally, we found that the differences between 10 and 25 clusters in the mixture are not statistically significant.

Figure 5.4 shows a comparison of MINPATH's performance with a range of mixture model sizes on the February 2002 data. We found no significant difference based on the number of clusters, but this result could be explained by one of two reasons. First, if a test trail is produced by a visitor who has not previously browsed the site, then MINPATH will use mixture model weights following the class priors. That is, all test trails produced by first-time visitors will be predicted using the same combination of mixture components, which is equivalent to a single model whose transition matrix and initial state vector is an interpolation of the mixture components. Thus, additional mixture components may not help, because MINPATH has no insight on how to select the right component for the first-time visitor.

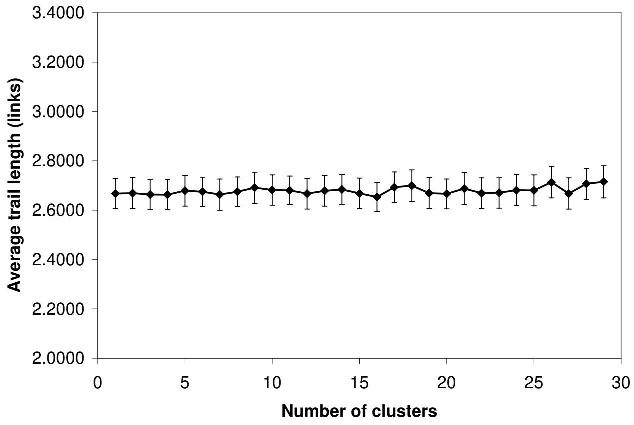

Figure 5.4: **Varying number of clusters.** MINPATH's performance suggesting three shortcuts per page and using a mixture of first-order Markov models. Error bars denote 95% confidence intervals; results taken from February 2002 data.

A second explanation is that a single first-order Markov model can encode the behavior of several potentially very different groups of users. For example, users in cluster C_1 may frequently follow the trail $\langle p_a, p_b, p_c \rangle$, and users in C_2 follow $\langle p_d, p_e, p_f \rangle$. A single first-order Markov model can predict navigation for *both* clusters with the same accuracy as two separate models, because the relevant entries in the transition matrix for the two trails share no common source or destination pages. We call such clusters of users *non-overlapping*, as their behavior does not "overlap" in the site. For our purposes, conflating non-overlapping models is not detrimental—all that MINPATH requires is reasonable predictive accuracy. However, tools that visualize communities of users based on their browsing, such as WebCANVAS [24], must separate these clusters into separate models. WebCANVAS separates non-overlapping clusters by allowing only one state in the Markov model to have non-zero initial probability. Thus, each model effectively predicts sequences that all begin with the same page request, and then perhaps diverge. We implemented the WebCANVAS splitting approach, as well as an alternative cluster-splitting approach of our design, to verify that the

predictive behavior of the conflated model was not worse than the split models. In our method of splitting conflated models, we separate non-overlapping components as a post-modeling step. We build a graph from the first-order Markov model's transition matrix, in which there is a node for each model state and an edge between states (nodes) if the transition probability between them is greater than some threshold τ. We then separate the connected components in this graph and build separate first-order Markov models for each component. This approach seemed to perform well—each model appeared to display only a single behavioral cluster—but we did not explore this approach in detail, as it does not improve predictive accuracy (and, hence, is not useful for MINPATH).

Finally, we note that the results of Figure 5.4 do not contradict our finding that a mixture of Markov models was best for the September 2000 data. The URL granularity of the February 2002 data was finer, allowing the Markov model to separate transition probabilities that would have been conflated in the September 2000 models. In the September 2000 models, because more pages are grouped together in the same URL stem, differences in behavior among those pages are lost in a single Markov model.

In Figure 5.5, we compare methods for selecting the mixture distribution for a trail prefix, using mixtures of 10 models. Each group of columns shows a different combination of model and assignment type (hard or soft). In each group, we in turn condition the assignment on no information (*i.e.*, we use a uniform distribution for the soft assignment and random selection for the hard assignment), the visitor's past trails, the visitor's current trail, and both the past and current trails. Our first conclusion is that soft assignment is a better choice for both mixture models (significant at the 5% level). Second, both past trails and the current trail prefix help MINPATH select an appropriate assignment to the cluster models. However, the combination of both features is not significantly better than using just the current trail prefix with the Naïve Bayes mixture model, and does slightly worse than just the current trail with the mixture of Markov models. This result is somewhat surprising; we had expected, especially when the prefix is short, that the past trails would provide valuable information. Apparently, however, even the first one or two page requests in a trail are sufficient to assign it to the appropriate clusters. It would be interesting to investigate if this result remains true for larger sites.

Our last variation of model conditions the probability on the ordinal position of the page request in the trail. We compared the unconditional and Markov models against positional and

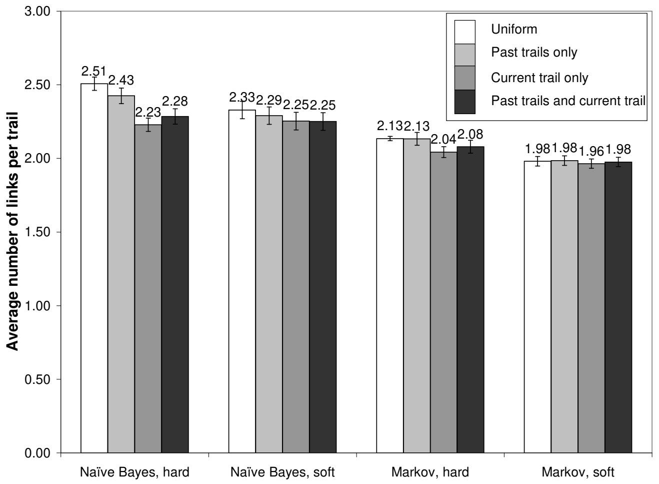

Figure 5.5: **Varying model assignment strategy.** Each of the four series represents a different model assignment strategy. Results drawn from September 2000 data.

Markov/positional models, choosing several values of the maximum number of distinct positions L. Figure 5.6 shows these results for values of L ranging from 0 (*i.e.*, a non-positional model) through 10. The extra positional information improves the performance of the unconditional model, significantly at the 5% level, but does not substantially improve the Markov model.

In his thesis, Perkowitz proposed an alternative method for finding shortcuts [79]. In his memory-based approach, for each page P viewed on the site, the personalizer records how often every other page Q not directly linked from P is viewed after P in some trail. Effectively, this approach estimates the probability that a visitor at P will eventually view Q, given that $P \rightarrow Q$ does not exist, by counting how often this event has occurred in the training data. When page P is requested in the future, the shortcuts are the top m most-requested pages Q. Note that, typically, the destinations with the highest probabilities will be *exactly* two links from P, because navigation to any further

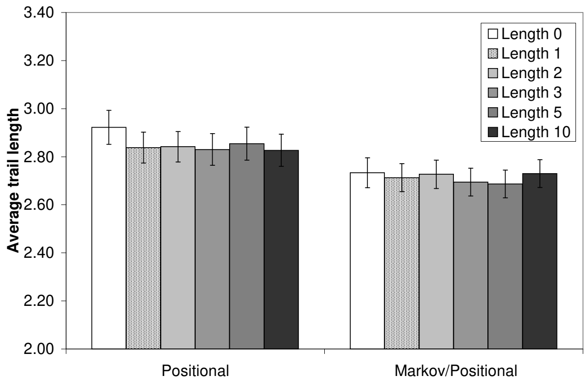

Figure 5.6: **Varying length of positional models.** Positional length zero corresponds to the non-positional model (either unconditional or first-order Markov). Results from February 2002 data.

destinations will vary rarely be more likely than navigation to the closer page[8]. As a consequence, these shortcuts usually save only one link, but typically lead down different trails in the site, because only one destination along a given trail is link length two from P.

MINPATH also estimates the probabilities of reaching any given Q from P, but does so by composing the page transition probabilities along a trail through the site. The advantage of our approach is that it reduces data sparseness, although at the expense of making a first-order assumption[9] that may not hold in practice. MINPATH can also require far less storage than the memory-based approach: the first-order Markov models need space linear in the number of visited links in the site, but the memory-based approach requires space linear in the number of visited trails—typically far larger than the number of links.

Figure 5.7 compares MINPATH using first-order Markov models with the memory-based approach for a range of shortcuts shown on each page. We found that the memory-based approach and

[8]Navigation to a distant Q may be more likely than to a closer page only when there are multiple paths through the site, through different interim pages, that all reach Q.

[9]Or, a second-order assumption, if MINPATH uses a second-order Markov model, *etc.*

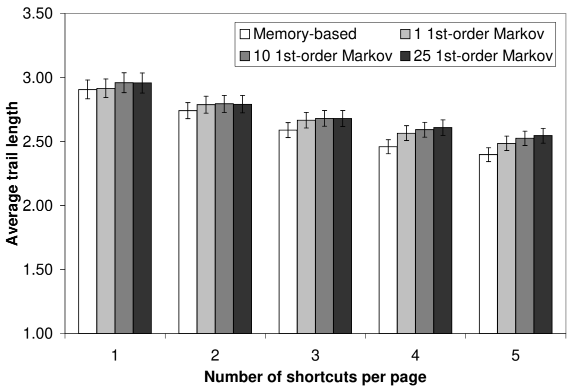

Figure 5.7: **Comparing MinPath and memory-based approach.** The Markov models group URLs together at the 0.010% threshold. The difference between the memory-based approach and the MinPath runs are significant for three, four, and five shortcuts. Results from February 2002 data.

MinPath performed approximately equivalently when suggesting only one or two shortcuts per page, but, as we expected, the memory-based approach takes better advantage of the larger number of shortcuts to suggest per page. Note, also, that the unmodified trail length is already rather small (3.33 in the February 2002 data) so the memory-based approach is not particularly disadvantaged by suggesting shortcuts saving only one link. When we restricted our test to trails that are much longer (for instance, a minimum of 10 links), MinPath seemed to perform much better than the memory-based approach (although, we have insufficient trails to measure this difference with statistical significance).

An advantage that MinPath has over Perkowitz's approach is that MinPath admits a more versatile selection of shortcuts. For example, MinPath can calculate the expected savings of each shortcut given the existence of the other shortcuts added to the requested page. Perkowitz's approach cannot take advantage of this conditional information, because it derives its recommendations directly from the original usage data. In addition, although the memory-based approach is limited

to predicting only previously-observed destinations, we will present a model of navigation in the next chapter that can predict behavior at previously unseen pages. MinPath takes advantage of this model to offer shortcuts to destinations that are novel in the testing data.

We finally note that MinPath's running time is quite small. The models MinPath uses are learned offline, but the process usually requires only several minutes. Given a model and the trail prefix, MinPath finds a set of shortcuts in 0.65 seconds on an average desktop PC. This delay is more than fast enough for delivering content to wireless web clients, and we are confident that with judicious code optimization it could be reduced to below one-tenth of a second. Additionally, by changing the depth bound and probability threshold that limit the recursion in the MinPath algorithm, we can directly trade off computation time for shortcut accuracy.

5.3.1 Discussion

The models MinPath uses have a few weaknesses, however. First, despite having perhaps millions of training examples, these models cannot learn reliable transition probabilities at the page-level—the number of probabilities to estimate is more than the available training data can allow. Moreover, web requests follow a Zipf-like distribution [51], so a very small set of pages receives an overwhelming majority of the traffic and most pages are visited infrequently. MinPath lessens this difficulty by grouping web pages by their common URL stems, but this heuristic is the second weakness: it fails to capture the semantic relations among the web pages. If two pages exist in the same directory, they are related (for example, an exam review page and a course assignment page), but perhaps not as closely related as some pages from different directories (for example, exam review pages from *all* previous offerings of that course). In the next chapter, we describe an approach that addresses both these weaknesses, by incorporating the relational structure of a web site into a Markov model of navigation.

5.4 Summary

This chapter presented our MinPath algorithm for efficiently finding high-quality shortcut links. Our experiments show that MinPath can be highly effective at improving navigation in sites, which is beneficial for wireless and desktop web visitors alike. We explored several predictive models of web usage, evaluated how they perform with MINPATH, and found that a mixture of Markov models worked best.

Chapter 6

RELATIONAL MARKOV MODELS

At the end of the last chapter we highlighted two main weaknesses of the probabilistic models used with MINPATH: the models have insufficient data for reliably learning page-grained accesses, and the models ignore the relational structure of the site, which may be of value. These criticisms apply broadly to many probabilistic models and applications, not just Markov models predicting web navigation. In this chapter, we present a novel technique for endowing Markov models with relational structure, thus producing relational Markov models (RMMs). We evaluate RMMs and traditional Markov models for predicting navigation in several web sites, and compare their effectiveness in finding shortcut links with MINPATH[1].

In the next section we briefly review how probabilistic models are used to model sequential processes. Section 6.2 introduces our relational Markov model approach and describes learning and inference in these models in detail. We compare relational Markov models with traditional Markov models in Section 6.4 and conclude with a brief survey of probabilistic models in Section 6.5 and summary in Section 6.6.

6.1 Background on probabilistic models

Markov models [85] are widely used to model sequential processes, and have achieved many practical successes in areas such as web log mining, computational biology, speech recognition, natural language processing, robotics, and fault diagnosis. However, Markov models are quite limited as a representation language, because their notion of state lacks the structure that exists in most real-world domains. A first-order Markov model contains a single variable, the state, and specifies the probability of each state and of transiting from one state to another. Hidden Markov models

[1]Material in this chapter also appears in the 8th International Conference on Knowledge Discovery and Data Mining [5].

(HMMs) contain two variables: the (hidden) state and the observation. In addition to the transition probabilities, HMMs specify the probability of making each observation in each state. Because the number of parameters of a first-order Markov model is quadratic in the number of states (and higher for higher-order models), learning Markov models is feasible only in relatively small state spaces. This requirement makes them unsuitable for many data mining applications, which are concerned with very large state spaces.

Dynamic Bayesian networks (DBNs) generalize Markov models by allowing states to have internal structure [93]. In a DBN, a state is represented by a set of variables, which can depend on each other and on variables in previous states. If the dependency structure is sufficiently sparse, it is possible to successfully learn and reason about much larger state spaces than using Markov models. However, DBNs are still limited, because they assume that all states are described by the same variables with the same dependencies. In many applications, states naturally fall into different classes, each described by a different set of variables. For example, a web site can be viewed as a state space where each page is a state and each hyperlink is a possible transition. Classes of pages for an e-commerce site include: product descriptions, shopping carts, main gateway, *etc.* Variables associated with a product description page might be the product-id, the price, the quantity on hand, *etc.* Variables associated with a shopping cart page include the customer's name, the shopping cart ID, any relevant coupons, *etc.* These variables can help predict a user's navigational patterns, but it clearly would make no sense to associate a price with the site's gateway page or a credit card number with a product description page.

Examples of multiple state classes from other areas include:

Speech and language processing. Parts of speech (*e.g.*, only verbs have tense), semantic contexts (*e.g.*, asking about flights versus asking about hotels), types of discourse, *etc.*

Mobile robotics. Types of location (*e.g.*, indoors/outdoors, offices, laboratories, bedrooms, *etc.*).

Computational biology. Components of metabolic pathways, regions of DNA, protein structures, *etc.*

Process control. Stages of a manufacturing process, machine types, intermediate products, *etc.*

Fault diagnosis. Fault states associated with different subsystems, each with a different set of sensor readings, *etc.*

This chapter proposes *relational Markov models (RMMs)*, a generalization of Markov models that allows states to be of different types, with a different set of variables associated with each type. In an RMM, a set of similar states is represented by a predicate or relation, with the state's variables corresponding to the arguments of the predicate. The domain of each argument can in turn have a hierarchical structure, over which shrinkage is carried out [69]. RMMs compute the probability of a transition as a function of the source and destination predicates and their arguments. RMMs are an example of a relational probabilistic representation, combining elements of probability and predicate calculus. Other representations of this type include probabilistic relational models [47], probabilistic logic programs [76] and stochastic logic programs [75].

We expect RMMs to be particularly useful in applications that combine low-level and high-level information, such as plan recognition from low-level actions, or speech recognition aided by natural language processing. An example of the former is inferring information-seeking goals of web site users from the sequence of links they follow. Doing this inference makes it possible to automatically adapt web sites for different users, and as a result, to minimize users' effort in reaching their goals. RMMs are able to predict user behavior even in web sites (or parts thereof) that the user has never visited before, and are thus potentially much more broadly useful than previous approaches to web log mining, including traditional Markov models.

6.2 Relational Markov models

Recall that a Markov model is a model of a discrete system that evolves by randomly moving from one state to another at each time step. A *first-order Markov model* is a model of such a system that assumes the probability distribution over the next state only depends on the current state (and not on previous ones). Let S_t be the system's state at time step t. Formally, a first-order Markov model is a triple (Q, A, π), where: $Q = \{q_1, q_2, \ldots, q_n\}$ is a set of states; A is the *transition probability matrix*, where $a_{ij} = P(S_t = q_j \,|\, S_{t-1} = q_i)$ is the probability of transiting from state q_i to state q_j, assumed the same for all $t > 0$; and π is the *initial probability vector*, where $\pi_i = P(S_0 = q_i)$ is the probability that the initial state is q_i. Given a first-order Markov model,

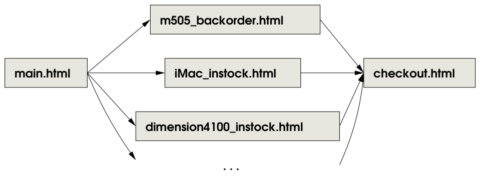

Figure 6.1: **Propositional Markov model for an e-commerce site.** Each box is a PMM state, representing a page in the site. Arrows indicate possible transitions in the PMM, and correspond to hyperlinks in the site.

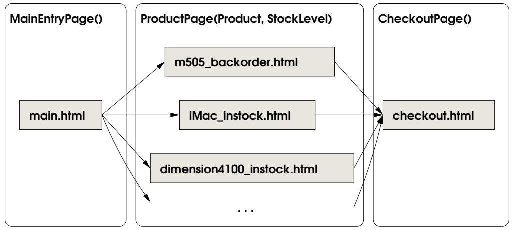

Figure 6.2: **Corresponding relational Markov model.** Each shaded box is a page/state, and states are are grouped (in rounded-corner boxes) by their relations.

the probability of observing a sequence of states $(s_0, s_1, \ldots, s_T)$ is $P(S_0 = s_0, S_1 = s_1, \ldots, S_T = s_T) = P(S_0 = s_0) \prod_{t=1}^{T} P(S_t = s_t \,|\, S_{t-1} = s_{t-1})$. Given a set of observed sequences, the maximum-likelihood estimate of an initial probability π_i is the fraction of sequences that start in state q_i, and the maximum-likelihood estimate of a transition probability a_{ij} is the fraction of visits to q_i that are immediately followed by a transition to q_j. In an nth order Markov model, the probability of transiting to a given state depends on the n previous states, and the transition matrix is $(n+1)$th-dimensional. We refer to Markov models of any order defined in this way as *propositional Markov models (PMMs)*.

Relational Markov models (RMMs) are obtained from the propositional variety by imposing a relational structure on the set of states. For example, consider a Markov model of an e-commerce

web site, in which each page is a state. A PMM would have a unique "proposition" for each page/state: for the main entry page, for each product description page, for the checkout page, *etc.* (see Figure 6.1). In a PMM each state is an atomic entity, and there is no notion of types of states. In contrast, an RMM groups pages of the same type into *relations*, with each relation described by its own set of variables (see Figure 6.2). For example, one relation might be "product description page," with a variable "product" representing the product the page describes, and "stock_level" representing whether the product is in stock or on back order. Additionally, these variables themselves are grouped together, forming a hierarchy of values; Figure 6.3 shows a fragment of such a hierarchy for products at an e-commerce site. A state instance is thus uniquely described as a tuple in a relation instantiated with leaf values from each variable's domain hierarchy. For example, ProductPage(iMac, in_stock) would represent the page describing an iMac computer that is currently in stock at the site's warehouse. Moreover, a tuple using non-leaf values is possible and corresponds to an *abstraction*—a distinguished set of states that are similar to each other in the sense that they have the same type and their arguments belong to the same sub-trees in the domain hierarchies. RMMs leverage these state abstractions for much richer learning and inference than PMMs, and make useful prediction possible in very large state spaces, where many (or most) of the states are never observed in the training data. In this paper, we focus on first-order[2] RMMs, but our treatment is readily generalizable to RMMs of any order. The next subsections describe representation, learning, and inference in first-order RMMs.

6.2.1 *Representation*

Formally, an RMM is a five-tuple $\langle \mathcal{D}, \mathcal{R}, Q, A, \pi \rangle$. $\mathcal{D}$ is a set of domains, where each domain, $D \in \mathcal{D}$, is a tree representing an abstraction hierarchy of values. Each leaf of D represents a ground value. $\mathcal{R}$ is a set of relations, such that each argument of each relation takes values from the nodes of a single domain in $\mathcal{D}$. Q is a set of states, each of which is a ground instance of one of the relations in $\mathcal{R}$, *i.e.*, where each argument is instantiated with a leaf of the corresponding domain. A (the transition probability matrix) and π (the initial probability vector) are the same as in a PMM.

[2]"First-order" is sometimes used in the literature to mean the same as "relational" or "predicate-level," in opposition to "propositional." In this paper we use it in the Markov sense, to denote the assumption that future states are independent of past states given the present state.

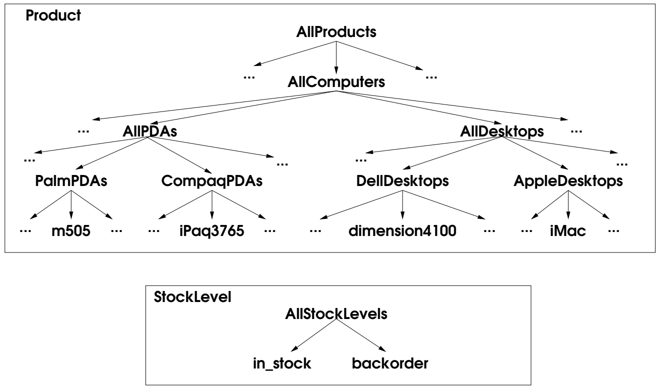

Figure 6.3: **Abstraction hierarchy of products.** Leaves in the tree represent ground values, while internal nodes denote categories of related values.

To continue our simplified e-commerce example, suppose that $\mathcal{D}$ contains abstraction hierarchies for Products and StockLevels as shown in Figure 6.3. $\mathcal{R}$ is the set {MainEntryPage(), ProductPage(Product,StockLevel) CheckoutPage()}, where ProductPage(Product,StockLevel) specifies that the first and second arguments of the ProductPage relation must come from the Product and StockLevel domains, respectively. Q consists of several states, one of which is ProductPage(iMac, in_stock).

We now show how to use the relations and domain abstraction hierarchies to define sets of states as abstractions over Q. These abstractions are distinguished sets of states whose members are similar to each other by virtue of their relations and parameter values. That is, states whose parameter values are in common subtrees of their respective domains will appear in many abstractions together, while states with very different parameter values (or belonging to different relations) will appear together in only the most general abstractions.

We define these abstraction sets by instantiating a relation, R, with interior nodes (instead of just leaf nodes) from the domains of R's arguments. More formally, Let nodes(D) denote the nodes of a domain D. If d is a node in domain D, then let leaves(d) denote the leaves of D that are

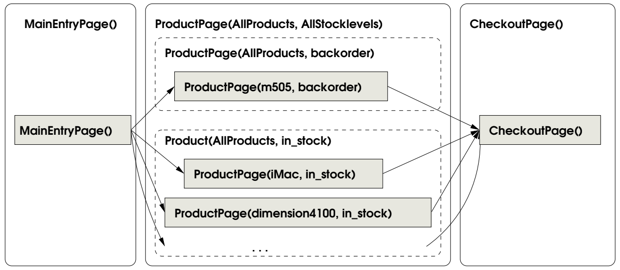

Figure 6.4: **State abstractions for the relational Markov model.** The hierarchy of Figure 6.3 defines abstractions over the RMM of Figure 6.2; the abstractions are depicted as rounded-corner boxes, labeled with their relations and arguments, and surrounding their ground states.

descendants of d. Let $R \in \mathcal{R}$ be a k-ary relation with domains $D_1, \ldots, D_k$. Let $d_1, \ldots, d_k$ be nodes in the corresponding domains. We define the *state abstraction corresponding to* $R(d_1, \ldots, d_k)$ to be the following subset of Q.

$$\{R(\delta_1, \ldots, \delta_k) \in Q \mid \delta_i \in \mathsf{leaves}(d_i),\ \forall i, 1 \leq i \leq k\}$$

For example, given the domain trees shown earlier, Figure 6.4 shows several abstractions for the e-commerce RMM. Note that the abstraction ProductPage(AllProducts,in_stock) is the set of two ground states: {ProductPage(iMac,in_stock), ProductPage(dimension4100,in_stock) }.

Given a particular state $q \in Q$, it is especially interesting to know all the abstractions of which q is a member. Without loss of generality, suppose that $q = R(\delta_1, \ldots, \delta_k)$ and the domains of R's arguments are $D_1, \ldots, D_k$. We then define the *set of abstractions* of q, written $\mathcal{A}(q)$, as the following subset of the power set of Q:

$$\{R(d_1, \ldots, d_k) \subseteq Q \mid d_i \in \mathsf{nodes}(D_i) \wedge \delta_i \in \mathsf{leaves}(d_i), \forall i, 1 \leq i \leq k\} \tag{6.1}$$

For unary relations there is a total order on $\mathcal{A}(q)$, from the most specific ($\{q\}$) to the most general (Q). For n-ary relations, there is a partial order on $\mathcal{A}(q)$ (*i.e.*, $\mathcal{A}(q)$ forms a *lattice* of abstractions).

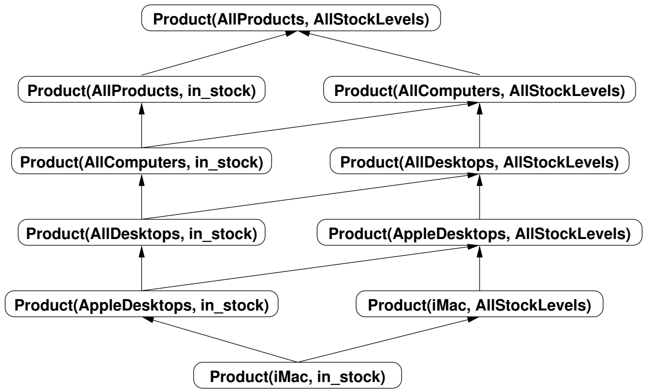

Figure 6.5: **A lattice of abstractions.** Boxes represent abstractions and arrows point in the direction of more general abstractions. These particular abstractions form the lattice for the ground state Product(iMac, in_stock).

For example, the abstractions of Product(iMac, in_stock) are shown in Figure 6.5, where arrows point in the direction of increasing generality. Finally, the *rank* of an abstraction $\alpha = R(d_1, \ldots, d_k)$ is defined as $1 + \sum_1^k \mathsf{depth}(d_k)$, where depth() is defined as the depth of a node in a tree (with the root being at depth zero). The rank of Q (the most-general abstraction) is defined to be zero, and ranks increase as abstractions become more specific.

In the case of finite domains, RMMs are no more expressive than PMMs; given an RMM, an equivalent PMM can be obtained simply by creating a proposition for each tuple in Q. The advantage of RMMs lies in the additional support for learning and inference that the relational structure provides, as described in the next subsection.

6.2.2 Learning and inference

In PMMs, the only possible learning consists of estimating the transition probabilities a_{ij} and initial probabilities π_i, and these estimates can be done reliably only for states that occur frequently in the training data. In many cases (*e.g.*, when modeling a user of a large web site), most states are not

observed in the training data, but it is still possible to generalize usefully from the observed behavior to unseen states. RMMs provide a formal framework for doing this generalization.

For each possible state abstraction α, we can define the corresponding initial probability π_α as the probability that the initial state is an element of α: $\pi_\alpha = \sum_{q_i \in \alpha} \pi_i$. Similarly, for each pair of state abstractions (α, β) we can define the corresponding transition probability $a_{\alpha,\beta}$ as the probability of transiting from a state in α to any state in β:

$$a_{\alpha,\beta} = \sum_{q_i \in \alpha} \left[P(q_i|\alpha) \sum_{q_j \in \beta} a_{ij} \right]$$

where $P(q_i|\alpha)$ is the probability that the current state is q_i given that the current state is a member of α. The abstraction transition probabilities $a_{\alpha,\beta}$ can be estimated directly from the training data by counting. By making suitable simplifying assumptions, they can then be used to estimate the probabilities of transitions that are absent from the data. For example, if we assume that the destination state q_d is independent of the source state q_s given the destination abstraction β, then $a_{sd} = a_{\alpha,\beta} P(q_d|\beta)$, where α is the source abstraction. $P(q_d|\beta)$ can be estimated as uniform: $P(q_d|\beta) = 1/|\beta|$, where $|\beta|$ is the number of states in abstraction β. To make maximum use of all the available information, we propose to use a *mixture model* for each transition probability:

$$\begin{aligned} a_{sd} &= P(S_t = q_d \mid S_{t-1} = q_s) \\ &= \sum_{\alpha \in \mathcal{A}(q_s)} \sum_{\beta \in \mathcal{A}(q_d)} \lambda_{\alpha,\beta} a_{\alpha,\beta} P(q_d|\beta) \end{aligned} \tag{6.2}$$

where the sum is over all abstractions of the source and destination states, and the $\lambda_{\alpha,\beta}$'s are non-negative *mixing coefficients* that sum to 1. The generative model implicit in Equation 6.2 is that, to generate a transition, we first choose a pair of abstraction levels (α, β) with probability $\lambda_{\alpha,\beta}$, and then move to destination state q_d with probability $a_{\alpha,\beta} P(q_d|\beta)$. Effectively, this model performs *shrinkage* between the estimates at all levels of abstraction. Shrinkage is a statistical technique for reducing the variance of an estimate by averaging it with estimates for larger populations that include the target one [69]. Equation 6.2 applies shrinkage across an entire abstraction lattice, rather than over a single abstraction path (as is more usual). For example, a forecast of the number of Apple iMacs sold at a given store can be shrunk toward a more reliable forecast for the average of this quantity at all stores in the same city as the store of interest. The comparative values for the $\lambda_{\alpha,\beta}$'s effectively trade off bias and variance in the probability estimate. Terms corresponding to more general abstractions have lower variance, because they are estimated with more training data, but have a larger bias than terms from more specific abstractions. Thus, good shrinkage weights have two desirable properties: (1) they reduce the influence of abstractions with very little data; and (2) they allow increasingly specific abstractions to dominate as the training set size grows, with the RMM reducing to a PMM in the infinite-data limit. The mixing coefficients $\lambda_{\alpha,\beta}$ can be estimated in a number of ways, corresponding to different variations of our system:

- **RMM-uniform**: Uniformly (*i.e.*, all $\lambda_{\alpha,\beta}$'s are equal). This approach has the advantage of being extremely fast, but may lead to poor results.

- **RMM-EM**: Using the EM algorithm, as described in McCallum *et al.* [69]. In preliminary evaluation this option performed poorly, due to insufficient training data, so we did not evaluate it further.

- **RMM-rank**: Using a heuristic scheme based on the rank of the abstraction. In particular, we experimented with the following method:

$$\lambda_{\alpha\beta} \propto \left[\frac{n_{\alpha\beta}}{k}\right]^{\text{Rank}(\alpha)+\text{Rank}(\beta)} \tag{6.3}$$

  where $n_{\alpha\beta}$ is the number of times that a transition from a state in α to a state in β could have occurred in the data (*i.e.*, the number of visits to a state $q_i \in \alpha$ to which a transition to a state $q_j \in \beta$ is possible), k is a design parameter, and the proportionality constant is derived from the requirement that the $\lambda_{\alpha\beta}$'s sum to 1. This heuristic meets our desiderata for the weights, although many other variations are possible. The choice of k controls how much data must be seen at a given abstraction level before that level can have a significant weight; when $n_{\alpha\beta} < k$, $\lambda_{\alpha\beta} \approx 0$. In experiments with validation data, we have found that setting $k = 10$ works well in practice.

The size of the abstraction lattices, and hence the number of terms in Equation 6.2, increases exponentially with the arity of the source and destination relations. Thus, when these arities are large, and/or when the abstraction hierarchies are deep, it may not be practical to compute all the terms in Equation 6.2. Instead, we can select the more informative ones, and set the mixture weights of the rest to zero (thus ignoring them). An efficient way of doing this culling is to learn a decision tree with the destination abstraction as the class, and the arguments of the source relation as the attributes. We will not use the decision tree for prediction directly, but rather take advantage of its learning machinery for selecting attributes with high information gain. Each node along the path in the tree that the source state follows corresponds to an (α, β) pair that will have a non-zero weight in Equation 6.2. These weights can then be chosen using any of the methods suggested earlier; this approach is simply selecting *which* terms will have non-zero weight.

More precisely, we learn a *probability estimation tree* or *PET* [84], because the goal is to estimate the probability of each destination abstraction, rather than simply predicting the most likely destination. Any set of abstractions that form a partition of the destination states can in principle be used as the class. In this paper, we consider only the highest level of abstraction—the relation $R_d \in \mathcal{R}$ of the destination state. We learn a PET for each source relation separately. The candidate attributes include each argument of the source relation at each level of its domain hierarchy; thus, a k-ary relation each of whose arguments has n abstraction levels yields kn attributes. Figure 6.6 shows an example of such a PET.

To select the most informative terms in Equation 6.2 for a given source state, we consider the path the state goes down in the PET. Each node in the path has an associated probability distribution over destination abstractions (shown in Figure 6.6 in dashed-line boxes for a few nodes), and corresponds to a set of $a_{\alpha,\beta}$ terms in Equation 6.2, one for each destination abstraction. The α corresponds to the decisions made from the root to the node, and β belongs to the set of abstractions that the PET predicts. For example, the highlighted leaf node in Figure 6.6 corresponds to the source abstraction ProductPage(AppleDesktops, in_stock) and (like all other nodes in the tree) includes destination abstractions MainEntry(), ProductPage(AllProducts, AllStockLevels), and Checkout. The terms gathered from all the nodes along the path to the leaf are combined according to Equation 6.2, with the shrinkage coefficients computed using any of the methods described above. We evaluated all three methods, and call these variants **RMM-PET-uniform**, **RMM-PET-EM**, and

RMM-PET-rank. Using EM is possible with the PET approach, because, unlike earlier, we are predicting the destination relation, and not more specific abstractions. Thus, there is sufficient data to perform EM reliably.

More generally, we could build multiple PETs, each one estimating transitions to a different partition of the states; thus, the source state would follow multiple paths, one in each PET. For example, one PET would predict the destination relation, another would predict the destination at a lower level of abstraction (for instance, the relation and one variable's ground value), *etc.* The terms collected from all the PETs would then be combined according to Equation 6.2.

In practice, in large state spaces it is often the case that only a fraction of the states are directly reachable from a given state. For example, on a web site only the pages that the current page links to are directly reachable from it. In this case, the $P(q_d|\beta)$ terms in Equation 6.2 can be replaced by terms that also condition on the knowledge of the set of states $\mathcal{C}(s)$ that are directly reachable from q_s. For states that are not reachable from q_s, $P(q_d|\beta, \mathcal{C}(s)) = 0$. For states that are reachable from q_s, under the previous assumption of uniform probability, $P(q_d|\beta, \mathcal{C}(s)) = 1/|\mathcal{C}(s)|$.

Notice that, in principle, any machine learning method could be used to predict the destination state as a function of properties of the source state. The approach proposed here implicitly encodes the learning bias that the abstraction hierarchies over the relation arguments are useful for generalization (*i.e.*, two states whose values are closer in their respective hierarchies are more likely to transition to the same state than states that are far apart).

6.2.3 Complexity

Inference in RMMs is slower than in PMMs, but is still quite fast. The computation of a single a_{sd} requires combining up to $|\mathcal{A}(q_s) \times \mathcal{A}(q_d)|$ estimates, but each of these estimates is obtained from the learned model by a simple lookup. Thus the overall computation is still fast, unless the abstraction hierarchy is very large. In this case, an approach such as those used by the RMM-PET variants can greatly reduce the number of abstractions that need to be considered, by identifying the few estimates that are most informative. Further, we can trade off time for memory by pre-computing and storing the a_{sd}'s, in which case inference becomes as fast as in PMMs.

Learning in RMMs conceptually involves estimating a transition probability for every pair of

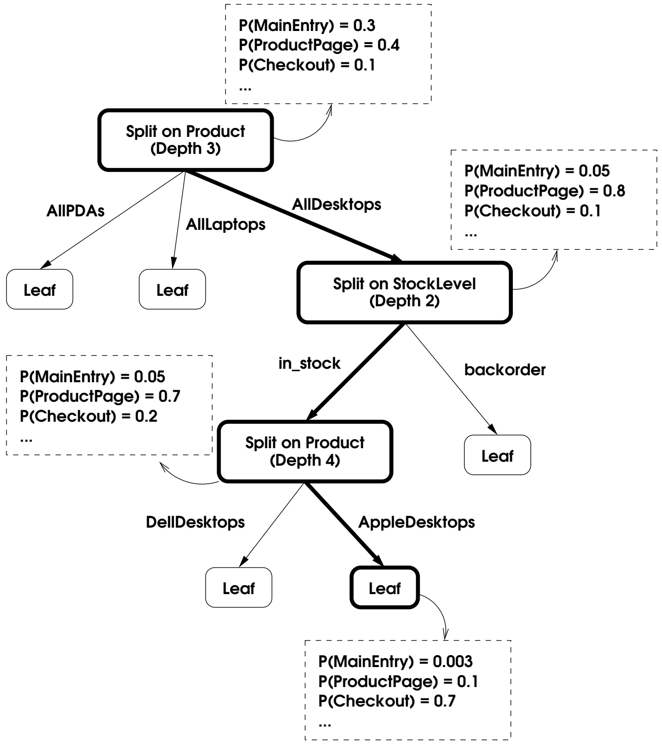

Figure 6.6: **A PET predicting destination relation from ProductPage.** Rounded-corner boxes represent tree nodes. The path the page ProductPage(iMac, in_stock) follows is highlighted. Each node has an associated distribution over the destination relations; these are shown for the highlighted nodes. Each node along this path selects a set of (α, β) abstraction pairs that are combined according to Equation 6.2. The α abstraction for a node is derived from the decisions made along the sub-path from the root to that node, and the β abstractions are the abstractions the PET is predicting (the destination relation in this example).

abstractions, rather than for every pair of states. However, only the transitions between abstractions that actually occur in the data need be considered (in the same way that, in PMMs, only the page transitions that actually occur need be considered). The transition probabilities for non-leaf abstractions can be computed without looking at the data, by aggregating counts for the lower-level abstractions. As a result, the dominant term in the learning time of an RMM is often the computation of the leaf-level probabilities, which is the same as for PMMs.

6.3 Using RMMs for web navigation

The last chapter observed that MINPATH's performance is limited by the quality of the underlying page navigation model, and, as we mentioned earlier, first-order PMMs have a number of weaknesses. The most significant is that PMMs cannot offer informed guidance at pages for which there is no training data. If a web page did not exist during the training period (or simply was not visited), the Markov model can do no better than predict a uniform distribution over the out-adjacent pages[3]. This phenomenon is very common on large and/or dynamically-generated web sites: on a portal site the news stories change every day; customers at an e-commerce site typically view product descriptions they have not previously read; and after a semester is over, students begin viewing the course pages for a different set of courses. Instead, ideally, we would like the model to take advantage of the relational structure of the space of pages. For example, visitors prefer news stories of a particular genre and products of similar types. If a student views numerous homework pages for a particular course in a given department, then the visitor is likely to continue preferring homework pages, pages for that course, and courses in that major.

As we demonstrate in the next section, RMMs effectively address the issue of sparse training data in large sites, by making use of a relational model of the web site identifying semantic correspondence between pages, both previously visited and not visited. The relational model is frequently already available, because such a model is often created when the human designer develops the site's navigational and content structure, or implements the database to serve the content. Based on the formal model from Section 2.3.1, the set of relations $\mathcal{R}$ in a web navigation RMM is an element of

[3]Or, worse: if a page is created and linked from an oft-visited page, then the maximum likelihood or maximum *a posteriori* probability of the new link will be very low.

the site's set of page partitions, Π. In our evaluation we measure the predictive accuracy of RMMs for page navigation, and incorporate them into our MINPATH system.

6.4 Empirical evaluation

In this section, we address the following questions: (1) Is our hypothesis correct that RMMs outperform propositional Markov models when data is sparse? (2) In data-rich environments where PMMs perform well, are RMMs at a disadvantage? (3) Are RMMs competitive in terms of CPU time required for learning? (4) Which of the RMM variants (uniform, rank, or PET) performs best? (5) How well does MINPATH perform when using RMMs versus PMMs?

To answer these questions, we selected four sets of log data taken from two real web sites, `www.gazelle.com` (the e-commerce site introduced in the KDDCup 2000 competition [61]) and the instructional pages from our home institution `www.cs.washington.edu/education/courses/`. At both sites, we explicitly modeled when users ended a browsing trail, by creating a distinguished sink page that was linked from every page in the site and which users implicitly visited at the end of a trail. We represented each page in the site as a state and the input to the models was the links users followed during the training period. The experimental task is to predict the probability a user will follow each link given the user's current page. The KDDCup data has the advantage that it represents the large class of sites dynamically-generated from database queries and page templates, but was not ideal because some domain modeling questions could not be answered without the "live" site[4]. Our home institution's site was useful because it is operational and we have substantial amounts of data available for mining.

For both sites we collected clickstream data and the list of links on each page. Determining hyperlink connectivity was easy at our home institution—we crawled the site and parsed linkage data to create the model. However, although we had log data for `www.gazelle.com`, the site was no longer operational. Hence, we were forced to generate an approximate linkage model composed of the subset of links that were actually followed in the log data. While this solution is suboptimal (even if a link was never followed, its presence may have influenced the behavior of visitors), the

[4]`www.gazelle.com` is now defunct.

alternative (attempting to randomly add spurious but untraveled links to each page) seemed questionable.

Generating good relational structure at each site was straightforward. At our home institution, for example, our model includes CourseOccurrence(Course, Term) pages for the main page of each term's offering of a course, Assignment(Course, Term, Assignment) pages for each problem set assigned, *etc.* Content on `www.gazelle.com`, as at many large web sites, was generated dynamically by combining queries over a database with HTML templates to produce pages. The challenge, however, was in inferring the schemata of pages—the set of allowable templates and the parameters that they each required—without having access to the live web site. Fortunately, the KDDCup log data encodes a comprehensive set of parameters as part of each request, and most of these parameters have an obvious intuitive meaning (page template, product identifiers, *etc.*). We removed records for all but the nine most frequently accessed page templates and for templates whose arguments are not present in the clickstream data (*e.g.*, search results pages); this set of nine templates was our initial candidate for the relation set $\mathcal{R}$. The next challenge was determining the arguments to each relation. By analyzing the frequency of non-null parameter values, it became clear that some of the templates took optional arguments. Because our framework requires relations to have constant arity, we "split" such a relation into two or more relations, one for each non-null argument pattern. This process yielded 16 distinct relations in $\mathcal{R}$. Finally, for the hierarchies over the parameter values, we used the trees defined for those parameters in the KDDCup data. Appendix B provides the detailed relational models for both sites.

In the following experiments, we compared PMMs with five RMM variants: RMM-uniform, RMM-rank, RMM-PET-EM, RMM-PET-uniform, and RMM-PET-rank. We employed Laplace smoothing [52] in the PMM and in the RMM-PET variants. For RMM-rank and RMM-PET-rank we set the k parameter at 10.0, a value which had produced good results on validation data. For each data set, we trained the models with varying numbers of examples, and we recorded the average negative log-likelihood of a test example. A negative log-likelihood score is the number of bits needed to encode an average test instance given the model; a perfect model would have a score of zero.

Our first experiment, which uses KDDCup data from `www.gazelle.com`, shows the substantial advantage that RMMs can have over PMMs (see Figure 6.7). With only 10 training examples,

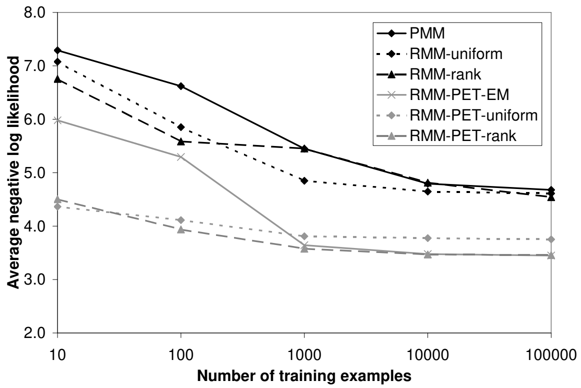

Figure 6.7: **KDDCup 2000 data (`www.gazelle.com`).** The x-axis shows the number of training instances scaled logarithmically, and the y-axis is the average negative log-likelihood of a testing example. Curves are based on 2000 testing instances. RMMs outperform PMMs with as few as ten training examples.

the RMMs perform significantly better than PMMs (the 95% confidence interval around the mean of the pairwise difference lies completely above zero). As the amount of training data increases, all models improve their prediction, but the RMM-PET variants consistently outperform the PMM. In particular, this trend holds true even with abundant training data. This result is somewhat surprising: the RMM-PET variants predict only the destination relation (and subsequently assume all pages with the same relation are uniformly likely). *A priori*, we would expect this restriction to decrease performance, but, instead, this approach apparently allows the RMM to eliminate noisy abstractions and predict navigation much better.

Our second experiment uses log data from November 2001 at our home institution. When trained with successively more data, RMM-rank and the RMM-PET variants showed a slight improvement over PMMs, but only when trained on up to 10,000 examples. Because the UW CSE education pages form a small site, it is a very data-rich environment (*i.e.*, the training data densely covers the visited pages), and we were pleased that RMMs were not trumped by PMMs.

Our third experiment also uses data from our home institution; it represents traffic to the pages of a single course, CSE 142 "Computer Programming I," over a full year. Here, we trained the models on data from the instances of CSE 142 in Winter, Spring, Summer, and Fall 2001, and tested the models on data from the instance in Winter 2002. Note that the instructors (and course webmasters) were different in the various instances; indeed, none of the test pages even existed at the time that the training data was collected. As a result, the PMM can do nothing better than predict a uniform distribution over the links on each page. In contrast, an RMM takes advantage of the related common relational structure of the training and test data to significantly improve prediction (see Figure 6.8).

The computation time required for the RMM variants is not substantially more than that for PMMs. The RMM variants require some preprocessing of the data, to build the abstraction sets, but this work can be done at learning time, independent of the test set. In this third experiment, for example, preprocessing the site (containing 3,749 pages) for RMM-rank and RMM-uniform took four minutes (our RMM code is implemented in C++ and the experiments were run on an 850MHz Pentium III)[5]. Inference in a PMM for a test example requires only a single ratio of counts, while a more complex set of counts must be shrunk together in the RMM variants. However, we found that, on average, RMM-rank, RMM-uniform, and the PMM method all required the same amount of time (about 430 milliseconds) per example for inference. The added cost of the RMM is hidden largely because the work for one test instance may be cached and applied to another (*e.g.*, two instances with a common source state). The RMM-PET variants require a different preprocessing (to learn the PET) which took 76 seconds and completed the prediction runs in 1960 milliseconds.

Finally, we used RMMs to predict navigation for use in the MINPATH algorithm. We used data from February 2002 (the same source as in Chapter 5), but limited the training and testing sets to only those pages in the `/education` hierarchy. We compared MINPATH's performance for four different models: a single unconditional model with URL stemming at 0.1%, a single first-order Markov model with stemming at 0.1%, the RMM-uniform model, and the RMM-rank model. We found that RMMs performed significantly better than the PMMs, allowing MINPATH to save users more links—often 50% to 80% more links than with PMMs, particularly when training data

[5]Our implementation calculates the non-leaf abstraction transition probabilities directly from the data, and not from lower-level abstractions as we suggested in Section 6.2.3. Thus, our computation could be improved substantially.

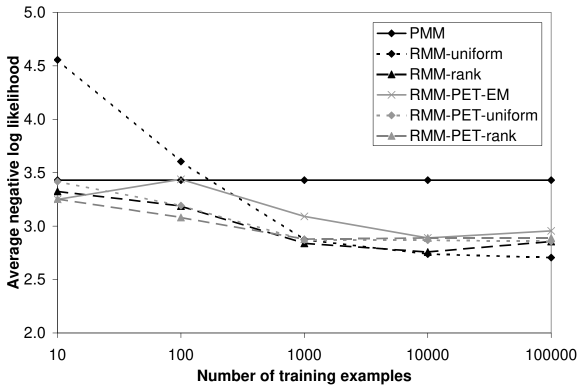

Figure 6.8: **Winter 2002 data from UW course CSE 142.** Pages in the testing set (Winter 2002) did not exist during the training period (Winter 2001 - Fall 2001). RMMs can take advantage of the conceptually similar states to greatly improve prediction.

was sparse. Between the RMM variants, RMM-uniform outperformed RMM-rank in this data set, perhaps because the RMM-rank variant was too eager to increase the shrinkage weights for more-specific abstractions, regardless of how well these abstractions truly predict navigation.

In summary, we conclude that RMMs significantly outperform PMMs when data is sparse and perform comparably when data is rich. Computation time for RMMs is competitive with PMMs, particularly when the training data can be preprocessed. The RMM-PET approach offer significant advantages by selecting the most informative mixture terms, and the EM and rank approaches perform favorably for computing the mixing coefficients.

6.5 A brief survey of related probabilistic models

Considerable work has been performed on a variety of different probabilistic models; we illustrate this space in Figure 6.9. The lower left corner represents a simple model containing a number of states of varying probability. Moving rightward adds *sequence* information and leads to a Markov model. Moving upwards adds *structure* by which we mean the notion of defining the states in terms

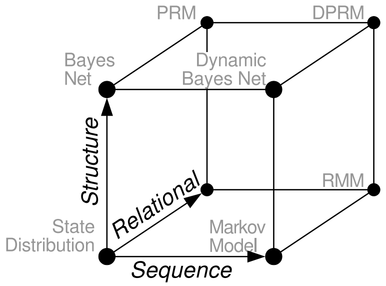

Figure 6.9: **Probabilistic models.** Probabilistic models can be described by their support for three features: sequential processes, relations between variables and states, and structure within a state description.

of variables and representing the joint probability distribution compactly with explicit conditional independence assumptions. Moving backwards into the page adds *relational* information—a set of predicates and a domain of variables for each argument.

Viewed in this context, the connection between RMMs and other first-order probabilistic representations becomes clearer. Friedman *et al.* [47] extended the notion of Bayesian network to propose probabilistic relational models (PRMs). Objects in a PRM are divided into a set of classes, and a different probabilistic model is built for each class, specifying how its attributes depend on each other and on attributes of related classes. Dynamic Bayesian networks (DBNs) [32, 33, 93] form a probabilistic dependency graph for uncertain temporal reasoning. A DBN has a separate Bayesian network for each time step, in which the values of variables for time t can depend on the values of variables in previous time slices. Thus, DBNs "improve" on RMMs in their use of explicit conditional independences amongst a set of variables, but in contrast to an RMM every state in a DBN is treated the same way—it has the same variables and dependencies. To our knowledge, RMMs are the first probabilistic first-order model of sequential processes to be proposed. However, it is interesting to note that dynamic Bayesian networks can be viewed as a special form of PRM where there is only one class (the state) and the only relation is the sequential order between successive states. PRMs have been extended to allow the class to be chosen from a hierarchy [49]. RMMs al-

low hierarchies over the attributes in each class, and combining models at all levels using shrinkage. This approach should be useful in PRMs also. One obvious area for future work is to combine ideas from RMMs, DBNs, and PRMs to define "Dynamic probabilistic relational models" (DPRMs).

Some work has been done in automatically inferring structure among states in a model. For example, TUBA [55] automatically categorizes the states and actions in a model, based on the similarity of the utility function from the states and actions, and builds hierarchical abstractions for each. These hierarchies offer insight to a human data miner in understanding the learned model, and Horvitz and Klein suggest that the utility-based approach can assist in selecting the single best abstraction from which to base a prediction. Automatically building the state abstractions complements and supports our relational Markov model work, but, instead of choosing only one abstraction level, RMMs employ shrinkage to smoothly combine observations at all abstraction levels.

Hidden Markov models have been extended in a number of ways to accommodate richer state and observation information. For example, factorial hidden Markov models [50] decompose model states into k components, described by k state variable, that depend on each other only via the observation variable. A factorial hidden Markov model can be viewed as a special case of an RMM with hidden state, in which all states belong to the same k-ary relation, but which has a conditional independence assumption that state variables in subsequent states depend only on the corresponding variables in the previous state. An area of future work is in exploring how these conditional independences can be leveraged by relational Markov models. Hierarchical hidden Markov models (HHMMs) [41] impose a hierarchical temporal structure on the states, in effect allowing states at higher levels in the HHMM to represent a set of states in a sub-HMM. HHMMs are particularly useful for modeling actions at different length scales, for example speech (the highest-level states are words; each word-state is an HMM of phonemes; *etc.*). The temporal hierarchies are particularly useful as a device to model the underlying process at a number of abstraction levels, but they do not enable a more robust learning such as what RMMs support. Freitag and McCallum [46] build abstraction hierarchies for the observations in HMMs and use these hierarchies to perform shrinkage over the observables.

Other extensions of HMMs have been proposed (*e.g.*, Lafferty *et al.* [62]). It should be possible to subsume these within our framework; this goal is a matter for future research. RMMs are also related to work on abstraction in reinforcement learning (*e.g.*, Dietterich [36], Dzeroski *et al.* [39]), and may be useful in that field.

6.6 Summary

This chapter introduces relational Markov models (RMMs), a generalization of Markov models that represents states with first-order relational predicates and leverages this information for better learning and inference. We believe that RMMs are applicable to a wide variety of domains besides web navigation, such as mobile robotics, speech processing, process control, fault diagnosis, and computational biology.

Our experiments have shown that relational Markov models are a suitable alternative to traditional Markov models—RMMs infrequently perform worse, and can perform much better. When data about all states is available in quantity, or if the relations between states are not reflected in the distribution of the data, RMMs offer no advantage relative to traditional Markov models. However, when data is scarce or non-existent about some states, but abundant for conceptually similar states (based on relational abstractions of the states), relational Markov models significantly outperform traditional Markov models. Intuition suggests that this latter case holds true for the vast majority of web sites, and that RMMs should prove widely useful.

Chapter 7

THE MONTAGE SYSTEM

This chapter presents our final web personalizer, MONTAGE. MONTAGE builds dynamic, personalized web portals for visitors by combining content and links from many sites into one unified view. MONTAGE predicts the user's goal in each browsing session and places appropriate content on the portal based on this prediction. In contrast to MINPATH and PROTEUS, MONTAGE only creates new pages—MONTAGE does not modify existing content. In addition, the expected use of MONTAGE is as the browser start page, so the user will view the personalized portal only once per session, at the beginning. Thus, instead of improving individual site-sessions, MONTAGE aims to improve entire web sessions.[1]

7.1 Introduction

Despite the exploratory ring of the terms "browsing" and "surfing," web usage often follows routine patterns of access. For example, Chapter 1 presented the scenario of a software developer whose browsing is very regular day after day. However, despite the regularity with which users view content, few mechanisms exist to assist with these routine tasks. Lists of bookmarks must be authored and maintained manually by users and are presented in a cumbersome hierarchical menu. Links and content on personalized portals, such as MyMSN [70] or MyYahoo! [95], are more easily navigable, but still must be explicitly chosen and managed by users. Our work on MINPATH can improve routine tasks only if the goal is a single target on a site. If the task is to view many pages (*e.g.*, "interesting news stories"), or a series of pages, then MINPATH can offer little assistance.

The challenge that we tackled is to develop tools that assist users with *routine web browsing*. Routine web browsing refers to patterns of web content access that users tend to repeat on a relatively regular and predictable basis (for example, pages viewed at about the same time each day, or in the

[1]This chapter was substantially published in the 11th International Conference on the World Wide Web [6].

same sequence, or when working on the same task, *etc.*). In responding to this challenge, we pose the following three hypotheses:

- **Hypothesis 1**: Users want "one-button access" to their routine web destinations (*i.e.*, users want to minimize their effort in retrieving and viewing content).

- **Hypothesis 2**: Tools for routine web browsing are enhanced by tailoring links and views to a user's current browsing context (as opposed to displaying a static set of content under all circumstances).

- **Hypothesis 3**: Past web access patterns can be successfully mined to predict future routine web browsing.

To test these hypotheses, we designed and implemented the MONTAGE system. MONTAGE builds personalized web portals for its users, based on models mined from users' web usage patterns. These portals both link to web resources as well as embed content from distal pages (thus producing a *montage* of web content; see Figure 7.1). After completing the MONTAGE prototype, we fielded the system to evaluate its effectiveness as a tool and to explore the validity of our hypotheses.

In the next section, we discuss routine browsing in more detail, followed by an outline of MONTAGE in Section 7.3. We discuss implementation specifics of MONTAGE in Section 7.4 and present our experimental findings in Section 7.5. Section 7.6 concludes with a brief summary.

7.2 Routine browsing

As we noted earlier, not all web usage is random or novel; web users also tend to revisit sites and pages in a regular, predictable manner. In the example of the graduate student presented in Chapter 1, the pages the student viewed depended entirely on his current *context*, where context is taken as the time of day and the general topic of the pages viewed previously. More generally, we define the context of a web browsing session as the set of attributes that influences (either consciously or subconsciously) the selection of pages a user views in that session. Many factors can be included in a formalization of context. For instance, the context can include the time of day, the period

of time elapsed since the last session, the general topic of the last session, the most recent non-browsing computer activity (*e.g.*, the most recently viewed e-mail message), *etc.* We define routine web browsing as the overall pattern of content access that a user performs whenever in the same or similar contexts. For example, if a user reviews a stock portfolio at around 1:15 P.M. every day, then viewing the stock portfolio is a routine behavior—it happens at about the same time each day. On the other hand, if the user spends an hour searching for information about fishing in the Pacific Northwest on only one day, then this behavior is not routine because the user does not repeat it in a similar context.

Our intuition strongly suggests that routine browsing is a common mode for interacting with the web; our informal surveys early in our study suggested that many people tend to view the same page or constellation of pages when in similar contexts. The more important questions that must be answered are whether we can formulate good notions of context, identify the routine browsing associated with each context, and leverage this information to assist the user. These questions are embodied in the hypotheses we posed in the introduction and are answered with our experiments with the MONTAGE system.

7.3 The MONTAGE system

A web montage is a page that offers "one-stop shopping" for users to find the information they want. A montage combines content from many different pages, linking to pages or embedding distal content, saving the user the need to follow even a single link to view content. A montage is more than "just a list of favorites," for three reasons. One reason is that remote pages may be embedded in the montage, aggregating content from multiple sites. Another is that the content depends on the user's context of browsing (*e.g.*, the topic of recently viewed pages, or the time of day, *etc.*), so the content is relevant to the task at hand. Third, MONTAGE automatically selects and manages the appropriate content, without the need for explicit input from the user. In contrast, the browser favorites list requires much setup and maintenance by the user to be useful, and this requirement is often more effort than users are willing to expend.

Three typical montage views are shown in Figures 7.1, 7.2, and 7.3. Figure 7.1 is the "Main Montage" which displays links and embedded content grouped by topic. This particular montage has

Figure 7.1: **Main montage.** Content is grouped by topic, and each topic heading links to a topic-specific montage. The page with the highest expected utility for the user is embedded on the montage itself; other relevant pages in the same topic are included as links on the montage.

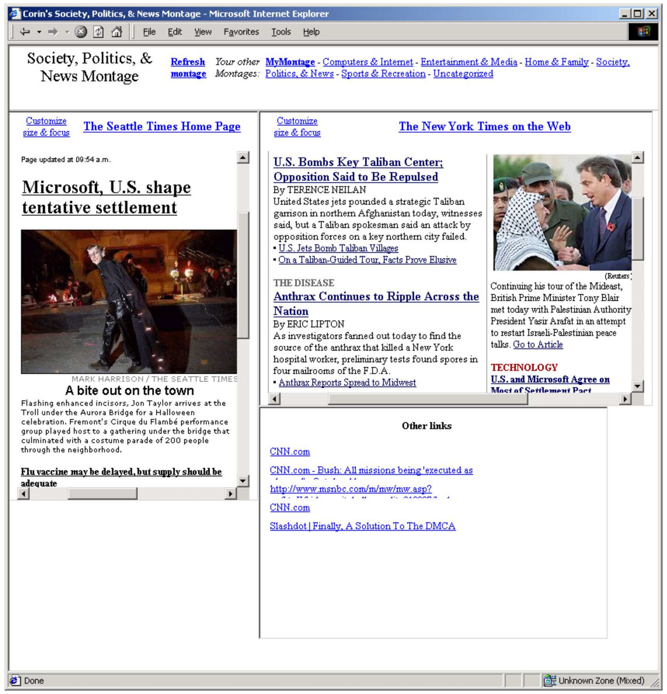

Figure 7.2: **A topic-specific montage.** This montage embeds content from several pages and includes a pane containing many links to other pages in this same topic.

Figure 7.3: **A links-only montage.** Embedded content is omitted to allow many more links in this view. Links are grouped by topic, and MONTAGE selects the placement of topic regions on the display. Link anchors are the destination `<title>`, if it exists, or the URL itself.

three topic-specific panes: "Society, Politics, & News," "Computers & Internet," and "Entertainment & Media," each containing a cropped view of a distal web page and links to other pages within the respective topic. Thus, the user can immediately view the afternoon's current news, the user's most frequently-viewed programming documents, and the current traffic conditions around the area. The position and selection of content is determined automatically by MONTAGE. The user can set the size and location of the "lens" on the remote content page, or accept MONTAGE's defaults (we discuss the user interface for adjusting lenses in Section 7.3.3). In Figure 7.2 is a topic-specific montage, which shows several embedded pages and related links. Both the topic-specific montage and the main montage are assembled automatically to fit just within the user's current browser window, to eliminate the need to scroll the page. The user can navigate between these montages by following links at the top of each view. Figure 7.3 shows a simplified montage that contains only links. Still other visualizations are possible: a browser toolbar showing iconic representations of pages; a persistent display of bookmarks auxiliary to the browser window; *etc.* We plan to explore these alternatives in future work.

Building a web montage is a two-step process, as we described in Chapter 1. We summarize how MONTAGE carries out these steps next.

7.3.1 Step 1: Mine the user model

The primary source of information for the user model is the sequence of pages the user requested. MONTAGE records the time and date of each page visit, the page's URL, and the topic of the page's content (this data is the context ϕ of the request, as described in Section 2.3.1). The topics can be determined using text classification procedures [25, 27, 71]. We applied a content classifier trained to assign topics to web content, developed by Chen and Dumais [28]. The topic classifier employs the linear support vector machine (SVM) method and assigns each page a probability of being in each category of a static topic ontology[2]. We took as the topic of a page the category with the highest probability.

[2]An extension to MONTAGE work would be to adaptively adjust the level of detail in the topic ontology, specializing into subtopics any topics that are overpopulated, while leaving large, sparsely-populated topics general. Although we used high-level categories for our initial user study, our content-tagging subsystem tags content more finely, employing a hierarchical ontology of concepts.

The result of evidence collection is a sequence of requests tagged by topic that MONTAGE further refines into *sessions*. For MONTAGE's purposes, a session is a sequence of page requests that begins with a visit to the user's *start page*—the first page the browser displays, or the page visited when the user clicks the "home" button. In practice, MONTAGE does not know which page is the start page and, thus, uses heuristics to identify when one session ends and when the next session begins. Section 7.4 provides the exact details of how we clean and segment the data into sessions.

MONTAGE uses the page sequences and sessions to compute five aspects about the user for the model. We summarize these aspects here; Section 7.4 describes how these aspects are computed.

- **Candidate pages.** MONTAGE selects a subset of the user's previously visited pages as candidates for inclusion in the montage. MONTAGE places no upper limit on the number of pages selected, but does set minimum requirements for inclusion in this set (such as a minimum number of times the user has ever viewed the page).

- **Interest in page.** MONTAGE estimates the user's apparent interest in each page, primarily by how much time the user spent looking at the page, how many links the user followed from the page, *etc.*

- **Interest in topic.** MONTAGE also estimates the user's interest in the higher-level topic of pages viewed. Because, for instance, although the user may show relatively little interest in several different specific pages, he or she may be strongly interested in the single topic encompassing them all.

- **Probability of revisit.** MONTAGE estimates the probability that a user will revisit a page in the next browsing session, given the user's current context.

- **Savings possible.** Placing a link or embedding a page on the montage saves the user navigational effort. MONTAGE measures this savings as the effort the user expended navigating to the page originally. All things being equal, MONTAGE will favor including pages on the montage that would be difficult to revisit manually.

7.3.2 Step 2: Assemble the montage

Equipped with the user model, MONTAGE is ready to assemble the content montage. Because the montage depends on the user's current browsing context, MONTAGE builds a new page each time the user revisits his or her montage. MONTAGE begins the assembly by calculating the overall expected utility of viewing each content topic or candidate page. We take a simplified approach compared to PROTEUS's utility function, because the motivation here is in evaluating pages previously visited, and not necessarily the navigation through the site. In particular, we ignore extrinsic utility, and compute the probability of visiting a page by conditioning on the browsing context. The cost of navigation is incorporated into our measure of intrinsic utility. Another advantage of this model is that, because we are concerned only with estimating the value of revisited pages, and not novel pages, we can do so independently of all other pages (*i.e.*, because we do not explicitly model navigation). This model will allow (for these specific pages) more effective learning than possible with MINPATH or PROTEUS (which require more training data, to build a model of navigation).

We approximate the value of the page p to a user as a function of the interest, $I(p)$, and the navigation savings, $S(p)$. In the general case, the value is some combination of these two factors: $f(I(p), S(p))$. We treat these factors as independent and have explored both a weighted additive model and a weighted multiplicative model[3]. In our experiment, we chose the multiplicative model, and took as the value of a page, $I(p)^{k_1} \times S(p)^{k_2}$. If we assume the utility of displaying uninteresting content to be zero (and not, for instance, negative), then the expected utility of a page is the product of the probability the user will visit the page given the current context, $Pr(p|\mathcal{C})$, and the value of the page. Thus, we take as the expected utility of a page:

$$E[U(p)] = Pr(p|\mathcal{C})(I(p)^{k_1} \times S(p)^{k_2}) \tag{7.1}$$

Likewise, we compute the expected utility of a topic $\mathcal{T}$ as the product of the probability that the user

[3]Of course, an additive model can be viewed simply as the logarithm of the multiplicative model. The significant difference arises when computing expected utility: in a multiplicative model, doubling the interest in a page, for example, always doubles the expected utility, but in an additive model, one term may dominate, and thus doubling the other will have little effect on expected utility.

will view any page with topic $\mathcal{T}$ in the current context, $Pr(\mathcal{T}|\mathcal{C})$, and the user's interest in the topic:

$$E[U(\mathcal{T})] = Pr(\mathcal{T}|\mathcal{C})I(\mathcal{T})$$

This choice has the effect of favoring "larger" topics, because every page with non-zero visit probability and interest will positively contribute to $E[U(\mathcal{T})]$ (by increasing either $Pr(\mathcal{T}|\mathcal{C})$, $I(\mathcal{T})$, or both). Moreover, this behavior is what we desire: even if all pages of some topic $\mathcal{T}$ individually have low re-visit probability and interest, if the user has viewed many more pages of topic $\mathcal{T}$ than any other, than that topic has high expected utility for the user and should be represented in the montage.

MONTAGE uses these computed values to place content on the montage to maximize the total expected utility, subject to the sizes of the browser window and each of the embedded pages. Effectively, MONTAGE solves a 0-1 knapsack problem where the knapsack is the browser window area and each item is a candidate page or topic with associated size (specified by the user or from a default setting) and utility (computed as above). In practice, we could provide users with tools to inspect and tune measures of interest and navigation savings, and allow them to provide feedback on the function for combining these factors. For example, for the multiplicative model, we could assess from users the relative weighting ascribed to interest versus navigation savings. For our studies, we considered these factors to be equal and provided fixed functions for interest and savings. In Section 7.4 we describe how MONTAGE mines the interest, savings, and probabilities from the web usage logs.

7.3.3 User control of montage clippings

In the previous section, we saw that the content embedded on the montage was cropped to a web page *clipping*, a smaller window than the original page (see Figure 7.4). MONTAGE allows the user to have complete control over the size and position on the distal page of this clipping. By specifying the length, width, and focal point of the clipping, users create persistent lenses onto particular portions of the content of pages. Users can also dictate the frequency with which the content refreshes itself during the day (*i.e.*, if the user simply leaves his or her browser at the montage, MONTAGE will automatically refresh the embedded content with the given frequency).

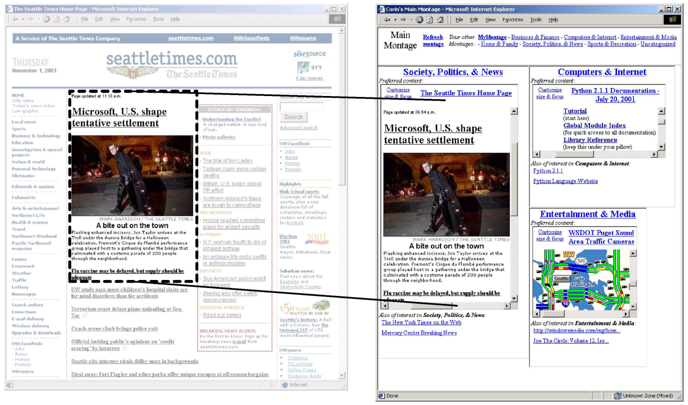

Figure 7.4: **Cropping content.** Distal pages are cropped to a small window when embedded on the montage.

7.4 Implementation

The implementation of MONTAGE follows the framework we presented in the previous section; in this section we describe the specific details. MONTAGE was coded in Python and runs on both Windows and Linux platforms using Internet Information Services (IIS) or Apache web servers.

7.4.1 Collecting data and mining models

Users of the MONTAGE system direct all of their web browsing through a proxy that logs each request. In our experiments, we used a single proxy running on a central server, although the MONTAGE framework supports running the proxy on individual users' computers. An important advantage of the individual proxy installation is user privacy—if the proxy and the rest of the MONTAGE system all operate on the user's computer, then the user minimizes the risk of unintentionally sharing private information with third parties. We chose the centralized proxy approach for convenience of the experiment.

MONTAGE cleans the logs following the techniques described in Appendix C. With the proxy logs cleaned and sessionized, MONTAGE proceeds to select the candidate pages and topics. Any page or topic that has been visited more than once is a candidate. For each candidate, MONTAGE builds a naïve Bayes classifier [38] to estimate the probability the user will view the page in a future context. The model classifies a session as to whether the user will view the page or topic in that session. The particular evidential features employed are: the overall rate with which the user views the page; the rate of viewing the page for each 3-hour block of time in the day (*i.e.*, midnight – 3:00 A.M., 3:00 A.M. – 6:00 A.M., *etc.*); and the predominant topic of the pages viewed during the last 4-hour block of time. We evaluated many other features, such as the time since the previous session and the last URL visited, but these other features either offered less lift in predictability or required more training data than we had available.

The final aspects of the user model are the savings possible when embedding a page on the montage and the user's interest in a page or topic. We estimated the savings as the average number of links followed to reach the candidate page from the first page in any session that includes the candidate page. The user's interest in a page is estimated heuristically as a weighted sum of the average number of links followed from the page, $L(p)$, and the average number of seconds spent in sessions starting with the page, $D(p)$:

$$I(p) = L(p) * 0.50 + D(p) * 0.03$$

The constants were chosen to equate an average of two links followed from p with an average session time length of 30 seconds. We chose these number empirically; an extension of MONTAGE would expose these values to the user to enable more direct control over this trade-off. The user's interest in a topic $\mathcal{T}$ is the sum of interest over all pages whose topic is $\mathcal{T}$:

$$I(\mathcal{T}) = \sum_{p \in \mathcal{T}} I(p)$$

We sum, instead of average, to reflect the user's cumulative interest in a topic. If the user has low interest in many pages all about one common topic, by virtue of viewing the large number of pages, we presume the user has a higher interest in the common topic.

7.4.2 Displaying montages

As the browsing context is potentially different each time the user requests his or her montage, the montage may be rebuilt frequently. Our implementation requires only a few seconds to rebuild a montage, with the time dominated by solving the knapsack problem to place embedded content and topic panes in the browser window. We are confident we could improve this time by an order of magnitude with judicious optimization. In our experiments, we rebuilt and cached test subjects' montages only once per hour, both for convenience and because the set of features we chose for browsing context does not change faster than about once per hour.

As we described previously, we developed two different visualizations for a montage: the embedded-content montage (Figure 7.1) and the links-only montage (see Figure 7.3). The links-only montage is quite simple to display: it is a two-dimensional table and contains only links to web sites—no embedded content, images, Javascript, *etc.* The link anchors are chosen as either the target page's `<title>` or, lacking a title, the URL itself.

The embedded-content montage is a bit more complex. It is formed as a set of nested `<frame>`s: the navigation bar, each topic pane, and the content panes within each topic pane, are all `<frame>`s. The hosting `<frameset>`s specify the size of each pane; it is this mechanism that also sets the size of the cropping window for distal content. To scroll the content to the appropriate position on the distal page, MONTAGE sets the `src` of the frame to be the corresponding URL and additionally adorns the URL with a tag the MONTAGE proxy intercepts (recall that the user directs all browsing through the proxy, including requests made for content embedded in the montage). The MONTAGE proxy passes the request along to the appropriate server (removing the adornment) and inserts a small amount of JavaScript into the resulting HTML stream sent back to the user. The proxy makes no other changes to the returned HTML, but the grafted JavaScript will scroll the page to its appropriate position as it is loaded by the browser. We note that an alternative approach would be to pass the URL directly to MONTAGE and have it fetch the page and modify the content itself—no need for adorning URLs or intercepting requests with the proxy. However, because the URL the browser would see is a MONTAGE page, rather than the actual target site, the browser will not communicate any cookies to the remote server. Thus, to ensure the browser *believes* it really is communicating directly with the remote site, we chose the URL adornment approach.

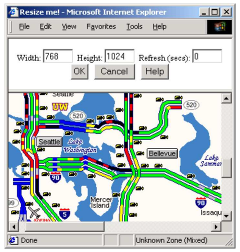

Figure 7.5: **Customizing embedded content.** The user can change the cropping window on the distal page by simply resizing and scrolling the browser window.

The user may change the size and position of the cropping window on distal content by clicking the "Customize size & focus" link in the upper-left corner of any content pane. In response, MONTAGE opens a new browser window as shown in Figure 7.5. The user can control three aspects of how the content is displayed in the montage. First, the user can directly change the size and position of the clipping window simply be changing the size and scroll position of the browser window: drag the window larger, and the clipping window becomes larger. Second, the user can control how text flows on the page by specifying the width and height (in the text fields in the figure) of the *virtual browser window* the page is rendered in. For example, if the user wants to crop the content very narrowly, he or she could specify the virtual browser be only of width 400 (pixels) for that page; the page would then be formatted very narrowly.[4] Finally, the user can control how often each clipping reloads itself in the browser window by setting the period, in seconds; zero seconds disables auto-refreshing. By default, MONTAGE sets this value to zero (refreshing disabled), although MONTAGE could suggest a refresh interval based on the user's past revisit frequency to each page.

[4]Technically, each embedded clipping is placed within an `<iframe>` on a page sourced by a `<frame>`. The width and height fields the user enters simply control the size of the `<iframe>`; the actual browser window size controls the size of the `<frame>`.

7.5 Experimental evaluation

In Section 7.1, we presented three hypotheses about users' routine browsing behavior. A key step in answering these questions is actually fielding MONTAGE among users. We conducted a two-week user study of 26 web-savvy users during early October 2001. Users had extensive experience on the Web but varied in the amount of daily browsing activity. The users were computing professionals and, by large, held a "time is money" philosophy (*i.e.*, waiting for web pages to load or hunting for the right link to follow on a page is a wasteful experience). Thus, MONTAGE could significantly improve their browsing experience by making their routine web browsing more efficient.

During the study, users directed all their web browsing through a central proxy running on our "MONTAGE server" (which ran the proxy, a web server, and the MONTAGE implementation). In the first week, we only collected usage information—users browsed the web as they normally would. At the end of the first week, we began building models of each user, once per day, to use with MONTAGE. Recall that, although the user's browsing context changes relatively frequently, the predictive model for the user does not—a single additional hour or half-day of browsing rarely changes the model dramatically.

During the second week, we presented users their montages. We instructed users to make their montage their browser's start page, and to revisit the page at least a few times each day. We also added an additional pane to the montage to elicit feedback whenever the user viewed the page (a simple rating reflecting how "pleased" the user was with that montage, ranging from 1 meaning "Not pleased at all" up to 7, "Very pleased"). During this period, 21 of our 26 users actually viewed their montages, on average 25.1 times in the week, and provided feedback 28% of the time (we attribute most of the missing 72% of feedback to users simply forgetting to do so). We concluded the study with a questionnaire.

To explore the validity of our hypotheses from Section 7.1, we tested two variables influencing the montage. First, we varied the montage visualization approach: embedded-content or links-only. Second, we varied the model complexity: the expected-utility, context-sensitive model ("complex model") as described in Section 7.4, or a simple model using only the overall frequency of revisits to determine the probability of returning to the page (*i.e.*, ignoring interest in page and savings). We presented every user in the study with two different styles of montage, drawn from the four

Table 7.1: **Study groups.** Each group viewed two montage variations: the embedded-content / complex model montage and another as shown in this table. We count a participant as active if he or she viewed and rated at least one montage during the study. On average, users rated their montage 8.06 times in the study.

Group	Visualization	Model	Number users
1	Links-only	Simple	6
2	Links-only	Complex	6
3	Embedded-content	Simple	6

Table 7.2: **Feedback scores.** Users rated their montages between 1 ("Not pleased at all") and 7 ("Very pleased").

Visualization	Model	Average score
Links-only	Simple	3.32
Links-only	Complex	4.97
Embedded-content	Simple	1.88
Embedded-content	Complex	3.22

total configurations; one style in the first half of the second week and the second style in the second half. We particularly sought feedback on the embedded-content / complex model style, so each user viewed that montage, and one other style. Table 7.1 shows our resulting study groups.

7.5.1 General results

Our study groups were quite small, so our results can give only a general impression of the comparative trends. In future work, we would like to conduct a much more extensive user study. Table 7.2 shows the average feedback score for each of the four montage styles. Recall that a higher score means the user was more pleased with his or her montage. Table 7.3 shows the scores comparing only one variable at a time, and Table 7.4 displays each study group's score for their respective styles. Overall, the links-only montage using the model that incorporates context appears to be the favorite. We shall next analyze how these findings apply to our hypotheses.

Table 7.3: **Results by variable.**

Visualization	Average score
Links-only	4.40
Embedded-content	2.98
Model	**Average score**
Simple	2.64
Complex	3.79

Table 7.4: **Results by study group.** Each study group viewed two different montage styles, switching half-way through the second week of the experiment.

Visualization	Model	Grp 1	Grp 2	Grp 3
Links-only	Simple	3.32		
Links-only	Complex		4.97	
Embedded-content	Simple			1.88
Embedded-content	Complex	3.08	4.00	2.50

7.5.2 Hypothesis 1

Users want "one-button access" to their routine web destinations

As a result of our study, users could access their routine destinations in four different ways: follow a bookmark; follow a link on the links toolbar (a small toolbar that displays only four to six bookmarks); follow a link on a links-only montage; and view content or follow a link on the embedded-content montage. Although measuring bookmark and links toolbar usage was outside the scope of our original study, users' qualitative feedback indicates that users value the montage higher than the manual approaches. Some users suggested a hybrid, where they can manually add links to the automatically-generated montage. We are considering such an extension to our system.

Between the two visualization approaches, we were surprised to find that the links-only montage was unanimously preferred over the embedded-content montage (see Tables 7.2 and 7.3). *A priori*, we had expected users to prefer their target content embedded directly on their montages. Instead, users apparently do not mind following at least one link to view their destination.

We believe several factors may have influenced users' opinions on this issue. One is that the links-only montage loaded nearly instantly as it is only formatted text. The embedded-content montage, however, would take up to 30 seconds to load completely (downloading all the distal content embedded on the page). It's not clear how much of this delay was caused by our particular experimental setup (*e.g.*, was the proxy a bottleneck?) and how much delay is inevitable (*e.g.*, network delays). We plan to investigate this issue in a future study by prefetching the content embedded on the montage.

Another reason is that the links-only view allowed MONTAGE to place many more links on the display than with the embedded-content view. With more links to choose from, users had a greater chance of finding a link to their destination immediately, without drilling down into a topic-specific montage. Due to a someone large default size of the embedded-content web clipping, only one or two clippings could fit on a single montage, unless the user customized the clipping size. We plan to investigate other means for providing greater numbers of clippings, including rendering scaled-down views of portions of pages.

7.5.3 Hypothesis 2

Tools for routine web browsing are enhanced by tailoring links and views to a user's current browsing context

The test for this hypothesis is the comparison between our complex model and the simple one. The complex model conditions the links and content on the montage on the user's current context, while the simple model ignores context. The latter half of Table 7.3 shows this comparison: the model conditioned on context outscored the simple model by well over a point. Of course, this result only scratches the surface—context is clearly useful, but *what* context should we use? In a future study, we plan to evaluate how much predictive lift each feature offers to determine the most valuable set of features for the user model.

7.5.4 Hypothesis 3

Past web access patterns can be successfully mined to predict future routine web browsing

The proof of this hypothesis lies in how often users found their target content on or using their montage. Based on a post-study survey, many users agreed that they found the montage helpful in

suggesting pages they indeed wanted to visit. However, there were several cases in which MONTAGE suggested pages that were clearly never going to be revisited. In particular, MONTAGE tends to suggest search engine results pages to users, as MONTAGE estimates the user's interest in these pages highly (MONTAGE estimates interest in part as the number of links followed from the page). We plan to refine MONTAGE's interest estimate in a future implementation and test this hypothesis further.

Quantitatively, 73% of users responded in our post-study survey that they felt that MONTAGE selected appropriate links and content at least occasionally. We feel that this result supports our initial approach to mining routine web browsing. Moreover, we can improve this value in two ways. First, due to a technical issue, MONTAGE does not suggest intranet content; omitting these pages accounted for 18% of the sessions in which MONTAGE failed to suggest the correct target. With the experience of the user study, we are confident that we can overcome this technical detail and expand the universe of content MONTAGE can offer. Second, 45% of the missed sessions led to pages that users had visited sometime earlier in the study. With a longer history of web usage, and perhaps a more sophisticated user model, MONTAGE could potentially suggest useful content for many of these sessions.

7.5.5 Discussion

Overall, it appears that the MONTAGE user modeling components work well—the context-sensitive model scored higher than the simpler model. Additionally, users are concerned that their start pages load quickly; 64% of our users indicated speed of loading the start page as "very important." This concern is more important, in fact, than having the target of their session display in the start page. We are thus interested in incorporating MONTAGE with a web prefetching system to greatly improve the load time of the embedded-content montage. One could also extend the utility model (Equation 7.1) to incorporate the cost of waiting for the MONTAGE to load the remote content, in the same way that PROTEUS incorporates the cost of navigation actions (Equation 3.1).

Users appreciated MONTAGE's efforts to automatically select and place content on the screen, but they still want some manual authoring mechanism. A number of users point to their links toolbar as what they feel MONTAGE must compete with. Although it is true that MONTAGE was able to identify a number of these preselected shortcuts automatically, it is clear that users would be more inclined to adopt a hybrid system that also allows for direct manipulation and authoring of content to include.

7.6 Summary

The goal of MONTAGE is to improve the experience of routine web browsing—the browsing that users tend to repeat over and over in similar situations. We implemented the MONTAGE prototype system by coupling proxy-based monitoring and predictive user modeling machinery with layout and display techniques that compose web montages. We posed several hypotheses about routine web browsing and addressed them in a user study. We found that users appreciated having an automatic system that could suggest links for them to follow, given a broad view of their current browsing context. However, we found that users particularly wanted the system to operate as fast as possible, being effectively transparent in there access to the Web. We found that loading the montage page should take no more than one or two seconds, and users wanted some manual authoring capabilities. These findings encourage us to push forward in a number of directions with MONTAGE, as we have highlighted in this chapter and outline in Section 9.4.

Chapter 8

RELATED WORK

In this chapter we compare our research with relevant current and recent work. We previously discussed how relational Markov models compare to other probabilistic models; here, we survey other web personalization techniques.

8.1 Adaptive hypermedia

The goals of adaptive hypermedia ally closely with our work: to improve web (hypermedia) interactions by modeling users and adapting the experience. Perhaps the most significant difference lies in the application domain. Most adaptive hypermedia work seeks to improve: online educational systems (*i.e.*, adapt the learning to the student); help systems (adapting to the particular context of the help request); information retrieval (helping users find as much relevant content as possible); or online information systems (helping users find high-quality content quickly). Brusilovsky provides an excellent overview [19] of these systems, and the May 2002 issue of the *Communications of the ACM* [18], as well as the book by Brusilovsky *et al.* [17], feature contributions from several of the most prominent members of this community.

8.2 Personalized content and portals

Our work has been inspired by Perkowitz and Etzioni's adaptive web sites [80, 82, 79]. Our work differs from theirs on three important points:

- Adaptive web sites find singular transformations that appeal to all visitors at the site, while PROTEUS, MINPATH, and MONTAGE personalize the web content for several sizes of audience, from all visitors down to each visitor.
- The only transformation Perkowitz and Etzioni's adaptive web sites produce is synthesizing

index pages—hubs of links to other pages in the site. Our personalizers consider a much wider range of adaptations, that personalize individual pages (*e.g.*, elide-content), site-sessions (add-shortcut), or entire browsing sessions (embedding distal content into a single portal).

- Their site evaluation metric measures only the navigation aspect of the site's visitors. Our approach, estimating the expected utility, combines visitors' content interests with a model of their navigation behavior, and is applicable both as individual pages change content, and to newly created (or never-before visited) pages.

Another proxy-intermediary approach is the Web Intermediaries (WBI) [11, 67] project. WBI proposes an architecture of pluggable intermediaries that exist between web servers and web clients. These intermediaries generate, transform, and monitor the content they see in connection with requests made from web visitors, and can be used either individually or in chains. In many ways, our approach to web site personalization can be viewed as a particular transformation intermediary, providing highly personalized content for users. An interesting line of future research would be to reimplement PROTEUS in the WBI framework and investigate what new capabilities are available to it as a member of WBI's federation (in particular, what other intermediaries are available that could be usefully composed with PROTEUS, MINPATH, or MONTAGE).

The PersonalClipper [44] allows visitors to build their own custom views of web sites by recording navigational macros using a VCR-metaphor and selecting components of the target page to view with a mobile device. In contrast to PROTEUS, the PersonalClipper requires the visitor to *manually* create these navigational macros and place the result on the personal clipping. However, based on feedback from our MONTAGE study, it would be interesting to combine PersonalClipper with PROTEUS or MONTAGE to support a mixed-initiative personalized experience.

Among research on personalizing web portals, the most similar in spirit to MONTAGE are PersonalClipper [44] (or *MyOwnWeb* [8]) and the Web-based Object-Oriented Desktop (WOOD) [23]. The PersonalClipper architecture relies on *site descriptions*, which are essentially programs that run on a web site (following links, filling in forms, *etc.*) and produce a block of HTML as output (a *clipping*). These site descriptions are built by the user following a VCR-metaphor [9]: the visitor starts recording a descriptor, navigates the site to find a destination page, and ends the descriptor by

selecting the HTML fragment that is of interest. Users of PersonalClipper would select the site descriptions desired on a start page, and the system would execute the site descriptions and concatenate the results for display. The WebViews [45] project extends these ideas to allow for parameterizable site descriptions (*e.g.*, a macro for selecting airline flight status given the carrier and flight number) and to output for mobile defines (*e.g.*, transcoding the output into XML for voice browsing).

WOOD employs external information components to extract and process useful content from many different sites. Users of WOOD manually select which components to display and how to lay out the resulting content on the screen. MONTAGE improves on these approaches in two ways. First, MONTAGE automatically selects the content to display, freeing the user from the need to manually maintain the personalized page. Second, MONTAGE embeds web clippings, an approach that we believe captures a more intuitive and robust approach to viewing distal content than filtering and processing content based on the HTML markup of the page.

Newsblaster [12] is a system for clustering news articles from multiple sources and automatically authoring summaries of the stories. Newsblaster's multi-document summarization could be used with MONTAGE to author more descriptive labels for the content panes in the montage. Moreover, with automatic summarization, MONTAGE would no longer need a topic classifier, and could instead cluster content and links based on usage or keywords, and label the clusters based on the automatic summary.

Also similar to MONTAGE is work on automatically building bookmark lists. PowerBookmarks [63] automatically builds a Yahoo!-style web directory of the pages each user visits, selecting which pages to include by how often the user visits them and by their link structure. The Bookmark Organizer [66] is a semi-automated system that maintains a hierarchical organization of a user's bookmarks while letting the user keep control of certain aspects (such as "freezing" nodes in the hierarchy to prevent them from being changed). These systems reduce the effort required to maintain the bookmark lists, but they do not address all the drawbacks of such lists. PowerBookmarks and the Bookmark Organizer are insensitive to the user's browsing context, and may require substantial user effort to find the target link (navigating a hierarchical menu structure or drilling down through a web directory). In contrast, MONTAGE's user model leverages the content of pages viewed and the frequency of links followed to automatically select high-quality bookmarks, and to display the most appropriate bookmarks for the user's current context. MONTAGE also displays both embedded content and links in a single page, requiring no scrolling and at most one link to follow.

Building a web montage is related to the more general interest in building web-based document collections. Systems such as Hunter Gatherer [92] assist users in building collections of web content, either web pages or within-the-page blocks of content. In contrast to these systems, MONTAGE builds its collection automatically and dynamically, but is effectively "hard-wired" to only one collection (the "material for a start page" collection). The techniques in MONTAGE and collection building systems nicely complement each other, and it would be interesting to apply the user-assisted assembly techniques to MONTAGE. For example, both MONTAGE and the user could add components (links and embedded content) to the start page collection, and MONTAGE would automatically group and display the components to the best of its ability. The user could assist MONTAGE with components that are incorrectly displayed or that are added manually. The participants in our study agreed that this type of mixed-initiative system would likely be their favorite.

8.3 Improving mobile web content

A number of companies [10, 89, 56, 37] have taken the first commercial step to bring web content to wireless devices: syntactic translation. Syntactic translation re-codes the web content in a rote manner, usually tag-for-tag or following some predefined templates or rules. This method enjoys some success, particularly for mobile clients that have at least some graphical display (*e.g.*, a Palm Connected Organizer, but not text-only pagers). However, this approach essentially produces a scaled down version of the original web site: all the original content and structure designed for the desktop, broadband visitor, but in miniature form. The mobile visitor must wade through a morass of possibly irrelevant links and content to find the single gem of information for which he or she is looking; browsing such a site with a small screen and a low-bandwidth network connection only exacerbates this problem. Syntactic translation is not a flawed approach—quite to the contrary, it is a necessary component of a successful mobile web site. What it lacks is an awareness of the particular needs of each visitor—syntactic translation simply perpetuates the "one-size-fits-all" philosophy from the original web site.

Two systems similar to our PROTEUS work are Digestor [13] and Pythia [43]. Neither system personalizes web content, but both concentrate simply on transforming content for display on a

small screen with limited network bandwidth. Digestor uses a steepest-descent search similar to ours, to find an optimal page. However, Digestor rates the quality of a web page not on the visitor's expected utility, but only on how much screen space the page occupies (smaller pages have higher quality). This quality metric has two significant weaknesses. First, different elements on a page will have different value to the visitor, and should not be measured simply by how large they are. Second, this metric encourages the system to create degenerate pages—a blank web page receives the highest quality value. Both these weaknesses are addressed by a richer visitor model, such as we define for PROTEUS. In contrast to Digestor and PROTEUS, Pythia does not use search, and instead performs a *distillation* of every image on the requested web page to reduce the screen size and file transmit cost of the content. Pythia allows the visitor to subsequently select images to restore to a larger size in a refinement step. Pythia's approach is well suited to improving how images are transmitted and viewed on mobile devices, but is unable to improve the textual content or navigational structure in a web page. PROTEUS addresses both of these issues and, in fact, would be beneficially complemented by Pythia's approach.

Several other systems also seek to improve the wireless web experience. The Daily Learner [14, 15] is an agent that learns a Palm VII user's preference for news content, by monitoring exactly which stories the user requests from both the Palm device and his or her desktop computer. Based on this feedback, the Daily Learner builds a model of the textual content the user prefers, and automatically builds a page of links to other news stories with similar content. This model is similar to that used by PROTEUS to calculate intrinsic utility, with the exception that The Daily Learner's TFIDF weighting is done over a much larger corpus of documents. It would be interesting to incorporate their content model into PROTEUS to predict intrinsic utility with higher confidence.

The goal of m-Links [91] is to enable access to web content from cellular phones, on which only two to four lines of text are viewable. With such limited display space, transcoding existing content is not feasible—reducing arbitrary content into such a small space is nearly impossible. Instead, m-Links automatically identifies objects of interest on web pages and builds its own hierarchical menu for accessing these objects. m-Links also associates with each object a set of services available, for example, printing PDF files, reading e-mail, or listening to the text of a document with text-to-speech synthesis. The m-Links approach can be viewed as a pre-processing step in our search framework: each web page is first converted into an m-Links structure, and then we could perform search through the space of arrangements of content, potentially moving web data from one page to another or moving important items higher in lists shown to the user. Successfully personalizing web content on a cellular phone could have great impact in mobile web usability, but also faces a significant challenge: training data is very sparse. The Daily Learner combines training data from mobile and desktop devices, and personalization for cellular phone web browsing must do the same.

The Stanford Power Browser [20, 21, 22] is a client-side approach that seeks to improve the viewing of web pages. Most significantly, the Power Browser enables "accordion summarization:" sections of text of a web page are first displayed "rolled-up" and replaced with a single line of text. The user can easily expand each rolled-up section by clicking on the text to view progressively more detail. The advantage of summarization is to allow a user to quickly view the whole content of a document, at a high-level, and then drill down into only those parts of interest (and avoid scrolling through irrelevant content). The Power Browser complements our PROTEUS approach by improving the interaction with each page the visitor views. An interesting item of future work would be to evaluate the improvement in navigation each of these features offers, and how they interact with one another.

8.4 Predicting navigation

Predicting web navigation well has impact beyond web personalization. Much of the earlier work on predicting navigation was done with an eye toward servers predicting future requests and pre-sending documents to clients. When the server guesses correctly, the latency of the next request is greatly decreased; and if the server guesses incorrectly, the client simply resets the connection and requests the intended next document.

The two leading approaches for predicting navigation are Markov models and n-gram models. Markov and n-gram models are very closely related, and we distinguish them by whether they predict *which* URL will be requested next (these are n-gram models), or if they compute a *probability* of the next request (these are Markov models).

An n-gram is a sequence of n web requests, and the n-gram model records how often each sequence of n requests was made in the training data. To predict a future request, an agent looks for an n-gram that matches the suffix of the user's recent browsing trail. Previous work [83, 90, 97, 35] has explored how to choose which n-gram and how to reduce the model storage requirement (given that the size of the model increases exponentially with n). A common technique is to learn n-grams for $n = 1, 2, 3, \ldots$ and select the longest n-gram that matches the current testing trail [83]. This process can be viewed as a form of shrinkage: the length of the n-gram is a measure of it specificity, and one can combine multiple applicable n-grams by shrinking estimates with larger values of n toward more general estimates with smaller values of n. Deshpande and Karypis [35] propose using pruning techniques to reduce the model size and improve predictive accuracy. A drawback of the n-gram approach is that it requires large volumes of data to train reliably and to ensure adequate coverage of 2-, 3-, and longer n-grams. Another drawback, common to both n-gram and Markov models, is that a model that treats URLs as unrelated, symbolic entities cannot generalize to previously unvisited pages. Also, n-gram models often simply predict *which* page will be requested next, and do not estimate the probability of that next request. The models our MINPATH algorithm requires must produce a visit probability for each link for a given page in the site.

In Chapter 5 we discussed how Markov models could be used to predict web navigation. The WebCANVAS system [24] also uses Markov models (mixtures of Markov models, in fact) to model navigation, for the purpose of visualizing the resulting groups of users. Sarukkai [88] builds first-order Markov models and Zukerman *et al.* [99] build first- and second-order models. Sarukkai suggests several applications enabled by predicting navigation, including server pre-sending, client-side highlighting of the user's next link, and automatic tour generation (*i.e.*, automatically finding high-likelihood trails in the site and showing these to visitors). Experiments from both works show that Markov models can predict the next page request relatively well; Sarukkai, for example, reports a 50% predictive accuracy (*i.e.*, 50% of the time, the page with the highest predicted probability was the next page requested). Zukerman *et al.*'s second-order model performed better than their first-order model, but an even greater gain was achieved by simply conditioning the model on the set of available links on the client's referring document. MINPATH also conditions its probability in this way, and for the same reason—the user is far more likely to follow a link from the current page than jump to a random page elsewhere in the site.

Other methods of web prediction are possible. Mladenić [72] proposed building naïve Bayes or k-nearest neighbor models to predict which link the user will follow given the text of the current page, the possible destinations, and the pages previously viewed by the visitor. Our work with relational Markov models differs in that we produce a probability distribution over the out-links, instead of selecting only the single most likely one.

8.4.1 Web recommender systems

Mobasher *et al.* [73] propose building frequent itemsets of URLs viewed, and then suggesting pages in these itemsets not yet seen. A partial user session is compared against each itemset to derive a matching score, and each URL in the itemsets is rated by the set's matching score, the URL's relevance to the itemset, and the minimum click-distance between the user's current session and the URL. The highest-rated URLs not visited in the current session are the recommendations. Mobasher *et al.* present anecdotal results showing that useful recommendations appear within the top 15 links, but do not evaluate their approach on a wide range of real-world user sessions. In addition, this method can serve only as a heuristic for page recommendation—unlike MinPath, this approach does not compute the probability of visiting a page, nor an expected savings for links (their distance formula is based on a minimum traversal, which may not be a reasonable assumption).

In later work, Mobasher *et al.* [74] combine their previous usage-based page clustering approach with a content-based approach: producing clusters of pages based on their textual features. They present anecdotal evidence that each clustering scheme can find recommendations that the other did not, but they do not demonstrate on real-world data which recommendations are, in fact, useful for web visitors.

Yan *et al.* [96] model visitors as vectors in URL-space—an n-dimensional space with a separate dimension for each page at a site—and cluster them using traditional vector clustering techniques (in their case, the leader algorithm). They explore various parameter values for leader, including minimum cluster (membership) size and maximum distance between any page and the median of the cluster, but leave for future work producing recommendations given these clusters.

Letizia [65] is a client-side agent that browses the web in tandem with the visitor. Based on the visitor's actions (*e.g.*, which links the visitor followed, whether the visitor records a page in a bookmarks file, *etc.*), Letizia estimates the visitor's interest in as-yet-unseen pages. Unlike MinPath, which resides on a web server, Letizia is constrained to the resources on the web visitor's browsing device, and is thus not well suited to a wireless environment. In addition, Letizia cannot leverage

the past experiences of *other* visitors to the same site—Letizia knows about the actions of only its visitor.

WebWatcher [57], Ariadne [58], and adaptive web site agents [78] are examples of web tour guides, agents that help visitors browse a site by suggesting which link each visitor should view next. With the assistance of a tour guide, visitors can follow trails frequently viewed by others and avoid becoming lost. However, tour guides assume that every page along the trail is important, and typically are limited to only suggesting which link on a page to follow next (as opposed to creating shortcuts between pages).

SurfLen [48] and PageGather [82] improve navigation by building lists of links to destinations sharing a common topic. SurfLen mines for association rules, performing a form of "market basket" analysis [1]. PageGather builds a co-occurrence matrix of all pairs of pages visited and finds clusters of pages frequently viewed in the same session. These algorithms suggest the top m pages that are most likely to co-occur with the visitor's current session, either by presenting a list of links in a window adjacent to the browser (SurfLen) or by constructing a new index page containing the links (PageGather). However, both of these systems assume the visitor can easily navigate a lengthy list of shortcuts, and thus provide perhaps dozens of suggested links. MinPath improves on these algorithms by factoring in the relative benefit of each shortcut, and suggesting only the few best links specific to each page request.

SurfLen [48] and PageGather [82] suggest pages to visit based on page requests co-occurrent in past sessions. SurfLen mines for association rules, performing a form of "market basket" analysis [1]. PageGather builds a co-occurrence matrix of all pairs of pages visited and finds clusters of pages frequently viewed in the same session. These algorithms suggest the top m pages that are most likely to co-occur with the visitor's current session, either by presenting a list of links (SurfLen) or by constructing a new index page containing the links (PageGather). However, both of these systems assume the visitor can easily navigate a lengthy list of shortcuts, and thus provide perhaps dozens of suggested links. MinPath improves on these algorithms by factoring in the relative benefit of each shortcut, and suggesting only the few best links specific to each page request.

8.5 *Adding conceptual knowledge to web pages*

The goal of the WebKB project [31] is to populate a relational knowledge base given the textual content and hyperlink connectivity of web pages. This goal is different from that of RMMs—RMMs presume the existence of a relational model and predict transitions using the model. However, it would be interesting to apply the WebKB learning approach to populate a model *describing a web site* and use RMMs to predict navigation in that model. Although most e-commerce sites are dynamically generated from database queries, many other large sites (*e.g.*, corporate intranets or academic institution web sites) exist only as large collections of static web pages. The WebKB approach could prove fruitful for producing the relational information RMMs need for such static sites.

Finally, much research has been done in recent years on classifying web pages (*e.g.*, Pazzani *et al.* [77], McCallum, *et al.* [69]). Any web page classifier that yields class probabilities can in principle be used in place of RMMs for adaptive web navigation. However, many of these classifiers are based on viewing web pages as bags of words, and are unable to take advantage of the relational structure of the site. Incorporating bag-of-words information into RMMs may be useful and is a direction for future work.

Chapter 9

FUTURE DIRECTIONS

Our work on web personalizers demonstrates that automated techniques can improve the web experience. We outline below how one could improve our existing systems, extend their capabilities, and carry out further research in this general area.

9.1 *The future of* PROTEUS

We noted in Section 4.5 that a weakness of content elision in PROTEUS was that links to elided content were indistinguishable from links to full-fledged content at the site. A possible improvement to this adaptation would be to annotate elided content links with a special symbol, such as "[&]", that would indicate how the original page had been modified.

It would also be valuable to perform a more exhaustive study to determine the cost of user actions (γ_s, γ_l, and $P(\mathsf{scroll})$ in Equation 4.1). In our work we have assumed that the user is equally likely to scroll to each subsequent screen on a page, but intuition also suggests that perhaps the probability of scrolling *at all* (*i.e.*, to the second screen) is much lower, and that scrolling to subsequent screens is (conditionally) greater. That is, a visitor may scroll to the second screen with probability 0.03, but will subsequently visit each following screen with probability 0.40. A more rigorous user study to measure these parameters could greatly improve the quality of the evaluation metric. Another direction would be to apply work from cognitive science to more closely model how humans understand online information, how they interact with it, *etc.* Drawing from this foundation has led to impressive improvements in information visualization (for example, the LineDrive system [2] for visualizing driving directions).

This thesis examined only three adaptations in PROTEUS—content elision, swapping content, and adding shortcuts—but many others are possible and could prove useful. For example, instead of replacing images with their HTML "alt" text, as PROTEUS does presently, we could scale or crop images to reduce their sizes. This approach, used in Pythia [43], would retain more information from the image but not overly burden mobile web clients. Another adaptation could help users interact with web forms, by predicting the user's input and automatically filling in fields.

We implemented PROTEUS as a proxy to enable personalization of all web sites, requiring (and expecting) no assistance from those sites' maintainers. However, some web sites may wish to place constraints on how content should be personalized, and are willing to provide guidance to PROTEUS to ensure these constraints are met. For example, a site designer might wish to specify that certain elements on the page are immutable (*e.g.*, copyright notice and link to privacy policy) but others can be elided, rearranged, or added to. Perkowitz and Etzioni proposed the idea of adaptive HTML, or A-HTML [81], an extension to HTML in which designers could express these constraints in the body of the documents themselves. An interesting line of future work would be to formalize A-HTML, restrict PROTEUS to obey these constraints, and explore how constraining the personalization could perhaps reduce the disorientation that users experience when they browse from mobile devices.

9.2 The future of MINPATH

It is clear that the quality of a shortcut link depends on the quality of the phrase that anchors it on the originating page. A visitor seeking information about colloquia at a university is more likely to follow a link labeled "Today's colloquia" than one labeled "Click here" or "Shortcut." One way to label shortcuts is by the `<title>` information from the destination page. This approach is generally how search engines label search results, but the `<title>` may be missing, incorrect or vague (*e.g.*, "Homework" for a page at a university), or overly verbose (*e.g.*, the concatenation of each page topic along the path from the main entry page to each destination). An alternative approach would be to generate perhaps several dozen candidate anchor phrases, and use a machine learning system to predict which would be the best anchor. The candidates could be word phrases from the destination page, or a combination of words on the destination and (not) on the originating page, or even phrases anchoring links to the same destination from other pages on the Web. The candidates could be evaluated by a model trained using human-tagged anchor phrases and features such as the source of the phrase, the TFIDF [87] value of the phrase compared to the source and destination pages, *etc.*

The results comparing MINPATH to the memory-based approach (Figure 5.7) show, in part, that MINPATH is not making as effective use of a set of shortcuts as possible. We discussed in Section 5.3 how MINPATH could be extended to condition expected savings on the set of shortcuts selected, and evaluating this approach is an interesting next step. Another fruitful avenue to explore is employing higher-quality models. Combining MINPATH with relational Markov models improved MINPATH's performance significantly when training data is sparse. Other possibilities include higher-order Markov models, models that condition on content of pages (*e.g.*, conditioning on the page topic), and models that condition on broader browsing context (*e.g.*, the context used by MONTAGE). It would also be useful to explore other means of assigning visitors to clusters (*i.e.*, mixture model components). One possibility would be to allow the user to optionally choose which archetypal behavior best fits his or her browsing patterns.

Finally, a valuable extension of this work would be a fielded user study, combining MINPATH with relational Markov models and deploying the system on one or more web sites. A fielded study improves on our earlier evaluation of MINPATH by avoiding the simplifying assumptions we made regarding user navigation. The MINPATH models would likely be re-learned often, to account for the influence that the automatically-created shortcuts may have on the navigation. The subtlety of the experiment is in measuring the success of a shortcut. For example, if a user follows a shortcut, and then navigates from that destination, then the shortcut was probably beneficial to the visitor. However, if a user follows a shortcut and immediately follows another link from the original page, then the shortcut may not have been useful, or, it *did* satisfy the user's then-current goal, and the user is now exploring for a new goal. One way to discern between these cases is to collect exit surveys from a sample of users, asking whether they found the content they were looking for, and then extrapolate these results to the larger population.

9.3 The future of relational Markov models

Relational Markov models are a fertile ground for continued research, both in directly extending RMMs and in applying the relational ideas to other probabilistic models. An immediate area to pursue is in further developing the RMM-PET approach, by building PETs to predict finer partitions among the destination abstractions. An interesting evolution of RMMs is to relational hidden

Markov models, where both the states and the observations are described by typed relations and shrinkage is carried out over their respective abstractions. Another direction is incorporating a model cluster identity into the transition probability, such as the identity of a cluster of visitors at a web site, and shrinking between many models learned for different sizes and scope of user cluster (such as a single user, a cluster of similar users, and the set of all users at the site). A third path of research is to apply RMMs to other domains, such as mobile robot localization or speech recognition.

Chiefly important to relational Markov models is the relational information: if the relational structure is uninformative, then RMMs are no more useful than PMMs. In our applications the relational structure was readily available, but a direction of future work would explore how the structure can be learned, and the trade-off between quality of relational structure and predictive accuracy of the RMM. We discussed earlier how a machine learning approach, such as used in the WebKB project [31], could learn where pages belong in a relational structure schema, given the schema and a set of training instances. Another area of future work is to learn the structural schema itself (*i.e.*, the relations and the abstractions of member states). The idea in TUBA [55], of clustering states based on their utility functions, could similarly be applied here by clustering states based on their transition probability distributions. A hierarchical clustering could produce abstractions of states, but learning a non-trivial lattice of abstractions is more subtle.

9.4 *The future of* MONTAGE

The MONTAGE user study highlighted several aspects of the system for improvement. One is the speed of loading the montage; the current system can require as much as 30 seconds to load a montage. The primary bottleneck lies in how the MONTAGE proxy is architected—a single unresponsive server can stall the proxy and cause the entire montage to load slowly. Instead of fetching montage content only on-demand, MONTAGE should pre-fetch content at regular intervals (or, at intervals suggested by the user model). Alternatively, MONTAGE could finesse the main montage load time issue altogether by using a text- and links-only main montage, with graphical embedded-content montages only for topic-specific views.

Although MONTAGE presently collects user models at a central proxy, each montage is based

on only one visitor's user model. A useful extension would enable collaborative filtering between groups of similarly behaving users. For example, a group of machine learning researchers may view several common sites but only one of the group has discovered a new machine learning related resource (*e.g.*, a new conference web). A collaborative montage could display the new resource for all members of the group, based on the information that the new site is machine learning related, matching the interests of all group members. Of course, as with all collaborative filtering work, one must carefully guard the privacy of members.

Another extension would enable users to more directly control the content displayed on the montage and the system trade-offs in the user model and in suggesting content. For example, MONTAGE users could optionally suggest content and links to include or exclude from the montage, and MONTAGE would display the manually- and automatically-selected components to the best of its ability. Users could also place constraints on how the content is displayed, for example, requiring that some particular web clipping always appears in the upper-right corner. To influence the underlying user model, the user could, for example, adjust the relative weights given to cost of navigation and value of information, thus indirectly controlling how frequently content is embedded or simply linked from the main montage. The user could also give MONTAGE feedback about its selection of content, either by rating individual suggestions, or by manually adding or removing items from a montage.

A final extension is enabling MONTAGE to dynamically select the level of granularity to model content topics. A machine learning researcher will likely want more fine resolution about web sites than "Computers & Internet,", but will not need nor want finer granularity uniformly among all topics. MONTAGE could place each page in a finely-structured topic hierarchy, and then display only those sub-topics receiving at least some threshold proportion of pages.

9.5 *New directions*

9.5.1 *Mixed-initiative user modeling*

A predominant weakness of automated personalization techniques is that automated user modeling builds only a blurry image of the user. For example, the PROTEUS model based on words occurring in pages viewed gives only a general indication of the words that the visitor apparently tends to prefer. Of course, these words may not be indicative of the content the user is actually viewing (for example, the text of advertisements may differ greatly from the text central to the page).

As an alternative, several systems allow (in fact, require) that users state their interests up front. The clear advantage is that this communication is clear—the personalizer need not guess this aspect of the model. Unfortunately, few users are willing to expend this effort initially. An interesting line of research would be to combine these approaches: automatically build an approximate model, and allow the user to augment and correct any aspects of it he or she chooses. A twist on this approach uses active learning [29]: the personalizer proposes questions to the visitor that would maximally improve the quality of the model with minimal effort. The DIAManD interface [94] takes this general approach.

9.5.2 Personalizing non-web interfaces

Personalizing web sites is a stepping stone to to personalizing more general-purpose interfaces, in which users can alter the state of the underlying system (as opposed to simply navigating through it). A key challenge in personalizing such an interface lies in modeling *how* the interface is used, and thus what adaptations are allowed. A logical extension of our model of web interactions would be to model applications as state-based processes where the state encodes the configuration of the entire application—the dialogs and windows displayed, the content of documents, *etc.*—and the transitions are actions a user can take. Although the state space is potentially infinite, effective inference is possible using relational Markov models.

9.5.3 Adapting declarative interfaces

The web models our personalizers have used are based largely on the graph nature of the pages and links. However, a competing approach is to model sites at a higher, abstract level (as described in Chapter 2), separating the site into data, structure, and presentational elements [40, 7]. Each of these aspects is specified separately be the web site designer, and the data and structure are often selected *declaratively*, in the form of queries over a database. As separate elements, the web site personalizer can transform each aspect of the site independently for both a computational cost savings and for an expected utility gain. For example, when the data on the site is separated from its presentation,

the personalizer can very precisely tune what content the visitor will see—largely independently of how the content is currently presented. Moreover, the declarative specification for data and structure allows the personalizer to more easily reason about the site (*i.e.*, the agent need not infer the structure by analyzing HTML pages).

Chapter 10

CONCLUSIONS

Most web sites today are designed with a one-size-fits-all philosophy: a single view of the content and navigational structure for every visitor under all conditions. But one size frequently does not fit all: visitors browse from different devices, look for different content, and have different styles of navigation. In order to maximize the value of a web interaction, we propose to personalize the experience for each user.

We view web personalization as a two-step process, of first modeling the users and then adapting the content to best meet the needs of each user (or group of related users). In this thesis we have described this process in detail (Chapters 1, 2, and 3), built two personalizers based on this architecture (PROTEUS, in Chapter 4 and MONTAGE, in Chapter 7), and developed the MINPATH algorithm (Chapter 5) for use in PROTEUS.

This thesis addressed two primary questions. First: *How can automated personalization improve the web experience?* To answer this question, we built and fielded two web personalizers, PROTEUS and MONTAGE, that automatically adapted the web experience for mobile and desktop users, respectively. Data from our user studies suggest that the personalized views of content did help web users achieve their goals, but personalizers must take care to not inconvenience users (*e.g.*, hiding visual cues on the page or shifting the placement on the screen of often-followed links). MONTAGE was perhaps more successful than PROTEUS because MONTAGE only created new pages and never modified existing content (thus never giving itself an opportunity to disorient users on existing web pages). But, for mobile web users, adapting existing content is a necessity. The challenge for future mobile content personalization is identifying which visual elements users rely on for navigation, which elements are useful content for users, and which elements are superfluous and should be removed.

The second question we addressed was: *How can automated personalization scale to adapt content for each visitor at each page impression?* This question embeds two parts—scaling to

real-time performance, and scaling to individual users—and we address both. On the run-time side, the original PROTEUS implementation required on the order of minutes to adapt a page for a mobile visitor, and this delay is too long. We developed our MINPATH algorithm (Chapter 5), in part, to directly address this concern, by finding shortcut links efficiently. Learning the models underlying MINPATH (*e.g.*, traditional or relational Markov models) requires a reasonable amount of time for batch processing, on the order of minutes to learn from tens of thousands of past sequences on thousands of web pages. The on-line performance task—finding shortcuts for a page given a learned model and current trail prefix—requires less than one second for reasonable results, and this number can be reduced by adjusting the recursion bounds in the MINPATH algorithm (depth bound and probability threshold in Table 5.1).

The other part of the second question is scaling to individual visitors and individual web pages. PROTEUS personalizes content for each visitor, but makes inefficient use of the data available, by ignoring all but the data for a single visitor at a time. MINPATH makes much better use of the training data available, by clustering users together and learning models for each cluster. In order to scale to individual web pages, where training data may be sparse or non-existent, we developed relational Markov models (Chapter 6), which learn models of navigation at varying levels of abstraction, and can rely on higher levels of abstraction when training data is sparse at the more specific levels.

10.1 Contributions

This thesis has made the following contributions:

1. **Precise statement of web personalization problem.** Chapter 2 outlines the space of web personalization systems and provides precise definitions of a web site and a web visitor, and how web navigation is modeled. Chapter 2 uses these definitions to state the web personalization problem: given a web site, log data describing past behavior, and a current user, adapt the page requested by the current user to maximize expected utility.

2. **Framework for personalization as search.** Drawing inspiration from Perkowitz and Etzioni's earlier work on adaptive web sites [80, 82, 79], we develop a richer model of a web site and a more sophisticated model of web site utility. The model of a web site includes the

set of relations between web pages (or web content objects, in general), allowing a personalizer to infer generalizations from these relations. The web site utility model calculates the *expected utility* of the site for a given visitor, and is based on the value of the content and the probability the visitor will view each page.

3. **Efficient approach to finding shortcut links.** To personalize content in real-time, we developed the MINPATH algorithm to find shortcut links efficiently (Chapter 5). MINPATH leverages the past behavior of a large population of visitors to the site (as opposed to a single user) and learns probabilistic models of web navigation, including Markov models and hybrid Markov models and clusters of models. MINPATH is capable of finding valuable shortcuts fast enough for personalizing content on-the-fly for each visitor.

4. **Development of relational Markov models.** In Chapter 6, we developed relational Markov models, described where they are applicable, and discussed how they relate to other probabilistic models (Section 6.5). We showed in experiments that relational Markov models outperform traditional Markov models for predicting web navigation when data is sparse, and that they can improve MINPATH's performance by 50% to 80%. The techniques underlying relational Markov models—endowing states with relational information and performing shrinkage based on lattices of abstraction—are applicable to many other models as well.

5. **Evaluation of several systems.** Throughout this thesis, we have presented experimental results evaluating our approach, including two user studies (Sections 4.5 and 7.5). The evidence supports our innovations and points to where future efforts should be concentrated.

10.2 Conclusion

The number and complexity of user interfaces with which we interact continues to increase. Intelligence in these interfaces, and personalization in particular, is a necessity in order that we may manage this growth. Personalizing the web experience is a step in this direction, as the web is the preeminent interface to online information systems. The techniques described in this thesis improve users' ability to access information, and aim to serve as groundwork for future personalized interface

and intelligent user interface efforts.

BIBLIOGRAPHY

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of ACM SIGMOD Conference on Management of Data*, 1993.

[2] M. Agrawala and C. Stolte. Rendering effective route maps: Improving usability through generalization. In *Proceedings of the 28th International Conference on Computer Graphics and Interactive Techniques*, 2001.

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

[4] C. R. Anderson, P. Domingos, and D. S. Weld. Personalizing web sites for mobile users. In *Proceedings of the Tenth International World Wide Web Conference*, 2001.

[5] C. R. Anderson, P. Domingos, and D. S. Weld. Relational Markov models and their application to adaptive web navigation. In *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining*, 2002.

[6] C. R. Anderson and E. Horvitz. Web Montage: A dynamic, personalized start page. In *Proceedings of the Eleventh International World Wide Web Conference*, 2002.

[7] C. R. Anderson, A. Y. Levy, and D. S. Weld. Declarative web-site management with TIRAMISU. In *Proceedings of the International Workshop on The Web and Databases (WebDB)*, 1999.

[8] V. Anupam, Y. Breitbart, J. Freire, and B. Kumar. Personalizing the web using site descriptions. In *Proceedings of the 10th International Workshop on Database and Expert Systems Applications (DEXA)*, 1999.

[9] V. Anupam, J. Freire, B. Kumar, and D. Lieuwen. Automating web navigation with the WebVCR. In *Proceedings of the Ninth International World Wide Web Conference*, 2000.

[10] AvantGo, Inc. *AvantGo*. `http://www.avantgo.com/`.

[11] R. Barrett, P. P. Maglio, and D. C. Kellem. How to personalize the web. In *Proceedings of ACM CHI 1997 Conference on Human Factors in Computing Systems*, 1997.

[12] R. Barzilay, K. McKeown, and M. Elhadad. Information fusion in the context of multi-document summarization. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, 1999.

[13] T. W. Bickmore and B. N. Schilit. Digestor: Device-independent access to the world wide web. In *Proceedings of the Sixth International World Wide Web Conference*, 1997.

[14] D. Billsus, C. A. Brunk, C. Evans, B. Gladish, and M. Pazzani. Adaptive interfaces for ubiquitous web access. *Communications of the ACM*, 45(5):34–38, May 2002.

[15] D. Billsus, M. J. Pazzani, and J. Chen. A learning agent for wireless news access. In *Proceedings of the 2000 Conference on Intelligent User Interfaces*, 2000.

[16] P. Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6(2-3):87–129, 1996.

[17] P. Brusilovsky, A. Kobsa, and J. Vassileva. *Adaptive Hypertext and Hypermedia*. Kluwer Academic Publishers, 1998.

[18] P. Brusilovsky and M. T. Maybury. From adaptive hypermedia to the adaptive web. *Communications of the ACM*, 45(5):31–33, May 2002.

[19] Peter Brusilovsky. Adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 11:87–110, 2001.

[20] O. Buyukkokten, H. Garcia-Molina, and A. Paepcke. Accordian summarization for end-game browsing on PDAs and cellular phones. In *Proceedings of ACM CHI 2001 Conference on Human Factors in Computing Systems*, 2001.

[21] O. Buyukkokten, H. Garcia-Molina, and A. Paepcke. Seeing the whole in parts: Text summarization for Web browsing on handheld devices. In *Proceedings of the Tenth International World Wide Web Conference*, 2001.

[22] O. Buyukkokten, H. Garcia-Molina, A. Paepcke, and T. Winograd. Power Browser: Efficient Web browsing for PDAs. In *Proceedings of ACM CHI 2000 Conference on Human Factors in Computing Systems*, 2000.

[23] Cheukyin C. WOOD – web-based object-oriented desktop. In *Poster Proceedings of the Tenth International World Wide Web Conference*, 2001.

[24] I. V. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White. Visualization of navigation patterns on a web site using model based clustering. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining*, 2000.

[25] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghavan. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *The VLDB Journal*, 7(3):163–178, 1998.

[26] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. AutoClass: A Bayesian classification system. In *Proceedings of the Fifth International Conference on Machine Learning*, 1988.

[27] C. Chekuri, M. H. Goldwasser, P. Raghavan, and E. Upfal. Web search using automatic classification. In *Poster Proceedings of the Sixth International World Wide Web Conference*, 1997.

[28] H. Chen and S. T. Dumais. Bringing order to the web: Automatically categorizing search results. In *Proceedings of ACM CHI 2000 Conference on Human Factors in Computing Systems*, 2000.

[29] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[30] R. Coley, B. Mobasher, and J. Srivasta. Data preparation for mining world wide Web browsing patterns. *Journal of Knowledge and Information Systems*, 1(1), 1999.

[31] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to construct knowledge bases from the World Wide Web. *Artificial Intelligence Journal*, 118(1–2):69–113, 2000.

[32] T. Dean and K. Kanazawa. Probabilistic Temporal Reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988.

[33] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5:142–150, 1989.

[34] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[35] M. Deschpande and G. Karypis. Selective Markov models for predicting web-page accesses. In *First SIAM International Conference on Data Mining*, 2001.

[36] T. G. Dietterich. State abstraction in MAXQ hierarchical reinforcement learning. In S. A. Solla, T. K. Leen, and K.-R. Muller, editors, *Advances in Neural Information Processing Systems 12*, pages 994–1000. MIT Press, Cambridge, MA, 2000.

[37] Digital Paths LLC. *DPWeb*. `http://www.digitalpaths.com/prodserv/dpwebdx.htm`.

[38] P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29:103–130, 1997.

[39] S. Dzeroski and L. de Raedt. Relational reinforcement learning. In *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.

[40] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. Catching the boat with Strudel: Experiences with a web-site management system. In *Proceedings of ACM SIGMOD Conference on Management of Data*, 1998.

[41] S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden Markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, 1998.

[42] D. Flanagan. *JavaScript: The Definitive Guide*. O'Reilly & Associates, 2001.

[43] A. Fox and E. Brewer. Reducing WWW latency and bandwidth requirements by real-time distillation. In *Proceedings of the Fifth International World Wide Web Conference*, 1996.

[44] J. Freire and B. Kumar. Web services and information delivery for diverse environments. In *Proceedings of the VLDB Workshop on Technologies for E-Services*, 2000.

[45] J. Freire, B. Kumar, and D. Lieuwen. WebViews: Accessing personalized Web content and services. In *Proceedings of the Tenth International World Wide Web Conference*, 2001.

[46] D. Freitag and A. K. McCallum. Information extraction with HMMs and shrinkage. In *AAAI'99 Workshop on Machine Learning for Information Extraction*, 1999.

[47] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*.

[48] X. Fu, J. Budzik, and K. J. Hammond. Mining navigation history for recommendation. In *Proceedings of the 2000 Conference on Intelligent User Interfaces*, 2000.

[49] L. Getoor, D. Koller, and N. Friedman. From instances to classes in probabilistic relational models. In *Proceedings of the ICML-2000 Workshop on Attribute-Value and Relational Learning*, 2000.

[50] Z. Ghahramani and M.I. Jordan. Factorial hidden Markov models. *Machine Learning*, 29:245–273, 1997.

[51] Steven Glassman. A caching relay for the world wide web. *Computer Networks and ISDN Systems*, 27(2):165–173, 1994.

[52] I. J. Good. *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. MIT Press, Cambridge, MA, 1965.

[53] Google, Inc. *Google*. `http://www.google.com/`.

[54] D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[55] E. Horvitz and A. Klein. Utility-based abstraction and categorization. In *Proceedings of Ninth Conference on Uncertainty in Artificial Intelligence*, 1993.

[56] ILINX, Inc. *Palmscape 3.0*. `http://www.ilinx.co.jp/en/products/ps.html`.

[57] 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*, 1997.

[58] J. Jühne, A. T. Jensen, and K. Grønbæk. Ariadne: a Java-based guided tour system for the World Wide Web. In *Proceedings of the Seventh International World Wide Web Conference*, 1998.

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

[60] A. Kobsa. Generic user modeling systems. *User Modeling and User-Adapted Interaction*, 11:49–63, 2001.

[61] R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng. KDD-Cup 2000 organizers' report: Peeling the onion. *SIGKDD Explorations*, 2(2):86–98, 2000. http://www.ecn.purdue.edu/KDDCUP.

[62] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.

[63] W. Li, Q. Vu, D. Agrawal, Y. Hara, and H. Takano. PowerBookmarks: A system for personalizable web information organization, sharing, and management. In *Proceedings of the Eighth International World Wide Web Conference*, 1999.

[64] H. W. Lie and B. Bos. *Cascading Style Sheets: Designing for the Web*. Addison-Wesley, 1999.

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

[66] Y. S. Maarek and I. Z. Ben-Shaul. Automatically organizing bookmarks per contents. In *Proceedings of the Fifth International World Wide Web Conference*, 1996.

[67] P. P. Maglio and R. Barrett. Intermediaries personalize information streams. *Communications of the ACM*, 43(8), 2000.

[68] I. Mani and M. Maybury. *Advances in Automatic Text Summarization*. MIT Press, 1999.

[69] A. McCallum, R. Rosenfeld, T. Mitchel, and A. Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.

[70] Microsoft Corporation. *My MSN*. `http://my.msn.com/`.

[71] D. Mladenic. Turning Yahoo into an automatic web page classifier. In *Proceedings of the 13th European Conference on Artificial Intelligence*, 1998.

[72] D. Mladenić. Machine learning for better web browsing. In *AAAI Spring Symposium on Adaptive User Interfaces*, 2000.

[73] B. Mobasher, R. Cooley, and J. Srivastava. Creating adaptive web sites through usage-based clustering of URLs. In *Proceedings of the 1999 IEEE Knowledge and Data Engineering Exchange Workshop (KDEX'99)*, 1999.

[74] B. Mobasher, H. Dai, T. Luo, Y. Sun, and J. Zhu. Combining web usage and content mining for more effective personalization. In *Proceedings of the International Conference on E-Commerce and Web Technologies (ECWeb)*, 2000.

[75] S. Muggleton. Stochastic logic programs. In L. de Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, Amsterdam, The Netherlands, 1996.

[76] L. Ngo and P. Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147–177, 1997.

[77] M. Pazzani, J. Muramatsu, and D. Billsus. Syskill & Webert: Identifying interesting Web sites. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[78] M. J. Pazzani and D. Billsus. Adaptive web site agents. In *Proceedings of the Third International Conference on Autonomous Agents*, 1999.

[79] M. Perkowitz. *Adaptive Web Sites: Cluster Mining and Conceptual Clustering for Index Page Synthesis*. PhD thesis, University of Washington, Computer Science and Engineering, 2001.

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

[81] M. Perkowitz and O. Etzioni. Adaptive web sites: Automatically learning from user access patterns. In *Proceedings of the Sixth International World Wide Web Conference*, 1997.

[82] M. Perkowitz and O. Etzioni. Towards adaptive web sites: Conceptual framework and case study. *Artificial Intelligence Journal*, 118(1–2), 2000.

[83] J. Pitkow and P. Pirolli. Mining longest repeating subsequences to predict world wide web surfing. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems*, 1999.

[84] F. Provost and P. Domingos. Tree induction for probability-based ranking. *Machine Learning*. To appear.

[85] L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[86] K. Railsback and L. Mitchell. Wireless LANs and internet access free workers from their desks. InfoWorld, December 2000.

[87] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, NY, 1983.

[88] R. R. Sarukkai. Link prediction and path analysis using Markov chains. In *Proceedings of the Ninth International World Wide Web Conference*, 2000.

[89] QUALCOMM, Inc. *Eudora Internet Suite 5.0*. `http://www.eudora.com/internetsuite/eudoraweb.html`.

[90] S. Schechter, M. Krishnan, and M. D. Smith. Using path profiles to predict HTTP requests. In *Proceedings of the Seventh International World Wide Web Conference*, 1998.

[91] B. N. Schilit, J. Trevor, D. Hilbert, and T. K. Koh. m-Links: An infrastructure for very small internet devices. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, 2001.

[92] m. c. schraefel, Y. Zhu, D. Modjeska, D. Wigdor, and S. Zhao. Hunter Gatherer: Interaction support for the creation and management of within-Web-page collections. In *Proceedings of the Eleventh International World Wide Web Conference*, 2002.

[93] P. Smyth, D. Heckerman, and M. I. Jordan. Probabilistic independence networks for hidden Markov probability models. *Neural Computation*, 9:227–269, 1997.

[94] S. A. Wolfman, T. Lau, P. Domingos, and D. S. Weld. Mixed initiative interfaces for learning tasks: SMARTedit talks back. In *Proceedings of the 2001 Conference on Intelligent User Interfaces*, 2001.

[95] Yahoo! Inc. *MyYahoo!* `http://my.yahoo.com/`.

[96] T. Yan, M. Jacobsen, H. Garcia-Molina, and U. Dayal. From user access patters to dynamic hypertext linking. In *Proceedings of the Fifth International World Wide Web Conference*, 1996.

[97] S. Zhong, Q. Yang, and H.-J. Zhang. A prediction system for multimedia pre-fetching on the internet. In *Proceedings of the Eighth ACM Multimedia Conference*, 2000.

[98] I. Zukerman, D. Albrecht, A. Nicholson, and K. Doktor. Trading off granularity against complexity in predictive models for complex domains. In *Proceedings of the Sixth International Pacific Rim Conference on Artificial Intelligence*, 2000.

[99] I. Zukerman, D. W. Albrecht, and A. E. Nicholson. Predicting users' requests on the WWW. In *Proceedings of the 7th International Conference on User Modeling*, 1999.

Appendix A

PROTEUS USER STUDY QUESTIONS

- "What is the top editor's choice for ⟨*device*⟩ at cnet.com?", where ⟨*device*⟩ was one of: a "semipro" digital camera; an ultralight laptop; a Palm OS handheld; a CD-RW; a flat panel display.

- "What is the best price for ⟨*device*⟩ at cnet.com?", where ⟨*device*⟩ was one of: a "semipro" digital camera; an ultralight laptop; a Palm OS handheld; a CD-RW; a flat panel display.

- "What is the current stock value of ⟨*ticker*⟩? Start at finance.yahoo.com", where ⟨*ticker*⟩ is in the set MSFT, YHOO, AMZN, ATHM, PALM, HAND, OMNY, AVGO, MCOM, RIMM, INSP, EBAY.

- "What is the 52-week range for ⟨*ticker*⟩? Start at finance.yahoo.com", where ⟨*ticker*⟩ is in the set MSFT, YHOO, AMZN, ATHM, PALM, HAND, OMNY, AVGO, MCOM, RIMM, INSP, EBAY.

- "What stocks were ⟨*upgraded/downgraded*⟩ to a ⟨*strong buy, buy, hold, sell*⟩ rating today? Start at finance.yahoo.com"

- "Find the research profile for ⟨*company*⟩. Start at finance.yahoo.com" ⟨*company*⟩ is one of: Microsoft, Yahoo, Amazon.com, Excite@Home, Palm, Handspring, OmniSky, AvantGo, Metricom, Research In Motion, InfoSpace, eBay.

- "What is the average analyst rating (recommendation) for ⟨*company*⟩. Start at finance.yahoo.com" where ⟨*company*⟩ is one of: Microsoft, Yahoo, Amazon.com, Excite@Home, Palm, Handspring, OmniSky, AvantGo, Metricom, Research In Motion, InfoSpace, eBay.

- "Find ⟨*item*⟩ for sale and report the highest bid (or report that none is for sale). Start at www.ebay.com.". ⟨*item*⟩ was one of: an Olympus D-340R digital camera; a Nikon Coolpix 950; a Nikon Coolpix 990, a Palm Vx, a Samsung Syncmaster 150mp flat panel display, a Pentax K-mount wide-angle lens, a Pentax K-mount 70-200 zoom lens, a Pentax K-mount telephoto (300mm or longer) lens; a 1900 Morgan Dollar; a 1938 Walking Liberty half-dollar, S mint mark; a 1976 proof quarter-dollar, S mint mark; a US commemorative "Hudson" half dollar circa 1935.

- "⟨*When/Where*⟩ is the lecture for CSE ⟨*class*⟩? Start at www.cs.washington.edu", where ⟨*class*⟩ was one of: 505, 531, 533, 573, 589, 594.

- "Is ⟨*room*⟩ available at ⟨*time*⟩ on ⟨*date*⟩? Start at www.cs.washington.edu", where ⟨*room*⟩ was either Sieg Hall 322 or Sieg Hall 114 and ⟨*time*⟩, ranged from 9:00 to 4:00 (by hour), and ⟨*date*⟩ ranged from November 13th to November 17th.

- "What office is ⟨*person*⟩ in? Start at www.cs.washington.edu", where ⟨*person*⟩ was one of sixteen selected graduate students or faculty at the University of Washington.

- "What is the topic of the upcoming colloquium, if any? Start from www.cs.washington.edu."

- "Is the colloquium in Sieg or in Kane, if any? Start from www.cs.washington.edu."

- "What is the top breaking news for today at cnn.com?"

- "What is the top business news for today at cnn.com?"

- "What's the latest news about ⟨*topic*⟩ at cnn.com?", where ⟨*topic*⟩ was chosen from: the first resident mission to the International Space Station (ISS); violence in the Mideast; the attack on the USS Cole; the U.S. presidential race; the Microsoft hacker attack; the Singapore Airlines plane crash in Taiwan.

Appendix B

RELATIONAL SCHEMATA FOR EVALUATION SITES

B.1 *www.gazelle.com*

The relations for `www.gazelle.com` take up to three parameters: Assortment, Product, and Collection. The domain hierarchies for these parameters are described explicitly in the KDDCup 2000 data.

- Home()
- Boutique()
- Departments()
- Legcare_vendor()
- Lifestyles()
- Vendor()
- AssortmentDefault()
- Assortment(Assortment)
- ProductDetailLegcareDefault()
- ProductDetailLegcare(Product)
- ProductDetailLegwearDefault()
- ProductDetailLegwearProduct(Product)
- ProductDetailLegwearAssortment(Assortment)
- ProductDetailLegwearProdCollect(Product, Collection)

- ProductDetailLegwearProdAssort(Product, Assortment)
- ProductDetailLegwear(Product, Collection, Assortment)

B.2 *www.cs.washington.edu*

The structure for `www.cs.washington.edu/education/courses/` was derived by reverse-engineering the structure of the existing site. The Term and Course domain hierarchies each contain a root node, a level of interior nodes (grouping courses by undergraduate, graduate, *etc.* and grouping terms by the academic year), and the ground leaf values. URL variables are URLs relative to the particular CourseSite(Course) or CourseOccurence(Course, Term) to which they apply. The domain hierarchies for URL and most other variables are flat, comprising only the root node and many leaf values.

- CourseWebs()
- CourseSite(Course)
- CourseSiteOther(Course, URL)
- CourseOccurence(Course, Term)
- CourseOccurenceOther(Course, Term, URL)
- CourseSampleCode(Course, Term, URL)
- Administrivia(Course, Term, URL)
- AllCoursework(Course, Term)
- CourseworkGeneralOther(Course, Term, URL)
- Coursework(Course, Term, Number)
- CourseworkCode(Course, Term, Number)
- CourseworkOther(Course, Term, Number, URL)
- Turnin(Course, Term, Number)

- AllExams(Course, Term)
- Exam(Course, Term, URL)
- AllLectures(Course, Term)
- LectureOtherGeneral(Course, Term, URL)
- Lecture(Course, Term, Number)
- LectureOther(Course, Term, Number, URL)
- MailIndex(Course, Term, SortBy)
- MailMessage(Course, Term, Number)
- Section(Course, Term, Section)
- SectionOther(Course, Term, Section, URL)

Appendix C

WEB LOG MINING

Web logs hold a wealth of information, much of which we have used in the work in this thesis. However, this information is often mired in a sea of misinformation: requests for pages that do not exist, transactions by users masquerading as others, interactions with content never requested from the server, *etc.* In this appendix, we outline several state-of-the-art techniques for separating the wheat from the chaff of web logs: inferring which visitor requested which pages, filling in missing requests hidden by proxies and caches, and building the sessions and trails used by web data mining systems. Although most of these techniques are used by researchers and practitioners, few have been treated in detail in the literature. One notable exception is discussion of the WEBMINER system [30], which describes how WEBMINER transforms an existing server access log into web sessions. Here we provide more details on how to build better logs, and how to infer web trails.

C.1 The web log

A web log is a transcript of transactions made between a set of users and a set of servers. Figure C.1 shows several lines of a typical log. For historical reasons, many web logs use the same format. The fields are separated by white space, typically a single space, although some fields are additionally quoted. Referring to Figure C.1, in order from left to right, the most important fields in the logs are:

- **Web client computer.** Either an IP address or the computer name. In an anonymized web log, this entry is omitted or replaced with a unique identifier that hides the client's true identity.
- **Time and date of request.** Often in square brackets, and including time zone offset from Greenwich Mean Time.
- **Requested document.** Technically, the field in the quotation marks is the first line sent from the client in the HTTP request. The first term is the HTTP verb: GET, HEAD, or POST being

```
rockhopper - - [01/Feb/2002:13:02:34 -0800] ”GET /homes/corin/ HTTP/1.0” 200 - ”-” ”Mozilla/4.79 [en] (X11; U; Linux 2.4.7-UP.26 i686)”
rockhopper - - [01/Feb/2002:13:02:36 -0800] ”GET /orgs/student-affairs/gsc/tgif/ HTTP/1.0” 200 1350
    ”http://www.cs.washington.edu/homes/corin/” ”Mozilla/4.79 [en] (X11; U; Linux 2.4.7-UP.26 i686)”
rockhopper - - [01/Feb/2002:13:02:37 -0800] ”GET /orgs/student-affairs/gsc/tgif/schedule.html HTTP/1.0” 200 3999
    ”http://www.cs.washington.edu/orgs/student-affairs/gsc/tgif/” ”Mozilla/4.79 [en] (X11; U; Linux 2.4.7-UP.26 i686)”
the4cs - - [02/Feb/2002:19:19:12 -0800] ”GET /homes/corin/job/ HTTP/1.0” 200 3216 ”-”
    ”Mozilla/4.75 [en] (X11; U; Linux 2.2.16-22 i686; Nav)”
the4cs - - [03/Feb/2002:23:15:40 -0800] ”GET /homes/stevaroo/ HTTP/1.0” 200 1614 ”-” ”Mozilla/4.75 [en] (X11; U; Linux 2.2.16-22 i686; Nav)”
rockhopper - - [04/Feb/2002:09:20:59 -0800] ”GET /490i HTTP/1.0” 302 246 ”-” ”Mozilla/4.79 [en] (X11; U; Linux 2.4.7-UP.26 i686)”
rockhopper - - [04/Feb/2002:09:20:59 -0800] ”GET /education/courses/cse490i/CurrentQtr/ HTTP/1.0” 200 16189 ”-”
    ”Mozilla/4.79 [en] (X11; U; Linux 2.4.7-UP.26 i686)”
rockhopper - - [04/Feb/2002:09:20:59 -0800] ”GET /home/cse.css HTTP/1.0” 304 - ”-” ”Mozilla/4.79 [en] (X11; U; Linux 2.4.7-UP.26 i686)”
rockhopper - - [04/Feb/2002:09:20:59 -0800] ”GET /home/cse2.js HTTP/1.0” 304 - ”-” ”Mozilla/4.79 [en] (X11; U; Linux 2.4.7-UP.26 i686)”
rockhopper - - [04/Feb/2002:09:21:04 -0800] ”GET /education/courses/cse490i/CurrentQtr/overview.html HTTP/1.0” 200 7883
    ”http://www.cs.washington.edu/education/courses/cse490i/CurrentQtr/” ”Mozilla/4.79 [en] (X11; U; Linux 2.4.7-UP.26 i686)”
rockhopper - - [04/Feb/2002:09:21:04 -0800] ”GET /home/cse.css HTTP/1.0” 304 - ”-” ”Mozilla/4.79 [en] (X11; U; Linux 2.4.7-UP.26 i686)”
rockhopper - - [04/Feb/2002:09:21:04 -0800] ”GET /home/cse2.js HTTP/1.0” 304 - ”-” ”Mozilla/4.79 [en] (X11; U; Linux 2.4.7-UP.26 i686)”
rockhopper - - [04/Feb/2002:09:22:12 -0800] ”GET /education/courses/cse490i/CurrentQtr/slides.html HTTP/1.0” 200 8529
    ”http://www.cs.washington.edu/education/courses/cse490i/CurrentQtr/overview.html” ”Mozilla/4.79 [en] (X11; U; Linux 2.4.7-UP.26 i686)”
rockhopper - - [04/Feb/2002:09:22:12 -0800] ”GET /home/cse.css HTTP/1.0” 304 - ”-” ”Mozilla/4.79 [en] (X11; U; Linux 2.4.7-UP.26 i686)”
rockhopper - - [04/Feb/2002:09:22:12 -0800] ”GET /home/cse2.js HTTP/1.0” 304 - ”-” ”Mozilla/4.79 [en] (X11; U; Linux 2.4.7-UP.26 i686)”
```

Figure C.1: **Sample web log entries.** Each non-indented line represents a distinct HTTP request made to `www.cs.washington.edu`. Indented lines continue the previous line's entry.

the most common. A GET request asks for the content of the document. HEAD asks only for the HTTP headers for the document—no content is sent. A POST request allows the client to send additional data to the server, for example, when making an e-commerce transaction. The second term is the document being acted upon, usually relative to the local server (*i.e.*, without the leading http://). The final term refers to the version of the HTTP that the client conforms to. HTTP/1.0 and HTTP/1.1 are most common.

- **Server response.** The web server sends a numeric status code to the client indicating the outcome of the HTTP request. Most common status codes are:
  - *200*: the document requested existed and was delivered.
  - *301*: the URL for the document has changed and the client should use the new URL (the server informs the client of the new URL)
  - *302*: the document exists under a new URL but the client should continue using the old URL.
  - *304*: the document has not changed (*i.e.*, the client should use the content stored in its local cache).
  - *404*: the document could not be located on the server.

- **Referring document.** When a client requests a document from a server, the client often provides the server the URL of the source of the request (*i.e.*, the URL of the page where the link was found). When this information is omitted (for example, if a user followed a personal bookmark, which does not have a URL), the field contains "-".

- **Web browser descriptor.** Also called the user-agent, this string describes the visitor's web browser. Most browsers identify themselves as "Mozilla" with additional information, such as the version number, the operating system, and perhaps the client language locale. The user-agent can also help identify web spiders; typically, any user-agent that does *not* conform to the standard descriptors (*i.e.*, those containing "Mozilla") is a spider. Also, spider web browser names may contain "bot," "spider," or "crawler."

- **User identifier.** Not shown in Figure C.1, one additional common field is a unique user identifier, or cookie. This identifier is given to the visitor in the first HTTP request their browser makes, and the visitor's browser sends this cookie to the server on each transaction.

C.2 Identifying users

One of the first steps in preparing web log data is to separate the page requests made by each unique visitor to the site. The most reliable approach is to ask users to log in with a username and password each session they visit the site. Of course, requiring visitors to explicitly log in may drive away a web site's customers, as well as add to the overhead of maintaining a database of users and passwords.

A more common technique that offers almost the same quality of matching users to requests is to sort by the users' unique identifier cookie. For users who accept and transmit cookies, this method will correctly segregate browsing by visitors[1]. Unfortunately some users prefer to reject cookies; for this group of visitors, the next best approach is to group together requests by client computer name or IP address. This approach will work well to group requests made in the same site-session,

[1]There are methods to maliciously confuse a web server using cookies; for example a visitor could manually change the value of their cookie to that of another visitor's cookie. A web site administrator must certainly ensure that their system is secure in the face of adversaries, but in this appendix we will assume that no such opponents exists.

but often is insufficient to track a user over a long period of time (because, for example, visitors entering from a dial-up modem may have a different machine name and address each visit).

All these approaches, however, are stymied by web proxies: third-party servers on the Web through which remote visitors direct all their browsing. The proxy caches web content on its local disks and passes along a request only when the local copy is out-of-date or missing. Thus, the proxy hides requests made by multiple visitors, and, even for individual requests, obscures the visitor's client computer name. To avoid the hidden-cached-content effects, a web server can mark its content as non-cacheable; thus, each request must be sent to the server. Of course, non-cacheable content can greatly increase the server's load, so this option should be chosen only for pages that truly should be tracked (*e.g.*, logged requests for inline images are often not useful for mining, so this content need not be marked non-cacheable). To deconflate the client computer name, the server can issue cookies to the client.

C.3 Gathering data

The content of web logs is often only requests made for web pages and images on the server—the logs do not contain information about how users interact more finely with content, or about requests made to other sites. We can peer into these additional actions using two popular techniques: web bugs and server-side redirection.

C.3.1 Web bugs

A web bug is a small inline image, typically 1-by-1 pixel, that is embedded on the web page. Whenever the visitor views the page, the web bug image is requested, and, thus, an HTTP transaction is recorded in the server log. We can use web bugs to observe whenever a visitor views or leaves a web page, even if that page is in the browser's cache. On each page we would like to monitor, we place a web bug and some JavaScript code that fires on the onLoad and onUnload events (see Table C.1). The code directs the browser to change the URL of the web bug, which generates a request to the server. The code includes a random string in the URL, which is ignored by the server, to ensure the browser will not fetch the web bug image from its cache. The result is that, whenever the user views (loads) the page or moves elsewhere (unloads), the web bug changes and the server

Table C.1: **JavaScript web bug.** Whenever a user views (loads) the HTML page or leaves to view another page (unloads), the web bug `/images/1x1.gif` is requested. The web bug URL is adorned with the URL of the page; a random number ensures the request has not been cached.

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript"><!--
function logpage(dir) {
  var url = location.href
  var i = url.indexOf("?")
  if (i > 0)
    url = url.substring(0, i)
  var rand = Math.random()
  im = "/images/1x1.gif?" + dir +
    "&" + url + "&" + rand
  document.images['1x1'].src = im
}
//--></SCRIPT>
</HEAD>
<BODY onLoad="logpage('load')" onUnload=="logpage('unload')">
<IMG ALT="" WIDTH="1" HEIGHT="1" NAME="1x1" SRC="/images/1x1.gif">
</BODY>
</HTML>
```

is informed, all without any apparent change to the visitor's view of the document.

C.3.2 Server-side redirection

Links followed on a web site to other pages on the same server will usually be observed by the server (modulo requests hidden by proxies). Requests made off-site, to servers elsewhere on the Web, are not reported. This information may be of interest, however; a site may wish to know which links are used by its visitors, regardless of the final destination. A technique to garner this information that has become popular recently is to redirect all links pointing off-site to a special script on the web server. This script does nothing more than direct the web server to return an HTTP status of "302: Temporarily moved" and send along the off-site URL (which was passed to the script in the HTTP GET request). The server records the HTTP status, and the user's client then makes the request to the off-site server. The disadvantages of this approach are that it increases the web server load and increases the visitor's delay in reaching the destination of the link. However, the gain in knowledge about site usage is often worth this price. Further, these costs can be reduced by enabling server-side redirection on only a fraction of the web requests, usually with an insignificant effect on accuracy.

C.4 Cleaning data

Given a log of web content accesses for a single visitor, the next step in the web log preparation pipeline is to remove requests that are irrelevant and to infer requests that are missing. For convenience, let us assume that the object is to model how users interact with pages (as opposed to sub-page objects, such as images); this assumption often holds in practice. Then, we may remove any log entries made for URLs whose extension is a known non-page type (*e.g.*, .gif, .jpg, .png, *etc.*). To be more complete, what we could do is parse each HTML page on the site, glean what objects are embedded on the page, and form a "kill list" of content to ignore. This approach is more complete because it would include embedded frames, which are HTML pages that appear as part of other pages, and images or other embedded content using non-standard naming conventions. Of course, the cost of parsing every web page on a site may be prohibitively expensive.

Another heuristic for inferring requests for embedded objects is the time delay between HTTP requests. If several web objects are requested within one to three seconds of each other, then it is quite likely that the requests after the first one are made automatically by the web browser, and not by the visitor quickly following several links. Thus, we may omit any requests made within a few seconds of a previous request. (Also of benefit is to compare the subsequent requests' referrer field with the previous request; if they match, the subsequent request is more likely to be an embedded object.)

Besides automatically requesting embedded content, web browsers may also request content on a regular time frequency if the page requests it. For example, `cnn.com` uses the "http-equiv" `<meta>` tag to indicate the page should be refreshed every 30 minutes. Thus, if the user viewed `cnn.com` at the end of the day on a Friday and did not return to the computer until Monday, then the server would have recorded over 100 page impressions for the site. Of course, none of those visits were "real" and should be removed if the goal is to model the user's actual navigation. In our MONTAGE work (Chapter 7), we compute the statistical mode of the revisit interval for each URL and, if at least 10% of the intervals belong to the mode, we remove any requests that are made within

a small tolerance of the mode. Thus, we effectively remove the second, third, fourth, *etc.* request for a page, but leave the first request (the actual visit the user made) intact.

Finally, not all page visits will generate a request to the server; some may be fulfilled by a proxy or the client browser's cache. We can infer these missing visits, however, by finding a likely path in the web site between each actual web request (and also taking into account each request's referring URL). The simplest case is when subsequent requests are made following a link from the previously-requested page. If the visitor uses the browser back navigation button to a previous page and follows a link, the server will see the request and the referring page; we can build a tree of this behavior as the shortest distance is the inferred "back" navigation (this technique is used in WEBMINER [30]). If the user followed several links serviced by a cache, and then a link that generated a request to the server, we can hypothesize likely paths through the site (perhaps using the same traversal algorithm as in MINPATH) between the last-known and present locations.

C.5 Sessionizing

The final step in mining the golden data from the web log mountain is to generate web sessions or trails. Techniques for generating (and definitions of) sessions and trails vary, but the common elements are: sessions are coherent requests in time, and trails are sessions coherent in space.

C.5.1 Building sessions

Intuitively, a session is the sequence of page requests made by a user in one "visit" to the site. More practically, a session is a sequence of page requests by a single user, each made within a small amount of time of the previous one, and followed by an idle period. The threshold for the idle timeout length can vary from 5 minutes to 30 minutes with approximately equivalent results.

C.5.2 Building trails

A trail is a sequence of page requests in which each subsequent request was made by following a link on the previous page, and doing so within a fixed amount of time (5 or 10 minutes). A session may give rise to many trails—one for each contiguous path of links followed. A session may form a tree (or forest) of navigation in a web site; a trail is a path from the root to a leaf in that tree. Thus, a single session may contain many trails, and the trails will likely contain the same requests (*i.e.*, a page request may belong to many trails, although to only one session).

C.6 Summary

This appendix summarizes the state-of-the-art in producing, cleaning, preparing, and mining web server access logs. Most of these techniques are used in research systems or fielded sites, although few written works have been published codifying them.

VITA

Corin Anderson earned B.S. degrees in Mathematics and Computer Science from the University of Washington in 1996. Corin continued at the UW, earning an M.S. and Ph.D. in Computer Science and Engineering in 1998 and 2002, respectively.