Adaptive Web Navigation for Wireless Devices

Corin R. Anderson, Pedro Domingos, Daniel S. Weld
University of Washington, Seattle, WA, USA
{corin, pedrod, weld}@cs.washington.edu

Abstract

Visitors who browse the web from wireless PDAs, cell phones, and pagers are frequently stymied by web interfaces optimized for desktop PCs. Simply replacing graphics with text and reformatting tables does not solve the problem, because deep link structures can still require minutes to traverse.

In this paper we develop an algorithm, MINPATH, that automatically improves wireless web navigation by suggesting useful *shortcut links* in real time. MINPATH finds shortcuts by using a learned model of web visitor behavior to estimate the savings of shortcut links, and suggests only the few best links. We explore a variety of predictive models, including Naïve Bayes mixture models and mixtures of Markov models, and report empirical evidence that MINPATH finds useful shortcuts that save substantial navigational effort.

1 Introduction

Perkowitz and Etzioni [1997] challenged the AI community to build *adaptive web sites*: sites that automatically improve their organization and presentation by learning from visitor access patterns. In the spirit of this challenge, many research projects have been proposed and implemented [Perkowitz and Etzioni, 2000; Fink *et al.*, 1996; Yan *et al.*, 1996; Jühne *et al.*, 1998; Joachims *et al.*, 1997; Pazzani and Billsus, 1999; Sarukkai, 2000]. Many of these projects, like much of the Web today, assume the visitor is browsing with a large color display and fast network connection. In addition, these works typically assume that a visitor's interest in a site lies in viewing many pages of content, as opposed to a specific destination. Consequently, the adaptations they emphasize include graphical highlighting of existing hypertext links and automatic generation of indices that link to many pages.

However, a growing community of web visitors does not fit this mold: *wireless web visitors*, who browse content from cell phones, pagers, and wireless PDAs. These mobile devices do not have the same display or bandwidth capabilities as their desktop counterparts, but sites (adaptive or otherwise) nonetheless deliver the same content to both desktop PCs and mobile devices. Moreover, our experience with mobile visitors indicates that they seldom browse through many pages, but rather are interested only in specific destinations, and would like to reach those pages by following as few links as possible. While this observation is also largely true of desktop visitors, reducing the number of links followed is particularly poignant for mobile visitors for two reasons. First, simply finding a specific link on a page requires the visitor to spend considerable time scrolling through the page on a tiny screen. Second, because of the low bandwidth and high latency of wireless networks, following a link to even a simple page of text can take as long as tens of seconds. Consequently, navigating through even four or five different pages can easily take several minutes, frustrating visitors to the point of abandoning their effort.

We believe adaptive web sites hold a great promise for improving the web experience for wireless devices, and, conversely, that wireless web browsing is a "killer app" of adaptive web sites. In ongoing research [Anderson *et al.*, 2001], we have developed a general framework for adapting web sites for wireless visitors that takes into account the value visitors receive for viewing a page and the effort required to reach that page (*i.e.*, the amount of scrolling and number of links followed). In this paper, we explore a particular adaptation for use in our framework: automatically adding *shortcut links* to each page a visitor requests. We assume that wireless visitor behavior is dominated by *information gathering tasks*, which can be accomplished by viewing specific destination pages on the site. Shortcut links connect two pages not otherwise linked together and can help visitors reach these destinations as quickly as possible. For example, if $A \rightarrow B$ denotes a link, and the only path from A to D is $A \rightarrow B \rightarrow C \rightarrow D$, then a shortcut $A \rightarrow D$ saves the visitor over 66% of the navigation effort, and perhaps more if one counts the scrolling avoided on the interim pages. Of course, there is a tradeoff between the number of shortcut links and their usefulness. In an absurd example, if one added shortcuts between every pair of pages on the site, a visitor could reach any destination in one link, but finding the right link would be practically impossible. Thus, we concentrate on generating only a small number of high quality shortcuts, for example, only as many as will fit on the wireless device without scrolling.

This paper makes the following contributions:

- We present an algorithm, MINPATH, for automatically finding high-quality shortcuts for mobile visitors in real time. Offline, MINPATH learns a model of web usage based on server access logs, and uses this model at runtime to predict the visitor's ultimate destination. MIN-

PATH is based on the notion of the *expected savings* of a shortcut, which incorporates the probability that the link is useful and the amount of visitor effort saved.

- We evaluate a variety of visitor models, including Naïve Bayes mixture models and mixtures of Markov models, and discuss their applicability in MINPATH.
- We provide experimental evidence that suggests a mixture of Markov models is the best model for MINPATH, and that MINPATH substantially reduces the number of links mobile visitors need to follow, and thus their navigational effort.

In the next section, we define our problem and present the MINPATH algorithm for finding shortcuts. Section 3 explores variations on the models for predicting web usage and section 4 evaluates MINPATH using these models. Section 5 discusses related research, and we conclude in section 6.

2 Finding shortcuts with MinPath

We begin by defining terminology, to facilitate the discussion.

2.1 Definitions

A *trail* [Wexelblat and Maes, 1999] is a sequence of page requests made by a single visitor that is coherent in time and space. Coherence in time requires that each subsequent request in the trail occurs within some fixed time window of the previous request. Coherence in space requires that each subsequent request be the destination of some link on the previous page. More precisely, if we denote the time of the request for page p_i as $\mathsf{time}(p_i)$, then a trail $T = \langle p_0, p_1, \ldots p_n \rangle$ is a sequence of page requests for which:

- $\forall\, i, 0 \leq i < n, p_i \rightarrow p_{i+1}$ exists; and
- $\forall\, i, 0 \leq i < n,$ $\mathsf{time}(p_i) \leq \mathsf{time}(p_{i+1}) \leq \mathsf{time}(p_i) + \textit{timeout}$

The *length* of a trail is n, the number of links followed. From the perspective of the adaptive web site which is watching a visitor's behavior midway through the trail, only a *prefix*, $\langle p_0, \ldots, p_i \rangle$ is known. The trail *suffix*, $\langle p_{i+1}, \ldots, p_n \rangle$, needs to be hypothesized by the adaptive web site.

2.2 Finding shortcuts

The objective of our work is to provide 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 $p_i \rightarrow q_1, \ldots, p_i \rightarrow q_m$ that *minimizes* the number of links the visitor must follow between p_i and the visitor's destination.

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 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.

2.3 The MinPath algorithm

If one had knowledge of the complete trail $\langle p_0, \ldots, p_i, \ldots p_n \rangle$, selecting the best shortcut at any page p_i would be easy: simply, $p_i \rightarrow p_n$. 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 trails 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: $0.6 \times 4 + 0.4 \times 2 = 2.4 + 0.8 = 3.2$.

The MINPATH algorithm is shown in Table 1. The ExpectedSavings function constructs the trail suffixes by traversing the directed graph induced by the web site's link structure (often called the "web graph"). 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 how we obtain the model required by MINPATH.

3 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 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

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 $\{q\}$
    $S \leftarrow$ ExpectedSavings($q, T, V, T_q, P_q, l+1, S$)
  Return S

Table 1: **MinPath algorithm.**

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. Only the evaluation of the model must run in real time.

3.1 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[1]:

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

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.

Because we have a limited volume of training data (approximately 129,000 page requests for approximately 8,000 unique URLs in a site with 240,000 web pages), we cannot build a model that predicts each and every page — many pages are requested too infrequently to reliably estimate their probability. Instead, we group pages together to increase their aggregate usage counts, and replace page requests by their corresponding group label (much in the spirit of [Zukerman *et al.*, 2000]). 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. The pages below each node share a common URL prefix, or *stem*, which we use as the label of the node. By varying the minimum usage threshold, we select more or fewer nodes; in section 4, we report how MINPATH's performance is correspondingly affected.

3.2 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:

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

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 [Cheeseman *et al.*, 1988]) 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 [Dempster *et al.*, 1977].

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 in the sum in Equation 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 4. 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

[1] More precisely, throughout our implementation we use MAP estimates with Dirichlet priors.

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 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 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. 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 $P(T_i|C_j)$ the probability that cluster C_j produced trail T_i; then $P(C_k|V) = \sum_{i=1}^{h} P(T_i|C_k)/h$.

3.3 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 previous 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 we did earlier, 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.

In addition to a single Markov model, we also evaluate MINPATH using a mixture of Markov models [Cadez *et al.*, 2000]. We use the same EM-based method to build these mixtures as we did to learn the Naïve Bayes mixture model.

3.4 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.

4 Results

We evaluate MINPATH's performance on usage at our home institution's web site. We used web access data during September 2000 to produce 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. We selected 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 bound.

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 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 test set is 3.42 links. 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 3.42 links and one link is the range of savings MINPATH can offer web visitors.

We first explored the relationship between the minimum URL usage threshold and the performance of MINPATH. We compared thresholds of 1% (which produces 42 URL stems), 0.5% (78 stems), 0.025% (1,083 stems), and 0.0% (all the unique URLs in the training data). We found that MINPATH's performance improves as we increase the number of URL stems, until the threshold falls below 0.025%. After that point, the average number of links per trail increases; we hypothesize that, because of data sparseness, we cannot learn the model as well. Thus, for all the experiments in this section, we use the best threshold we found, 0.025%. In ongoing work, we are evaluating the use of a lower threshold when MINPATH is given substantially more training data.

We next compared MINPATH's performance when using a variety of models (see Figure 1). The first column shows the number of links followed in the unmodified site. 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). This graph demonstrates 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 can save an average of 0.97 links, or 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 the 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 model for each test trail, and found the gains are significant at the 5% level. Finally, we found that the dif-

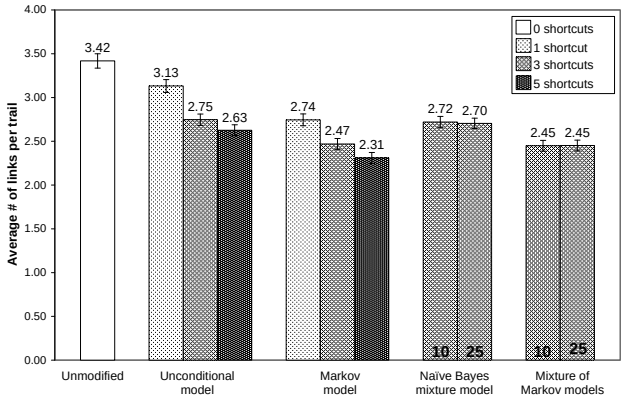

Figure 1: **MinPath's performance.** Each column shows the average number of links followed in a trail. The mixture model columns are annotated with the number of clusters. All error-bars denote 95% confidence intervals.

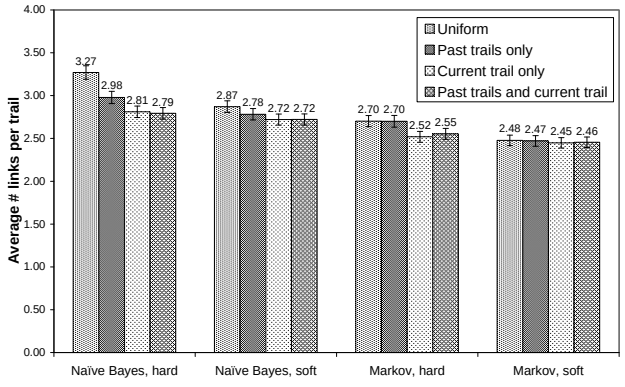

Figure 2: **Varying model assignment strategy.** Each of the four series represents a different model assignment strategy.

ferences between 10 and 25 clusters in the mixture are not statistically significant.

In Figure 2, we compare methods for selecting the mixture distribution for a trail prefix, using mixtures of 10 models. Each group of columns shows a different model and assignment type (hard or soft) combination. In each group, we 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. In future work we will investigate if this result remains true for larger sites.

Our last variation of models conditions the probability on the ordinal position of the page request in the trail. We compared the unconditional and Markov models against positional and Markov/positional models, choosing several values of the limit L of number of positions. In all cases, MINPATH did not perform significantly differently when using the positional information than when ignoring the position.

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, fast enough to suggest shortcuts in real time for wireless visitors.

5 Related work

Perkowitz [2001] addresses the shortcut link problem, but uses a simpler shortcut prediction method: for each page P viewed on the site, record how often every other page Q is viewed after P in some trail. When page P is requested in the future, the shortcuts are the top m most-requested pages Q. Effectively, this approach estimates the probability that a visitor at P will eventually view Q by counting how often this event has occured in the training data. MINPATH also estimates this probability, 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 that may not hold in practice. However, experience in other applications (*e.g.*, speech, NLP, computational biology) suggests the advantage outweighs the disadvantage. MINPATH offers two additional improvements relative to Perkowitz's approach. First, MINPATH can build more accurate models of visitor behavior by clustering visitors and building mixture models; in contrast, Perkowitz's approach builds a single shortcut table for all the visitors at the site. Second, MINPATH admits a more versatile selection of shortcuts. For example, we are currently extending MINPATH to 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.

Our MINPATH algorithm shares many traits with a number of web page recommendation systems developed in recent years. Letizia [Lieberman, 1995] 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 [Joachims *et al.*, 1997], Ariadne [Jühne *et al.*, 1998], and adaptive web site agents [Pazzani and Bill-

sus, 1999] 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 [Fu *et al.*, 2000] and PageGather [Perkowitz and Etzioni, 2000] suggest pages to visit based on page requests co-occurrent in past sessions[2]. 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.

The predictive web usage models we present are related to previous works on sequence prediction and web usage mining. These works are too numerous to review here, but we mention two closely related ones. Most similar to our own work, WebCANVAS [Cadez *et al.*, 2000] is a system for visualizing clusters of web visitors using a mixture of Markov models. We apply similar models to web behavior, although our goal is to build predictive structures, while WebCANVAS emphasizes visualizing the clusters themselves. Sarukkai [2000] uses a Markov model of web usage to suggest the most probable links a visitor may follow, and notes the need to reduce the size of the model by clustering the URLs. Our work explores this model as well as many others, and uses the expected savings of a link, not just the link probability, to sort the resulting suggestions.

6 Conclusions

Wireless web devices will soon outnumber desktop browsers, and sites must be prepared to deliver content suited to their unique needs. Because of the high cost of navigation on a mobile device, *shortcut links* are a fruitful adaptation to augment existing content. In this paper we have made the following contributions:

- We developed the MINPATH algorithm, which finds shortcut links targeted for each web visitor's information gathering behavior;
- We explored several predictive models of web usage and evaluated how they perform with MINPATH;
- We provided empirical evidence that MINPATH can find useful shortcut links. Using a mixture of Markov models, MINPATH can save wireless visitors more than 40% of the possible link savings.

MINPATH offers many fruitful lines of continued research, and we are currently exploring several directions. One direction is studying how MINPATH will scale to larger sites, with more pages, more links between pages, and more traffic. Another is automatically selecting concise and descriptive anchor texts for shortcut links. In a third direction we are integrating MINPATH with our general framework for adapting web sites, which includes a more elaborate visitor model and incorporates the cost of adding a shortcut link and the probability of erroneously following a shortcut. Finally, we are currently conducting a user study to evaluate MINPATH on a fielded web site.

[2] A session is like a trail but relaxes the requirement of coherence in space.

References

[Anderson *et al.*, 2001] C. R. Anderson, P. Domingos, and D. S. Weld. Personalizing web sites for mobile users. In *Proc. 10th Intl. WWW Conf.*, 2001.

[Cadez *et al.*, 2000] 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 *Proc. 6th Intl. Conf. on Knowledge Discovery and Data Mining*, 2000.

[Cheeseman *et al.*, 1988] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. AutoClass: A Bayesian classification system. In *Proc. 5th Intl. Conf. on Machine Learning*, 1988.

[Dempster *et al.*, 1977] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Stat. Soc., Ser. B*, 39(1):1–38, 1977.

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

[Fu *et al.*, 2000] X. Fu, J. Budzik, and K. J. Hammond. Mining navigation history for recommendation. In *Proc. 2000 Conf. on Intelligent User Interfaces*, 2000.

[Joachims *et al.*, 1997] T. Joachims, D. Freitag, and T. Mitchell. WebWatcher: A tour guide for the World Wide Web. In *Proc. 15th Intl. Joint Conf. on Art. Int.*, 1997.

[Jühne *et al.*, 1998] J. Jühne, A. T. Jensen, and K. Grønbæk. Ariadne: a Java-based guided tour system for the World Wide Web. In *Proc. 7th Intl. WWW Conf.*, 1998.

[Lieberman, 1995] H. Lieberman. Letizia: An agent that assists web browsing. In *Proc. 14th Intl. Joint Conf. on Art. Int.*, 1995.

[Pazzani and Billsus, 1999] M. J. Pazzani and D. Billsus. Adaptive web site agents. In *Proc. 3rd Intl. Conf. on Autonomous Agents*, 1999.

[Perkowitz and Etzioni, 1997] M. Perkowitz and O. Etzioni. Adaptive web sites: an AI challenge. In *Proc. 15th Intl. Joint Conf. on Art. Int.*, 1997.

[Perkowitz and Etzioni, 2000] M. Perkowitz and O. Etzioni. Towards adaptive web sites: Conceptual framework and case study. *Art. Int. J.*, 118(1–2), 2000.

[Perkowitz, 2001] M. Perkowitz. *Adaptive Web Sites: Cluster Mining and Conceptual Clustering for Index Page Synthesis*. PhD thesis, Dept. of Comp. Sci. and Eng., Univ. of Washington, 2001.

[Sarukkai, 2000] R. R. Sarukkai. Link prediction and path analysis using Markov chains. In *Proc. 9th Intl. WWW Conf.*, 2000.

[Wexelblat and Maes, 1999] A. Wexelblat and P. Maes. Footprints: History-rich tools for information foraging. In *Proc. ACM CHI 1999 Conf. on Human Factors in Comp. Sys.*, 1999.

[Yan *et al.*, 1996] T. Yan, M. Jacobsen, H. Garcia-Molina, and U. Dayal. From user access patters to dynamic hypertext linking. In *Proc. 5th Intl. WWW Conf.*, 1996.

[Zukerman *et al.*, 2000] I. Zukerman, D. Albrecht, A. Nicholson, and K. Doktor. Trading off granularity against complexity in predictive models for complex domains. In *Proc. 6th Intl. Pacific Rim Conf. on Art. Int.*, 2000.