To appear, IJCAI Workshop on Intelligent Techniques for Web Personalization (ITWP), 2001

Web Site Personalizers for Mobile Devices

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

Abstract

The fastest growing community of web users is that of *mobile* visitors who browse with wireless PDAs, cell phones, and pagers. Unfortunately, most web sites today are optimized exclusively for desktop, broadband clients, and deliver content poorly suited for mobile devices — devices that can display only a few lines of text using slow wireless networks.

To best serve the needs of this growing community, we propose building *web site personalizers* that observe the behavior of web visitors and automatically customize and adapt sites for each individual mobile visitor. In this paper, we give an overview of our approach to web site personalization as utility-maximizing search through the space of personalized web sites. Following this framework we have implemented two personalizers: PROTEUS and MINPATH. PROTEUS allows changes to site navigation (adding or removing links) as well as content manipulation (rearranging or eliding content), and evaluates the result with a learned model of the current visitor. MINPATH concentrates exclusively on adding "shortcut" links, but uses a model learned by clustering visitors based on their sequences of page requests. We introduce PROTEUS and MINPATH, and outline our current and future directions for these personalizers.

1 Introduction

The fastest growing community of web users is that of *mobile* visitors — people who browse the web with wireless PDAs, cell phones, and pagers. Ninety-five percent of cell phones sold today are "web-ready" and authorities predict that the number of wireless Internet devices will outnumber desktop computers by 2003. Despite this trend, however, few web sites today cater to mobile visitors, instead optimizing their content 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. Thus the user experience for mobile visitors at these "one-size-fits-all" sites suffers. To address this problem, we propose building *web site personalizers* that automatically adapt and personalize a web site to each individual mobile visitor.

Mobile web visitors exhibit a variety of browsing behaviors: random surfing, task completion (*e.g.*, buying stocks), information-goal seeking (*i.e.*, answering questions), etc. Information-goal seeking is of particular interest because it is generally *predictable*: visitors tend to have similar information goals in the future as in the past. Some example goals include: "What is the current stock price of MSFT?"; "Are there any Pentax K-mount zoom lenses on auction at eBay?"; "What office is Dan Weld in?". This behavior is predictable because visitors generally follow the same set of links, view the same set of pages, to achieve these goals each time, and attempt to do so in a direct and efficient manner. In addition, visitors tend to view pages with similar content as pages viewed in the past (*e.g.*, a photographer may frequently view pages containing words "zoom lens" and "f-stop", although the URLs requested may differ). By mining past interactions with the web site for these behaviors, we can automatically *personalize* the web content for each individual visitor. We envision *web site personalizers* that act on behalf of a mobile visitor to adapt web content as the visitor browses. A web site personalizer is an intermediary between the web site and the visitor and may be situated on the web server, on the visitor's device, or at a proxy server in between. A web site personalizer can:

- Make frequently-visited destinations easier to find, by highlighting relevant links or adding new links to a page.
- Highlight content that interests the visitor, by rearranging content on the page, or by adding visual cues.
- Elide uninteresting content and structure, replacing them simply with links to the omitted material.

Web site personalization follows a two-step process. In the first step, the personalizer builds a model of each visitor by mining the access logs and site content. The model includes information about navigational browsing behavior as well as content interests. This model could also include "out-of-band" information, such as visitors' geographic location or demographics. In the second step, the personalizer transforms the site to maximize the *expected utility* [9] for a given visitor. The expected utility of a personalized web site is a measure of how much benefit the visitor will receive by browsing the site; the personalizer computes this value based on the visitor model derived in the first step.

The focus of our work is in defining this framework of personalization as search and exploring how to instantiate this framework in a practical manner. This paper discusses our recent efforts along these lines and details current work. This paper draws heavily from previous papers describing our work [1; 2].

2 Personalization as search

We first describe our approach briefly. Our web site personalizer performs a search through the space of possible web sites. The initial state is the original web site of unmodified pages. The state is transformed by any of a number of adaptation functions, which can create pages, remove pages, add links between pages, etc. The value of the current state (*i.e.*, web site) is measured as the expected utility of the site for the current visitor. The search continues either until no better state can be found, or until computational resources (*e.g.*, time) expire.

2.1 State representation

Each state in our search space is an entire web site, W. Although an actual implemented system (such as those discussed in sections 3 and 4) may choose to personalize only a single page at a time, we model the entire web site to allow adaptations to be made anywhere in the site. The web site W is modeled as a directed graph whose nodes are pages, $p_0, \ldots, p_n$, and whose arcs are hypertext links, $l_0, \ldots, l_m$. A link l_k 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, and a is the anchor text. Each page p_i is modeled as a hierarchy of web content, much in the same way the parse tree of an HTML document confers a hierarchy of HTML tags. p_i 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 either plain text or (recursively) content nodes. The behavior B is the action that affects the human-viewable content. For example, if c were a "" node, then B would render its children in boldface; or if c were an "<a>" node, then B would render them as a hypertext link. Summarizing:

$S =$	$\{W_0, W_1, \ldots\}$	Each state in the space is a web site
$W =$	$(\{p_0, \ldots, p_n\}, \{l_0, \ldots l_m\})$	A web site is a directed graph of pages and links
$l_k =$	(p_s, p_d, a)	A link has a source, destination, and anchor
$p_i =$	c_i	A page is a 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

2.2 State Evaluation

We estimate the quality of the personalized web site as its expected utility from the point of view of the requested page. 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[1]. 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 over the wireless network. We transform this intuition into practice by recursively defining the utility of a page as the sum of its *intrinsic* utility — the utility of the page, in isolation — and its *extrinsic* utility — the utility of the linked pages[2]. We then calculate the utility of the site by evaluating this recursion beginning with the page requested by the visitor. We make these concepts more precise below.

Web site model for evaluation

We find it advantageous to transform the search state model slightly when calculating expected utility. Specifically, we now decompose a page p_i 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 screen s_{ij} is composed of web content (*i.e.*, text and graphics), which we denote as T_{ij}, and a set of links $l_{ij1}, ..., l_{ijk}$.

Expected utility

Let $\hat{p}$ be the personalized page for the requested URL u and let $U_V(\hat{p})$ be the utility of $\hat{p}$ for visitor V. The evaluation of W is the result of a recursive traversal through the site beginning with $\hat{p}$. Because only the first *screen* of $\hat{p}$ is initially visible to the visitor, the expected utility of $\hat{p}$ (or any p_i, in fact) is the expected utility of its first screen:

$$E[U_V(p_i)] = E[U_V(s_{i0})]$$

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

$$E[U_V(s_{ij})] = E[IU_V(s_{ij})] + E[EU_V(s_{ij})]$$

The intrinsic utility of a screen measures how useful the screen's content is towards fulfilling the visitor's information goal, independent of the rest of the web site. Typically, the intrinsic utility depends on the visitor model — past history and demographics. A more detailed description of intrinsic utility depends on particular assumptions regarding visitor interests and goals; section 3 discusses the method used in PROTEUS.

The extrinsic utility measures the value of a screen by its connections to the rest of the web site. To reach the rest of the site, we model the visitor as choosing from a fixed set of *navigation actions*, any number of which the visitor may select (*i.e.*, the actions are not mutually exclusive). Specifically, if the visitor is at screen s_{ij}, then 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. We maintain independent probabilities that the visitor will take each action, denoted as $P(\text{action})$, as well as the cost (*i.e.*, negative utility) each action imposes on the visitor, denoted γ_s and γ_l for scrolling and following a link, respectively. If we let d_{ijk} be the destination page of link l_{ijk}, then the extrinsic utility of screen s_{ij} is a sum weighted by probabilities:

$$E[EU_V(s_{ij})] = P(\text{scroll})(E[U_V(s_{i,j+1})] - \gamma_s) + \sum_k [P(l_{ijk})(E[U_V(d_{ijk})] - \gamma_l)]$$

[1] We actually compute utility at a finer granularity than a page.

[2] In many ways, intrinsic and extrinsic utilities are analogous to Kleinberg's authority and hub weights [10].

This equation recursively references 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 $E[U_V(s_{i,j+1})] < \gamma_s$ or $E[U_V(d_{ijk})] < \gamma_l$).

If evaluated naïvely, expected utility is not computationally tractable – it would require a screen-by-screen decomposition of potentially every page in the entire web site. Fortunately, the evaluation is made computationally much simpler with a few assumptions. First, when the cost of scrolling dominates the cost of following a link ($\gamma_l + \gamma_s \approx \gamma_l$), then we treat all pages but $\hat{p}$ as single-screen pages and can ignore the recursion due to scrolling on these pages. Second, we limit how deeply our recursion proceeds by setting a minimum threshold on the probability of viewing a screen or page — if the probability of it being visited is lower than the threshold, then it is excluded from the calculation. The threshold allows us 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.

3 Proteus

We have implemented this search framework in the web site personalizer PROTEUS. While most of the details of the implementation should be clear from the framework, we discuss a few implementation-specific issues here.

3.1 Search operators

To reduce the complexity of PROTEUS, we require that search operators directly affect the requested page $\hat{p}$ in some way, *e.g.*, adding links to $\hat{p}$ or manipulating content on $\hat{p}$. We exclude operators that, for instance, generate new pages or add links between two other pages.

PROTEUS supports two transformation operators[3]: elide-content and add-shortcut. elide-content replaces a block of content on $\hat{p}$ with a link to the original content in a fashion similar to Digestor [3] (Figure 1). The add-shortcut operator creates a new link from $\hat{p}$ to some other page that can be reached by following at most k links from $\hat{p}$. Thus, add-shortcut creates a link that "shortcuts" a longer path of links. For example, if the visitor previously followed the path $\hat{p} \rightarrow p_a \rightarrow p_b \rightarrow p_c \rightarrow p_d$, add-shortcut may create a link directly from $\hat{p}$ to p_d. Furthermore, add-shortcut places the new link $\hat{p} \rightarrow p_d$ next to the original link $\hat{p} \rightarrow p_a$, anticipating that, when the visitor wants to find p_d again, the visitor will look towards the link to p_a first. This placement is possible only if the visitor actually followed the path previously, otherwise, add-shortcut places the link near paths *other* visitors at the site have taken. The anchor text of the link is chosen heuristically as either the destination's <title> or <h1>.

3.2 Expected Utility

Our implementation measures intrinsic utility of a screen as a weighted sum of two terms: how well the screen's content matches the visitor's previously viewed content, according to a text-similarity measure, and how frequently the visitor viewed this screen. See earlier work [2] for more detail.

PROTEUS estimates the action probabilities by measuring the frequency with which the visitor took each 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. The probability for scrolling is derived empirically and is held constant at 0.85, although we are in the process of determining this number as part of the visitor model. Also through empirical evaluation, we set the cost of scrolling, γ_s, at 0.01 and the cost of following a link, γ_l, at 0.05. These values work acceptably in practice, although our results are largely insensitive to the exact values. In practice, the dominant term in the expected utility equation is the product of probabilities of taking chains of actions. For example, for all but the most probable links, the contribution of a remote page to expected utility is already vanishingly small, irrespective of the cost of following the link.

3.3 Results

In this section we present the results of a small user study of PROTEUS. We track ten test subjects' browsing habits on their desktop workstations and then measure how effectively they 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) required.

To collect training data, we asked the subjects to perform a suite of information-seeking tasks that we provided daily. We directed the subjects to attain their goals by browsing exclusively at the given site starting from a given page. An example question is: "Find the current stock price for MSFT, starting at `finance.yahoo.com`". The tasks in the seed suite were drawn randomly from a distribution of parametric questions and represent a coherent model of visitor interest.

Following the training phase, we asked the subjects to answer another suite of questions using a wireless Palm Connected Organizer. We asked them to answer the questions twice: once, on the unmodified web site, and again on a *personalized* version of the target site. PROTEUS personalized the target site for each visitor by first building a model of that visitor, based on the subject's past browsing data, and then creating adapted pages for the testing suite. 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 which pages for PROTEUS to adapt by using our human judgment of where the subjects would likely visit during the test. Note that we have not influenced the *personalization* at all — we have merely selected the subset of pages that PROTEUS personalizes, for efficiency.

Figure 2 compares links followed to attain each goal on the personalized versus unmodified web sites (the graphs of the time required and scrolling actions are similar). The y-axis shows the number of links while the x-axis shows the location of each goal listed chronologically. The graph shows

[3] PROTEUS supports a third operator, swap-siblings, but we omit its discussion for space considerations.

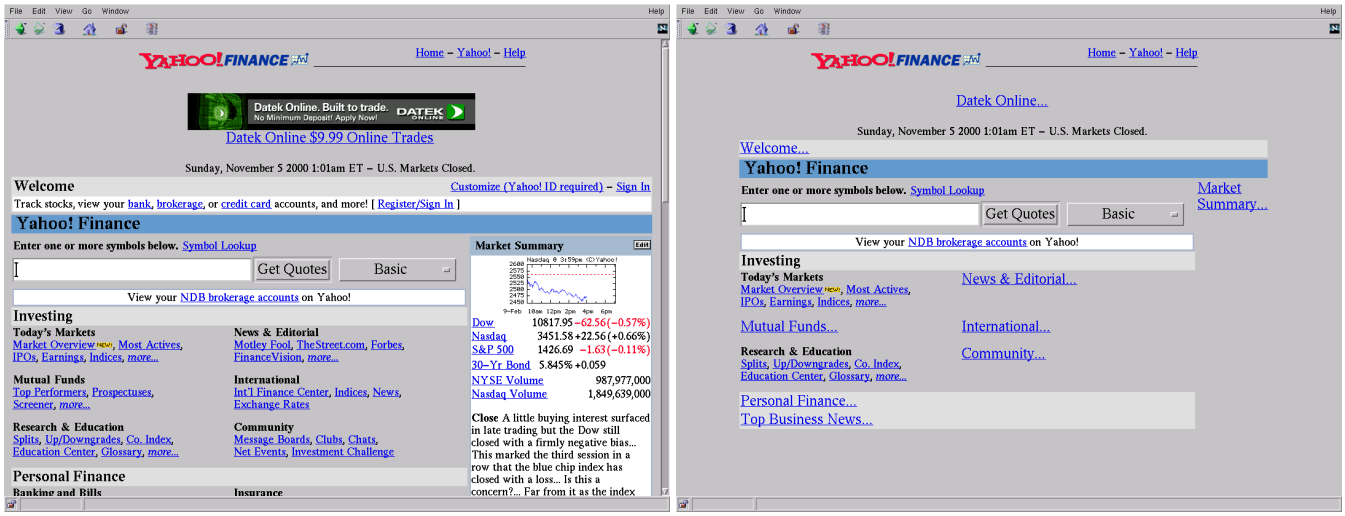

Figure 1: **Elided content.** On the left is an unmodified web page. On the right a number of blocks of content have been elided and replaced with hypertext links.

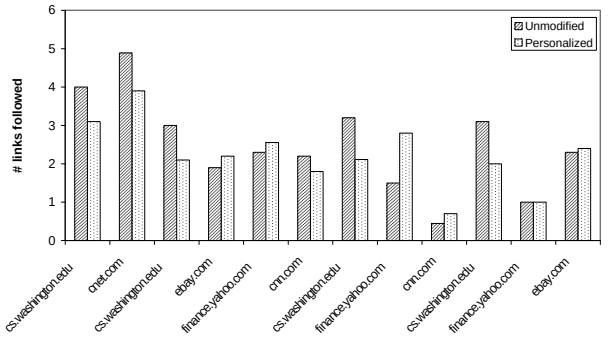

Figure 2: **Links followed.**

that for a majority of the sites, PROTEUS's personalizations appear quite useful: the addition of shortcut links and elision of unnecessary content reduced the amount of visitor navigation, and hence the amount of time spent, at the sites.

4 MinPath

One of PROTEUS's weaknesses is that it builds its visitor models in isolation of each other. When training data is sparse, these models can be quite inaccurate. Instead, we would like to combine data from many, similarly-behaving visitors to build more robust models. To this end, we developed MINPATH, an algorithm that finds high-quality shortcuts by modeling clusters of visitors, and using these models to predict where in the site the visitor is likely to travel. By leveraging data from many visitors in a single cluster, the models MINPATH builds are more accurate and lead to better personalizations. Moreover, while not as general as PROTEUS's expected utility, MINPATH's evaluation metric — expected savings — can be very efficiently computed.

4.1 The MinPath algorithm

A *trail* [16] $T = \langle p_0, p_1, \ldots p_n \rangle$ is a sequence of page requests such that each request occurs within some fixed time window of the previous request and is the destination of a link on the previous page. The personalizer watching a visitor's behavior midway through the trail sees only a *prefix*, $\langle p_0, \ldots, p_i \rangle$. The trail *suffix*, $\langle p_{i+1}, \ldots, p_n \rangle$, must be hypothesized by the personalizer.

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 is easy: simply, $p_i \rightarrow p_n$. Of course, given only a trail prefix, the personalizer must infer the remaining pages. Our approach uses 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 possible trails originating from p_i. Given a suffix and its probability, we assign an *expected savings* to the shortcut $p_i \rightarrow q_j$ for 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 many trail suffixes may pass through the same page q_j, and so the expected savings of a shortcut $p_i \rightarrow q_j$ is summed over all suffixes.

For example, suppose that a visitor requests the trail prefix $\langle A, B, C \rangle$ and we wish to find shortcuts to add to page C. Suppose further 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 probability 0.6, and $\langle I, J, H, K \rangle$ with probability 0.4. The expected savings from the shortcut $C \rightarrow E$ would be $0.6 \times 1 = 0.6$, because the trail with 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: $0.6 \times 4 + 0.4 \times 2 = 3.2$.

MINPATH constructs 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, MINPATH 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 is the product of the probability of reaching that page and the number of links saved. MINPATH collates the results and returns the best shortcuts. We next describe how we obtain the model required by MINPATH.

4.2 Predictive Models

The key element to MINPATH's success is the predictive model of web usage. The probabilistic model predicts 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: $P(p_i = q | \langle p_0, \ldots, p_{i-1} \rangle, V)$. Of course, a model may condition this probability on only part or even none of the available data; we explore such models in our experiments. We fit the models to past web usage data, either to all the visitors to the site at once, or to clusters of visitors. We group visitors by clustering their sequences of web page requests, and use EM to simultaneously find the clusters and fit the models. For the *en masse* and clustering approaches, we evaluate two types of models: unconditional, which predicts the next link without regard to previous browsing, and Markov, which predicts the next link given the previous request. When applied to clusters of visitors, we effectively have *mixture models*, either Naïve Bayes mixture models [6] or mixtures of Markov models [5]. We describe these models in more detail in our other work [1].

4.3 MinPath results

We evaluate MINPATH's performance on usage at our home institution's web site. We use web access data for 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 (1.5 days); the time period from which the test trails were drawn occurred strictly after the training period. We selected only those trails with link length at least two, because shorter trails cannot be improved. We measure MINPATH's performance by the number of links a visitor must follow to reach the end of the trail.

We compare MINPATH's performance using the models described earlier (see Figure 3). 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, each fitted to all the site's visitors, and produces 1, 3, or 5 shortcuts per page[4]. 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 that the shortcuts MINPATH finds are of high quality — when using a mixture of Markov models and suggesting just three shortcuts, MINPATH can eliminate an average of 0.97 links, or 40% of the possible savings. Of course, the actual effectiveness of adding shortcuts will depend on the visitor's ability to discern whether the link will be of value, and in section 6 we describe our ongoing work to intelligently select appropriate link anchor texts. Finally, we note that MINPATH's running time is quite small. The models 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. MINPATH's expected savings computation is substantially faster than and positively complements PROTEUS's more general approach.

[4]We limit MINPATH to only as many shortcuts as can reasonably be displayed on the small screen of a wireless web browser.

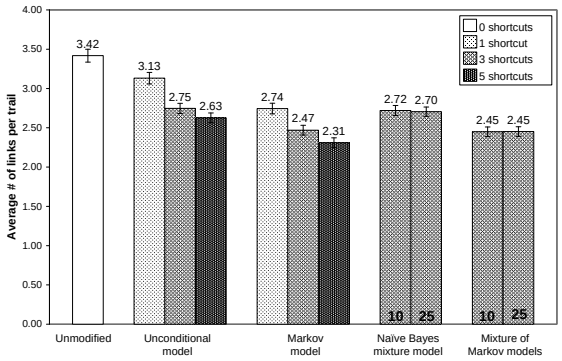

Figure 3: **MinPath's performance.** Each column shows the average number of links followed in a trail. Mixture model columns are annotated with the number of clusters. Error-bars denote 95% confidence intervals.

5 Related work

Two closely related lines of research are IndexFinder [15] and Digestor [3]. IndexFinder creates singular transformations that appeal to all visitors at the site, in particular, generating new *index pages* — hubs of links to other pages on the site. These pages are evaluated based strictly on the navigational usage patterns of past visitors — the pages requested — irrespective of their content. Digestor optimizes pages for small-screen display and uses a steepest-descent search similar to ours. However, Digestor rates the quality of a web page simply by how much screen space it occupies, a metric that encourages degenerate pages — a blank web page receives the highest quality value. In contrast, our approach personalizes content per visitor, and evaluates the adaptations using a principled, utility-maximizing approach.

The Web Browser Intelligence (WBI) [12] project proposes an architecture of pluggable intermediaries between web servers and clients. These intermediaries generate, transform, and monitor the content they see as visitors browse, and can be used either individually or in chains. An interesting line of future research would be to integrate PROTEUS into the WBI framework and investigate what what other intermediaries could be usefully composed with PROTEUS.

Countless other systems attack all or part of the web site personalization problem; we mention briefly several related systems. The Daily Learner [4] learns a Palm VII user's preference for news content, by monitoring exactly which stories the user requests from both the Palm device and the user's corresponding desktop computer. Mobasher *et. al.* [13] mine web usage patterns and web content to personalize the visitor experience, specifically, to recommend new content the visitor may like to see. The PersonalClipper [7] 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 the mobile device. Letizia [11], WebWatcher [8], and adaptive web site agents [14] guide visitors by suggesting pages they may like

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.