Simplifying Web Traversals By Recognizing Behavior Patterns

Christian Doerr, Daniel von Dincklage, and Amer Diwan
Dept. of Computer Science, University of Colorado at Boulder
Boulder, CO 80309, USA
{Christian.Doerr, Daniel.vonDincklage, Amer.Diwan}@colorado.edu

ABSTRACT

Web sites must often service a wide variety of clients. Thus, it is inevitable that a web site will allow some visitors to find their information quickly while other visitors have to follow many links to get to the information that they need. Worse, as web sites evolve, they may get worse over time so that all visitors have to follow many links to find the information that they need.

This paper describes an extensible system that analyzes web logs to find and exploit opportunities for improving the navigation of a web site. The system is extensible in that the inefficiencies that it finds and eliminates are not predetermined; to search for a new kind of inefficiency, web site admininstrators can provide a pattern (in a language designed specifically for this) that finds and eliminates the new inefficiency.

Categories and Subject Descriptors: H.3.3 Information Systems, Information Search and Retrieval I.5.5 Pattern Recognition,Miscellaneous

General Terms: Measurement, Performance

Keywords: web usage mining, behavior pattern recognition, navigation improvement

1. INTRODUCTION

Web sites often service a wide variety of clients. For example, students may visit the university's web site to look up the "Calendar of Events"; non-students may visit this web site to get directions from the airport. Over time these patterns may change. For example, each week students may visit a different "week" under the "Calendar of Events" web page to look up events for the current week. Thus, a good web site must (i) efficiently and effectively provide information to a wide variety of clients and (ii) change to accommodate the changing needs of its clients. If a web site does not meet these requirements then at least some clients will find it difficult to navigate.

HT'07, September 10–12, 2007, Manchester, United Kingdom.

Furthermore, as web sites evolve over time, they grow in content but also in complexity. If the web site is large, such as a web site for an institution, the evolution of the web site can result in information that is buried deep inside a hierarchy of web pages. Moreover, information may end up in places where it does not belong. Thus, as web sites evolve, they become increasingly difficult to navigate. For these reasons, even well-designed web sites may be difficult for many clients to navigate. These clients will have to follow links back and forth until they find the information that they need. Zhu [15] reports that "moving back and forth between links and the main nodes creates distruption and discontinuity" causing "disorientation and cognitive overload". This paper describes a system that automatically and dynamically adapts web sites based on their usage patterns so that they are easy for their clients to navigate.

We show that tracking and recognizing patterns in web logs can identify and fix navigational problems in web sites. More specifically, we identify opportunities for (i) automatically redirecting visitors (all or a subset) from a web page to another web page (*redirect*) and (ii) providing visitors (all or a subset) with a quicklink that enables them to get to their target without visiting all intermediate pages (*quicklink*).

While prior work has tracked and analyzed user traversals of web sites in order to address navigational problems, these approaches focused only on a single web site at a time: the user analyzes and improves each web site individually. This paper argues that different web sites often exhibit similar problems. Moreover, these problems and their solutions can be expressed naturally as high-level rules that are widely applicable to diverse web sites.

We describe and evaluate a system, `flexiweb`, that applies such high-level rules to perform the redirect and quicklink optimizations described above. These rules may be customized to a particular web site or may be general and thus applicable to any web site. The notation for rewrite rules is designed to be easy to use; yet it is expressive enough for many optimizations. We demonstrate our system by applying it to a number of different optimizations.

The contribution of our paper is two-fold: First, we introduce the concept of web site-independent navigational problems. Second, we propose a new query language that can be used to universally describe common user behavior patterns

on web sites. Patterns and navigational remedies expressed using this language are independent of one particular implementation and can be shared and applied across various sites thus making navigational remedies more portable.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 shows examples that show the existence of implementation-independent behavior patterns. Section 4 describes the system on a high level. Section 5 describes the language and Section 6 shows how to use our notation to express the patterns along with their optimizations presented in Section 3. Section 7 evaluates our work by proving the need for patterns, describing the expressiveness of the language, verifying that users adopt quicklinks and measuring the performance of `flexiweb`. Section 8 concludes and outlines future work.

2. RELATED WORK

Prior work falls into four categories: visualizing log files, algorithmic approaches for analyzing behavior, navigational assistants, and adaptive hypermedia. The former two categories relate to deriving and tracking of usage patterns, the latter two categories mainly address navigational problems.

2.1 Visualizing log files

This area combines a web site's structure (i.e. the individual pages and the links between them) with traversals of this site. This combined information gives webmasters a better understanding of how visitors use a site and how the site might be improved. These approaches typically use hyperbolic trees and discs in their visualization.

Examples of visualization using hyperbolic trees are Munzner's Site Manager [6] and Wexelblat and Maes' footprint system [13]. Since hyperbolic trees show every connection within the web graph, they are most useful for relatively small web sites. Large web sites are typically visualized using discs which display the graphs as a directed acyclic graph. This reduces complexity but also the accuracy of the display. Further, visualization is always connected to aggregation during which important information about single paths that users have taken is lost. Therefore, it can be hard to develop a detailed understanding about behavior patterns and identify improvements to navigational structure on a purely graphical representation.

While our approach is not directly concerned with visualization of web graphs, we can use similar analysis capabilities as a pre-processing tool to zoom in on specific parts of interest in a complete web log, thus reducing the complexity of the graph to be visualized. `flexiweb` can then supply this pre-analyzed behavior data in many output formats that are suitable for visualization with the tools mentioned about.

2.2 Algorithmic Approaches

To overcome problems with aggregated data, algorithmic approaches to log analysis have been developed. These systems analyze the behavior of a single user at a time to find patterns that can explain their navigational behavior.

Tools such as the one proposed by Chen et al. [2] analyze web logs for association rules and identify which pages are related to each other, i.e., given that a user visits a certain page which other pages is she likely to visit as well. Other algorithmic approaches search for navigational clusters [14, 8] to identify navigational behavior patterns that are unique to specific user groups. Pei et al. [7] and Perkowitz and Etzioni [8] further detect sequential patterns which identify paths that are repeatedly traversed on a web site.

Other researchers have created web log mining techniques and tools, such as WUM [11] and Webminer [5] that facilitate the data mining process to find potential navigational improvements. Using these tools, a web master can use SQL-like query languages to investigate for example how many users a traversing a link between two specific pages and count other events that occurred in the logs.

Our approach can express all the algorithms of the related work described above and provides further extensibility through its pattern language. In comparison to tools such as WUM our approach finds patterns on a more abstract level. For example, instead of manually searching for patterns by looking at the pages that users visit after visiting a particular page, X, in `flexiweb` you can describe the pattern on an abstract level and the system will find all concrete pages to which the pattern applies.

2.3 Navigational Assistants

Navigational assistants operate on the client-side to detect and instantiate navigational improvements. In this line of research, a stand-alone application monitors or is explicitly told the user's preferences and interests and as a result suggests or predicts future pages to visit and may even modify the design of the web page to help the user better identify the links of interest.

Following this approach, Lieberman's Letizia [4] tries to recommend pages the user could be interested in based on the previous search history and automatically searches the website for similar items. Similarly, Davidson [3] predicts future user behavior based on a content analysis of the HTML content, and uses this knowledge for prefetching content.

Tsandilas and Schraefel [12] developed an "adaptation controller" with which users can directly select topics of interests. The stand-alone application then modifies link colors and font sizes to highlight links of high importance based on the users settings in the adaptation controller.

The major difference between these systems and our approach is that navigational assistants require a tool to be installed at the client side which is infeasible in our setting as we don't have control over all clients accessing out web sites. Further, these approaches only consider local behavior and therefore are blind to global trends on a certain web site, for example highlighting items that are currently of interest to many people.

2.4 Adaptive Hypermedia

Adaptive hypermedia uses server-side tools that adapt the content of pages depending on the user visiting that page. De Bra et al.'s Tool AHA! [1, 9], for example, allows dynamic rewriting of a web site, based on a user model that encodes previous user actions and history of that particular user with

the site. This work also adapts navigation for individuals or classes of users, but does not automatically detect user behavior patterns or allow for a cross-site use of patterns.

3. PATTERNS

The original motivation for our system came about because we manually discovered a number of optimization opportunities in web sites that we were involved in. In this Section, we now discuss four of such optimizations as examples for patterns that can occur across multiple sites. Each discussion includes an ad-hoc formalization of the behavior to be matched. The full formalization is discussed in Section 6. These are four of the many possible optimizations that our system can perform (see Section 5).

We describe these optimizations in terms of operations on web graphs. A web graph represents a complete or partial web site. Nodes in the web graph represent pages. Edges represent links between the pages.[1] We build these web graphs by processing web logs. The thickness of the edges in a web graph indicates the frequency with which users traverse the corresponding link. The time attribute of a node gives the average time spent in that node before a user follow links to other nodes. If users do not leave a node, that node does not have a time attribute.

Our optimizations transform web graphs in two ways: (i) by adding links that enable users to get to the information of interest quickly (thus these are called "quicklinks"). We represent "quicklinks" as dashed edges in the web graph; and (ii) by adding transitions that automatically move visitors to one web page to another web page (we call these "redirects").

3.1 Skipping irrelevant nodes

Scenario. Imagine a professor's "My Teaching" web site which links to the home pages of all the classes that the professor is teaching. Further assume that the professor teaches only a single class per semester. While the web page is a useful way for the professor to organize her classes, it is just an extra web page that students need to visit on their way to a class web page. More specifically, most students who visit the "My Teaching" site are only interested in the course that the professor is currently teaching. We can improve the user experience with this web site by automatically redirecting visitors to the course page for the current semester. More specifically, we can automatically redirect any visitors to the "My Teaching" page to the current semester's course page. To support visitors that are interested in classes from other semesters, this optimization also provides a back link so that the visitors can get to the "My Teaching" page without being redirected. Note that our redirect does not apply if the page whose accesses are being redirected is accessed from the target of the redirect. This prevents endless loops.

Generalization. Large web sites often have "gateway pages" (e.g., the "My Teaching" page). These pages may announce information or contain links that most users are not interested in. These pages can benefit from the "skipping irrelevant nodes" optimization described here.

[1]We use nodes/pages as well as edges/links interchangeably.

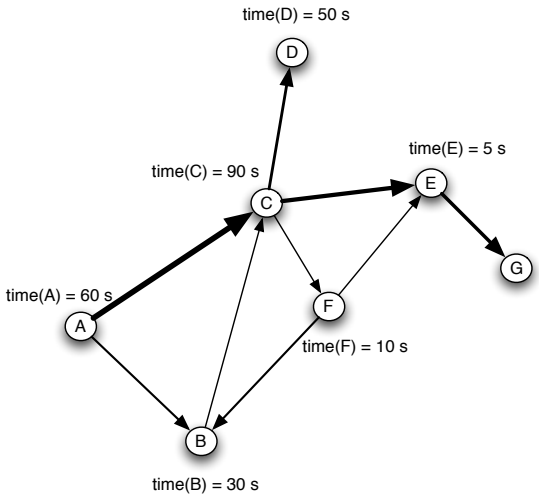

Figure 1: Irrelevant node: All users leave node E always towards node G with little average time spent within E - this node is likely to be irrelevant

Optimization. We can identify opportunities for this optimization by identifying pages that (i) most users leave towards a single other page; and (ii) most users spend little time on. Figure 1 models such a situation. Users spend little time on Node E and moreover all the visitors of Node E leave it for node G. Thus, we can automatically redirect visitors from of E to node G, saving users the additional step of having to visit node E and click on a link.

Node B satisfies condition (i) for the optimization but not condition (ii). While Node B has a single outgoing edge, users spend a significant amount of time in the node. Thus, Node B must contain useful information and thus we should not redirect visitors to Node B to Node C.

Node F satisfies condition (ii) for the optimization but not condition (i). More specifically, even though users spend little time at Node F, it still provides an important decision point to the user: they may leave F in equal proportions to go to Node B or to Node E. If we automatically redirected Node F to Node E (or Node B) we would be wrong half the time. Thus, we do not apply this optimization to Node F.

The following pattern describes the criteria for identifying nodes to which we can apply the "skipping irrelevant nodes" optimization. More specifically we search through all paths taken by users through the web site and identify nodes i in which users spend less time than t_{time} and leave i for the next node in the path at least t_{freq} percent of the time:

exists $p : Path$ **in**
  exists i : 0 <= i < (p.size - 1) **in**
    (avg. time spent in i^{th} node) $< t_{time}$
    && (fraction of visits to i^{th} node of p
    when visitors leave for the $i+1^{th}$ node in p) $> t_{freq}$

We redirect all visitors of $p[i]$ to $p[i+1]$. The above pattern iterates over all paths to make sure that the optimization only considers edges that users actually traverse. Once it has an edge, the pattern incorporates global information (e.g., average time spent in in a node) to determine if the edge is worth optimizing from a global perspective.

Note that our system communicates optimizations if applied and their effects to the user, so users know when a page is skipped and can backtrack to the skipped page. We can use techniques such as exponential weighting to penalize incorrect redirects (i.e., when users traverse the backtracking link). If a redirect accumulates enough penalty we can disable this optimization. In this way our system addresses the criticism that users do not have control over adaptive hypermedia systems [10].

3.2 Skipping irrelevant nodes on a per-user basis

Scenario. Imagine that our professor now teaches two courses per semester: a graduate course and an undergraduate course. Now, visitors to the "My Teaching" page may go to either the graduate course page or the undergraduate course page; we cannot redirect all visitors to the same page anymore. Most students will take either the graduate or the undergraduate course. If we track the history of individual students then we may still be able to redirect them to the relevant page. Of course, we cannot redirect students that are taking both courses (e.g., sitting in the undergraduate course as a remedial measure).

Generalization. Even though we cannot apply the "skipping irrelevant nodes" optimization globally, we can still apply it if we track information and apply the optimizations on a per user basis. This optimization will be particularly effective for web sites that require users to log in (and thus our system knows the identity of the user). If a web site does not require users to log in, we use standard heuristics (e.g. IP addresses and user agents) to make a best-effort attempt at user identification.

Optimization. We can identify opportunities for this optimization using the same criteria as the optimization in Section 3.1 except that we consider one user at a time rather than all users. The following pattern describes the criteria for triggering this optimization:

exists u:User **in**
  exists p:Path < User=u > **in**
    exists i:0 <= i < (p.size - 1) **in**
      (avg. time u spends in i^{th} node) $< t_{time}$&&
      (fraction of u's visits to i^{th} node of p when
      u leaves it for the $i+1^{th}$ node in p) $> t_{freq}$

3.3 Providing shortcuts to popular targets

Scenario. Imagine now that as the semester progresses, the professor adds information to the current course web sites. Rather than adding all the information at the top level, she organizes the site hierarchically so that users are not confronted by a huge number of links on any page. Unfortunately, as a consequence, students now have to traverse many links to get to this week's assignment: they have to go from the class' main page, to the weekly schedule, and finally to the current assignment. In the week before an assignment is due, many students will follow the above sequence of links to find the assignment. The "Providing shortcuts to popular targets" optimization automatically provides a quicklink to the current assignment from the class's main page so that students can get to the current assignment in fewer links.

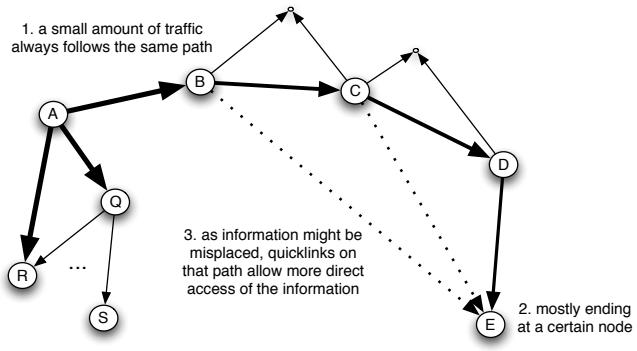

Figure 2: Similar Behavior: Many users that arrive at node B from node A will continue to node E

Generalization. Web site designers often organize information in deep hierarchies so that no single web page is overly complex. However this means that users need to click on many links before they can get to the information that they need. Some pages may be interesting to many users. The "Providing shortcuts to popular targets" optimization targets these pages.

Optimization. We can identify opportunities for this optimization by identifying pages that (i) many users visit and do not follow any links out of that page (i.e., it is the page they were looking for); and (ii) there is a path (or more than one path) that users frequently traverse in order to get to the page. Thus, we can put quicklinks from all nodes in the path to the page of interest. Figure 2 models such a situation. Once users get to Node E they stay there and many users follow the path B →C→D→E to get to Node E. Thus we insert quicklinks (shown with dotted arrows) from nodes B and C to Node E. Note that it is not necessary that all users follow the same path (as symbolized by the other edges leaving nodes B, C and D), it just has to be a share large enough to be worth optimizing for.

The path to Node E starting at Node A does not satisfy condition (ii) because more users leave Node A for Nodes Q and R than they do for Node B. Thus, a quicklink from Node A to Node E will be wasteful for most visitors to A. Path B→C→D satisfies condition (ii) for the optimization but Node D does not satisfy condition (i). Thus, we do not add quicklinks from Nodes B to Node D. We now develop the pattern for this optimization more formally in three steps. The first step finds paths that contain a node (at position k) where a significant percentage of traversals stop. This step identifies pages that satisfy condition (i) above.

exists p:Path in
  exists k:0 <= k < p.size **in**
    (percentage of traversals that stop at
    k^{th} page in p) > t_{stop}

The second step finds a node (at position j) in the path such that the path from this node to the k^{th} node (as found earlier) is commonly traversed; i.e., each edge is relatively frequently traversed.

exists j:0 <= j < k **in**
  forall q:j <= q <= k **in**
    (fraction of traversals that leave q^{th} node
    for $(q+1)^{th}$ node) > t_{freq}

Since we want a significant percentage of users to traverse this path, we have a problem if we stop with above. For example, if t_{freq} is 70% then after traversing five links we may have as few as 17% of the users left (0.7^5). The third step ensures that a significant fraction of users that enter the j^{th} node of p actually get to the k^{th} node. Collectively, the second and third step enforce condition (ii) above.

(fraction of users entering j^{th}node of p that get
  to k^{th} node of p) > r_{min}

Given a path and a node that satisfies the above pattern, we can now insert quicklinks. To prevent pages from ending up with too many quicklinks we decay the quicklinks over time: if few users use a quicklink then it eventually disappears.

3.4 Reacting to temporal phenomena

Scenario. In the example in Section 3.3 we added quick links to assignment pages. However, Section 3.3 ignored the temporal aspect of assignments: i.e., each week the assignment that students will want to go to changes. Thus, rather than analyzing all the paths to find opportunities for quicklinks, we should instead focus on the most recent behavior and expire quicklinks as soon as the corresponding assignment ceases to be popular.

Generalization. This optimization is identical to "Providing shortcuts to popular targets" except that it considers only recent behavior.

Optimization. We can identify opportunities for this optimization identically to "Providing shortcuts to popular targets" except that we will look only at the most recent paths. This optimization looks identical to "Providing shortcuts to popular targets", except that the first step is:

exists p:Path < Time=last2weeks > **in**
  exists k:0 <= k < p.size **in**
    (percentage of traversals
    that stop at k^{th} page in p) > t_{leave}

4. IMPLEMENTATION

We now describe the overall implementation of our system. Section 5 describes our notation for specifying optimizations. `flexiweb` has three components: *log analyzer*, *page optimizer*, and *page generator* (Figure 3).

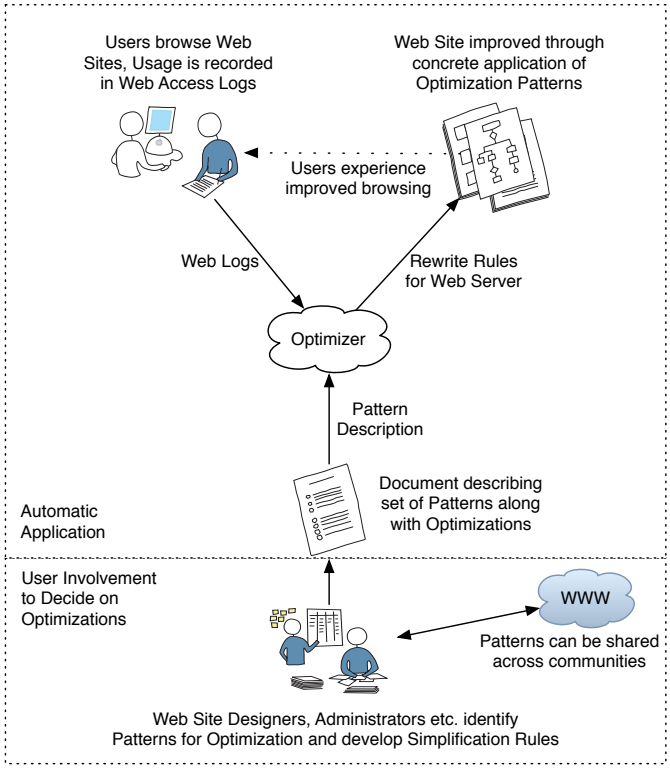

Figure 3: Process for optimizing a web site

The *log analyzer* periodically samples the log files of a web site to extract information about web page accesses. The log analyzer then groups web page accesses into paths. Each path is a sequence of pages visited by a specific user. The log analyzer uses standard heuristics (e.g. hostnames, login information, access times) to make a best-effort attempt at grouping page accesses into paths. The *page optimizer* uses the output of the log analyzer to determine which optimizations to perform. Finally, the *page generator*, which plugs into a web server (Apache in our implementation) applies the optimizations to the web pages before returning them to a client. Figure 3 illustrates this pictorially.

There are three steps to using our system.
First, the web site designers and administrators need to decide which patterns to use in order to improve the browsing experience of their web site's visitors. They may write their own patterns or obtain them from elsewhere. Since patterns model general problems rather than concrete situations they are easily shared.

Second, `flexiweb` uses these patterns to automatically rewrite web pages in response to user requests. Since our system works exclusively on the server side, clients do not need to do anything special to benefit from the optimizations. Despite `flexiweb`'s fast processing speed, this step would take too long to be evaluated at run time. Therefore, we suggest that the process of matching the patterns with the current set of log files should be done periodically and also does not necessarily need to be performed by the system running the web server itself.

Third, `flexiweb` applies rewrite rules while visitors are browsing the web site. In order to seamlessly fit in with the web site's design, web designers need to mark, using standard HTML tags, where `flexiweb` should place its links.

5. PATTERN LANGUAGE

As demonstrated in Section 3, `flexiweb` uses a pattern language to express its optimizations. Administrators can apply these optimizations to simplify their web site. However, our system is not limited to the application of a few pre-defined optimizations. Instead, administrators can introduce new optimizations as they recognize new opportunities. To introduce a new optimization, an administrator first expresses it in our pattern language and adds it to the set of optimizations that `flexiweb` should apply. We now show how to formalize the patterns shown in Section 3.

Each optimization pattern of `flexiweb` has two parts: match and rewrite. Section 5.1 describes the match part and Section 5.2 describes the rewrite part.

5.1 Match Block

A match specification describes the situations in which the optimization applies. Each match specification in `flexiweb` contains a single expression. When applying an optimization, `flexiweb` evaluates this expression and, possibly, its subexpressions. The value(s) returned from the overall expression is the result of the match.

5.1.1 Datatypes and Operators

`flexiweb` supports multiple datatypes. Apart from standard types such as integers and strings, our set of datatypes includes *User*, *Page*, *PageVisit*, *Edge*, and *Path*. We also support sets and lists of these types. A *User* is a unique visitor to our website. A user can visit a *Page*. Each page has a URL. A *PageVisit* represents a visit to a page by a user and thus each *PageVisit* has a *Page*, a *User*, and a time. An *Edge* links two *PageVisits* and represents a single edge traversal by a *User*. A *Path* is a sequence of *PageVisits* representing a traversal from when user arrives at a website to when the user stops traversing (i.e., the user stays at the last *Page*). Sometimes when convenient, we will think of a *Path* as a sequence of *Edges* rather than a sequence of *PageVisits*.

`flexiweb` supports a number of simple operators for computing with the above datatypes. These operators include standard arithmetic operators such as +, , and /, string concatenation, and set operations (unions and intersection). Each datatype supports appropriate operators that return information about its properties. For example, *p.length* returns the length of a path *p*. *p*[*i*] returns the i^{th} element of path *p*. Other operators on our data types include the combination of two pages to an edge (**makeEdge**(...)), the construction of a path from a single edge (**makePath**(...)), and the concatenation of two paths (**appendPaths**(...)).

5.1.2 Function definition

To encapsulate and reuse patterns, the language also allows the definition of parameterized functions. These functions can call each other to create recursion for example, and provide return values that can be used by the calling entity.

5.1.3 Iteration

The language described so far cannot describe properties that must hold over multiple pieces of data (e.g., over all paths). To remedy this, `flexiweb` contains the constructs forall and exists that take a set of elements. Both constructs return two values: First, a boolean value and a subset of the set supplied as argument.

forall ⟨ var ⟩ : ⟨ Set ⟩ **in** ⟨ body ⟩
exists ⟨ var ⟩ : ⟨ Set ⟩ **in** ⟨ body ⟩

forall evaluates *Set* and binds each value in the *Set* to *var*. For each such binding, *forall* evaluates body. If all of these evaluations evaluate to true, then the entire *forall* evaluates to true. If, however, even one evaluation of body evaluates to false, the entire forall evaluates to false. If the forall returns true, the second value returned are all values of ⟨ var ⟩. Otherwise the empty set is returned.

For example consider this pattern:
forall p : Path **in** p.length > 10

If all *Paths* have length greater than 10, the above pattern returns true, otherwise it returns false. If the pattern returns true, it will also return all values that *p* had while evaluating the body, i.e., all paths of length 11 or longer.

exists is different from forall in that it evaluates to true if any of the evaluations of *body* evaluates to true. Otherwise it evaluates to false. If exists returns true, it also returns those values of ⟨Set⟩ that caused the exists body to evaluate to true. If the exists returns false, it also returns the empty set. If multiple exists and forall invocations are nested, the outer construct returns the combination of the values of its variable and the values of the variables of the nested construct that correspond to the respective outer value.

For example consider this pattern:
exists p : Path **in** p.length > 10

This pattern evaluates to true if any of the paths have a length greater than 10. If the pattern returns true, it will also return a set of those values for which the body was true, i.e., the set of all paths that have a length greater than 10.

5.1.4 Global Predicates

Although the pattern language we have described so far is quite expressive, it is cumbersome to express global properties (such as the average time spent in a web page). Thus, `flexiweb` includes operations that compute global properties. Examples for global predicates of in `flexiweb` are:

avgtime(pg) returns the average time, in milliseconds, spent in Page *pg* by users that visited *pg*.

outratio($pg_1 \rightarrow pg_2$) returns the ratio of the number of users that left Page pg_1 via an edge to page pg_2 to the number of users that left node pg_1 by any edge. outratio does not count users that do not leave pg_1.

incoming(pg) returns the total number of users that visit page *pg* by an incoming edge.

outgoing(pg) returns the total number of users leaving *pg* by the outgoing edge.

5.1.5 Data constraints

We often need to restrict the data on which our patterns operate. For example, we may be interested only in paths from the last two weeks; Paths on the other hand gives us all paths in the logs. To support such restrictions, pattern writers can place $< \langle Expr \rangle >$ after the predicates or set variables that need to be restricted. The *Expr* specifies the restriction. For example:

User=user Restrict to a specific user.
Time=time Restrict to a particular time interval.

5.2 Rewrite Block

As discussed in Section 5.1, a match block evaluates to true if it finds opportunities for optimizations. In addition, a match block returns bindings for the *exists* and *forall* variables that make the match block evaluate to true. `flexiweb` invokes the rewrite block for each returned binding. The rewrite block can access these bindings. Given this information, the rewrite block actually performs the optimizations. Our current prototype of `flexiweb` optimizes the web graph using *redirects* and *quicklinks*.

redirect($a \rightarrow b$) causes a redirect from page a to page b. More specifically, every time a user reaches a, the web server redirects the user to b. *redirect* adds an appropriate message and link to b (when users reach it via the redirect) so that users can backtrack if they did not want to be redirected.

quicklink($a \rightarrow b$) inserts a quicklink into page a that leads to page b. *quicklink* takes optional attributes, such as a link description that can be set through an attribute *title*. Quicklinks are displayed by default as a hovering element that does not disturb the layout of the web site. The web designer can specify alternative placements. These quicklinks describe shortcuts to pages of interest depending on the current user's position and/or previous navigation path.

6. EXAMPLES

After having presented the core features of our language, we now show how to express the optimizations in Section 3 using our pattern notation was presented in Section 5.

6.1 Irrelevant intermediate nodes

The following pattern implements the optimization in Section 3.1. The match block finds all p and i such that users (i) on average spend little time (i.e., less than t_{time}) on the i^{th} page visited on the path and (ii) on average, when users visit the i^{th} page in p they usually go on to the $i+1^{th}$ page in p. Note the ".page", which extracts the *Page* from the *PageVisit* (recall that a *Path* is a sequence of *PageVisit*s). The rewrite part of the pattern inserts a redirect between the two pages identified in the inner *exists*.

```
match {
        exists p:Path in exists i:0<=i<(p.size - 1) in
             avgtime(p[i].page) < t_time&&
             outratio(p[i] → p[i+1]) > t_freq }

rewrite { redirect(p[i] → p[i+1])}
```

6.2 Skipping irrelevant nodes on a per user basis

This pattern differs from the previous one in that it performs the optimization on a per-user basis. Note how the pattern uses <User=u> to restrict *Path*, *avgtime*, *outratio*, and *redirect* to a particular user.

```
match {
    exists u:User in
        exists p:Path<User=u> in
            exists i:0<=i<(p.size - 1) in
            avgtime<User =u>(p[i].page) < t_time
          && outratio< User =u>(p[i] → p[i+1]) > t_freq}
rewrite {
    redirect< User =u>(p[i] → p[i+1])
}
```

6.3 Providing shortcuts to popular targets

```
match {
exists p:Path in
    exists k:0<=k<p.size in
    (outgoing(p[k].page / incoming(p[k].page)) < t_leaving
    && exists j:0<=j<k in
    forall q:j<=q<=k in
        outratio(p[q].page → p[q+1].page) > t_freq &&
        (incoming(p[k].page) / incoming(p[j].page)) > t_min
}
rewrite {
    forall r:j<=r<k in
       quicklink(p[r].page → p[k].page, title=p[k].page.title)
}
```

This pattern implements the optimization in Section 3.3. ($outgoing(p[k].page)/incoming(p[k].page)$) $< t_{leaving}$ evaluates to true if a significant fraction of visitors to $p[k].page$ actually stay at this page (i.e., they do not leave). *outratio*($p[q].page \rightarrow p[q+1].page) > t_{freq}$ checks that the edges on the subpath from $p[j].page$ to $p[k].page$ are frequently traversed. Finally, $incoming(p[k].page)/incoming(p[j].page) > t_{min}$ guarantees that a significant number of the users that enter $p[j].page$ actually reach $p[k].page$. The rewriting then inserts quicklinks along the paths identified by the match.

6.4 Reacting to temporal phenomena

The pattern in Figure 4 implements the optimization in Section 3.4. This pattern is identical to the pattern in Section 6.3 except that it restricts the pattern to only look at the last two weeks of data.

7. EVALUATION

We evaluate our system using four criteria. First, we show that the idea of web site-independant patterns can be useful. We do this by demonstrating that the patterns have a wide applicability and can be found across different web sites. Second, we demonstrate that our proposed language is expressive enough to model a wide variety of useful patterns. Third, we evaluate the effectiveness of our approach by showing that users adopt quicklinks introduced by our system into their usage behavior. Fourth, we evaluate the resource requirements of our system and show that it is fast enough to be deployed.

```
match {
    exists p:Path<Time=last2weeks> in
    exists k:0<=k<p.size in
        (outgoing<Time=last2weeks>(p[k].page / incoming<Time=last2weeks>(p[k].page)) < t_leaving &&
        exists j:0<=j<k in
            forall q:j<=q<=k in
                outratio<Time=last2weeks>(p[q].page → p[q+1].page) > t_freq &&
                (incoming<Time=last2weeks>(p[k].page) / incoming<Time=last2weeks>(p[j].page)) > t_min }
rewrite {forall r:j<=r<k in
    quicklink(p[r].page → p[k].page, title=p[k].page.title) }
```

Figure 4: The pattern for the temporal phenomena

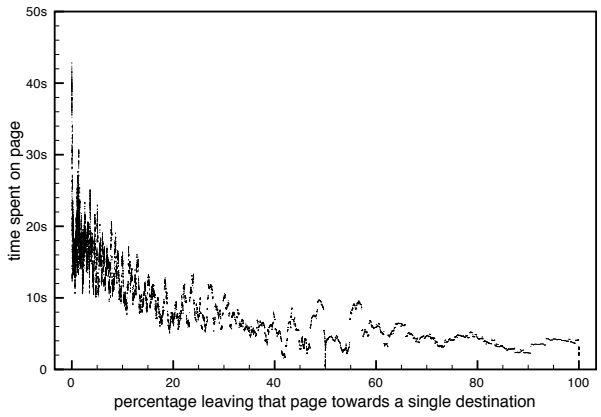

Figure 5: Relationship between time spent on a page and percentage of visitors leaving that page towards a single destination

For this evaluation, we analyzed the visitor behaviors on two departmental web servers over a period of 10 months. These two servers hosted both the official department web pages and 47 personal web sites. Over the course of the evaluation, these servers received more than 65 million page hits.

7.1 Usefulness of Patterns

In this subsection, we show that dynamic patterns are applicable and necessary. We do this by (i) showing that log files have properties that can be detected by patterns; and (ii) that user behavior is not predictable. Property (i) implies that we can apply patterns to web sites, property (ii) implies the need for an automatic application of patterns.

Page content and surfing behavior are related. The core idea of the patterns proposed in Section 3.1 and 3.2 was that some pages may be "gateway" pages with only little information or that ask visitors to make a selection that is obvious to most of them. To show that structure exists that can be detected by our patters, we examine two factors: (i) The usefulness of a page, as measured by the time visitors spent looking at it, and (ii) the ratio of visitors that leave a page towards a given second page. Both properties are easily measured by our patterns.

Figure 5 shows the relationship between the time spent on a given page and where users go from that page. The plotted data is based on an analysis of more than 31000 paths on 48 web sites represented as dots in the graph that were taken by at least 5 users each, and a clear relationship seems to exists between the overall time and the continuing surf behavior of the visitors. The less time a visitor spends on any given page, the more likely the user is to leave that particular page towards a *single* other page. Thus "gateway" pages as targeted by the patterns exist and can be identified.

Time-variant behavior is not predictable. We now show that user behavior changes unpredictably over time. This generates a need for frequent adaptation of quicklinks. Consider the 8-month timeline of visitors accessing three selected web pages on the department's web site in Figure 6. Accesses to the pages "Spring 2007 classes", "Grad Program Admission" and "Fall 2006 classes" vary over time and due to external events. The week before each semester starts, visitors become highly interested in the course listing for the current semester, but the week after the begin of the semester, this flow of visitors ceases rapidly.

A similar temporal pattern exists for the visitors' interest in the admission to the graduate program. Starting in October, the stream of visitors builds up until it highly decreases after the application deadline in the first week of December. For all three web pages, there exists a changing demand over time and during such peak periods one might decide to make the link to that page more visible or place more prominently.

Using the data from Figure 6, we also see that manually "hardcoding" times when quicklinks should be appear on the web site is not feasible: even though registration began on October 17th, we did not see any change in user's behavior until three weeks later. In another instance we observed, a research paper was featured on a government web site and in a news report. Within hours that paper became the second most accessed resource on the entire web server. Due to this unpredictability only a system observing user behavior will be able to make the right changes in a timely and relevant manner. This is the core idea of the patterns proposed in Section 3.3 and 3.4.

7.2 Expressiveness of Language

A pattern language for our application needs to be extensible and expressive enough to model any kind of behavioral pattern and associated optimization. In Section 5 we discussed only the core features of our proposed language, which was designed in a modular fashion. Developers can extend its features very easily using a dedicated API. The extensions can be written in Java.

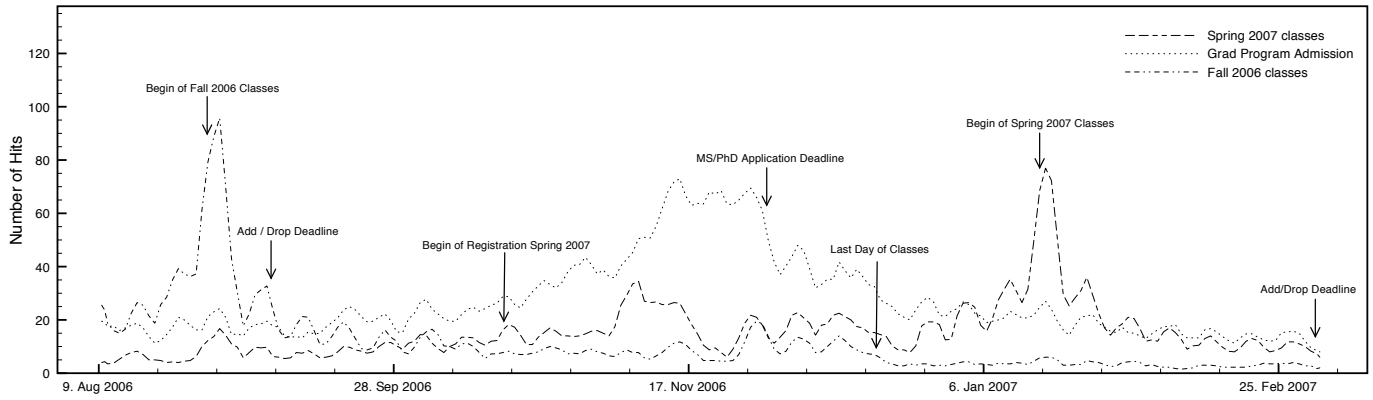

Figure 6: Development of user hits on three selected web pages over time

It is further possible using the semantics of our language to show that it is Turing complete. This guarantees that any possible computation can be modeled with our language. However, we omit this proof due to space constraints in this paper. This can be intuitively understood by using a mapping that translates the infinite tape into a path using recursive functions for the loop and *forall* statements to introduce conditionals.

7.3 Effectiveness of our System

This part of our evaluation addresses the question whether web users experience the introduced quicklinks as a benefit and use them. On the sixth month of our observation, we installed `flexiweb` onto the departmental web site. Every night, the system would analyze the log data of the past days, weight it according to a negative exponential function to account for a trade off between fast responsiveness and hysteresis and identify possible quicklinks to the most popular pages on the web server. To measure the usefulness of the introduced quicklinks, we monitored how user navigation and behavior patterns changed subsequent to placing quicklinks on the web site.

Figure 7 shows the percentage of visitors that use the introduced quicklinks instead of following their established patterns of navigation to reach the page "Current Course Schedule" or pages for the BS and MS programs. As the Figure shows, once the quicklinks were introduced, the visitors recognized their presence and adopted these quicklinks permanently into their usage pattern of this web site.

Visitors going to the "Courses", "BS", and "MS" page however did not adopt quicklinks to the same degree. Even though all three quicklinks were linking to pages that would otherwise have taken two links to get to, thus providing the users the same benefit of saved navigational effort, some links were used more than others.

The hypothesis we had prior to the evaluation of the usefulness of the quicklinks was that visitors coming to the web site more often know what to look for and where to look

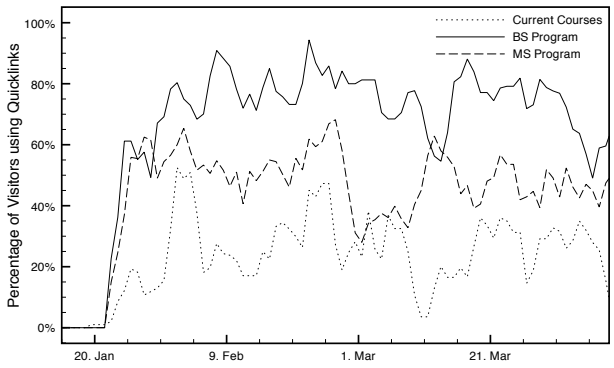

Figure 7: Percentage of all visitors getting to the current course schedule, BS program and MS program using quicklinks

for it. Visitors interested in the current class schedule are more likely to be current students than those accessing the information page on the Bachelor's program and therefore more likely to follow their established navigational behavior and less likely to switch to quicklinks. While this seemed to be true as about twice as many visitors to the BS program used quicklinks than visitors to the courses page, we defer this question which visitors adopt usability improvement under which circumstances to future work. However, we can already conclude that navigational improvements such as a quicklink do offer a benefit to users and change user behavior on web sites.

7.4 Performance of flexiweb

The time required by `flexiweb` to find each pattern depends on the complexity of the pattern matched. When it comes to the pattern "Skipping irrelevant nodes", `flexiweb` took about 2 min 45 seconds on a CoreDuo workstation to process a slice of an Apache log file containing about 2.3 million page hits. The complexity of the other patterns is similar.

8. CONCLUSIONS AND FUTURE WORK

We have introduced the concept of web site-independent user behavior patterns which can be found across web sites and showed examples extracted from an analysis of 48 different web sites totaling more than 65 million hits in 10 months. We have proposed and evaluated a new language in which webmasters can express optimizations for their web sites; each optimization is effectively a rewrite rule that matches user patterns and based on the matches, transforms web pages. Our system, `flexiweb`, implements our language. We have used `flexiweb` in a live deployment and demonstrated that visitors quickly adopt the usage of such optimizations.

There remain research questions to be investigated in future work. For example, we need to understand further how and under which circumstances navigational improvements such as quicklinks are adopted by visitors and how navigational improvements should be designed to facilitate that process. Further, we will need to investigate what is the "right" amount of optimization, as too much quicklinks to "hot" topics will certainly confuse visitors and eventually lead to a degradation of the system. Finally, researchers will have to look for and identify patterns that can be universally applied across web sites. This quest may partially be driven by the body of related work in human cognition and HCI and should eventually provide a set of common improvements that can be used as guidelines for create new web sites and provide enhancements to current ones.

9. REFERENCES

[1] P. D. Bra, A. Aerts, B. Berden, B. de Lange, B. Rousseau, T. Santic, D. Smits, and N. Stash. Aha! the adaptive hypermedia architecture. In *Proceedings of the ACM Hypertext Conference*, 2003.

[2] M.-S. Chen, J. S. Park, and P. S. Yu. Data mining for path traversal patterns in a web environment. In *ICDCS '96: Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS '96)*, page 385, Washington, DC, USA, 1996. IEEE Computer Society.

[3] B. D. Davison. Predicting web actions from html content. In *HYPERTEXT '02: Proceedings of the thirteenth ACM conference on Hypertext and hypermedia*, pages 159–168, New York, NY, USA, 2002. ACM Press.

[4] H. Lieberman. Letizia: An agent that assists web browsing. In C. S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 924–929, Montreal, Quebec, Canada, 1995. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[5] B. Mobasher, N. Jain, E.-H. Han, and J. Srivastava. Web mining: Pattern discovery from world wide web transactions. Technical Report 96-050 (96-050), September 1996.

[6] T. Munzner. Drawing large graphs with h3viewer and site manager. In *GD '98: Proceedings of the 6th International Symposium on Graph Drawing*, pages 384–393, London, UK, 1998. Springer-Verlag.

[7] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu. Mining access patterns efficiently from web logs. In *PADKK '00: Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications*, pages 396–407, London, UK, 2000. Springer-Verlag.

[8] M. Perkowitz and O. Etzioni. Towards adaptive web sites: conceptual framework and case study. *Artificial Intelligence*, 118(1-2):245–275, 2000.

[9] E. Ramp, P. D. Bra, and P. Brusilovsky. High-level translation of adaptive hypermedia applications. In *ACM Hypertext Conference*, 2005.

[10] B. Shneiderman and P. Maes. Direct manipulation vs. interface agents. *interactions*, 4(6):42–61, 1997.

[11] M. Spiliopoulou and L. C. Faulstich. Wum: A tool for web utilization analysis. *Lecture Notes in Computer Science*, 1590:184–203, 1999.

[12] T. Tsandilas and m. c. schraefel. User-controlled link adaptation. In *HYPERTEXT '03: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, pages 152–160, New York, NY, USA, 2003. ACM Press.

[13] A. Wexelblat and P. Maes. Footprints: history-rich tools for information foraging. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 270–277, New York, NY, USA, 1999. ACM Press.

[14] T. W. Yan, M. Jacobsen, H. Garcia-Molina, and U. Dayal. From user access patterns to dynamic hypertext linking. In *Proceedings of the fifth international World Wide Web conference on Computer networks and ISDN systems*, pages 1007–1014, Amsterdam, The Netherlands, The Netherlands, 1996. Elsevier Science Publishers B. V.

[15] E. Zhu. Hypermedia interface design: The effects of number of links and granularity of nodes. *Journal of Educational Multimedia and Hypermedia*, 8(3), 1999.