A Data Miner analyzing the Navigational Behaviour of Web Users

Myra Spiliopoulou[¶] Lukas C. Faulstich[*] [**] Karsten Winkler[‖]

Abstract

Web site design is currently based on thorough investigations about the interests of web site visitors and on less investigated assumptions about their exact behaviour. Concrete knowledge on the way visitors navigate in a web site could prevent disorientation and help owners in placing important information exactly where the visitors look for it.

Our Web Utilization Miner tool can provide such knowledge. The general problem we address is: *Given a number of traversed paths, discover subpaths with structural or statistical properties of interest.* In fact, we anticipate that not all nodes in a subpath are of equal importance. Hence, we allow that subpaths having only some nodes in common be combined into a pattern that shows the desired properties as a whole.

To capture the ambiguous expressions of this problem, we provide a powerful mining language, by which the expert can specify the desired structural and statistical properties of the patterns to be constructed. To efficiently discover paths which, when combined, result in such desirable patterns, we use an innovative technique based on the processing of aggregated sequences. Several optimization steps are performed to further reduce the mining search space.

INTRODUCTION

The emerging need for well-organized sites is common to non-profit institutions and commercial providers of goods. Both are interested in encouraging visitors to access pages they consider important, in exhibiting the links between relevant pages and in preventing disorientation. Data mining is the appropriate methodology for systematically analyzing the behaviour of past visitors and taking decisions on what has to be improved.

Analysis of user behaviour has two aspects, one concerning the interests of the users and the information they access, the other concerning the *way* of accessing this information. The first aspect is addressed by techniques for the establishment of user profiles and is not peculiar to web usage. For instance, student profiles are considered in intelligent tutoring systems. The second aspect is addressed by techniques analysing web server logs.

We argue that those two aspects are complementary, since a web user is characterized by her interests *and* by her navigational behaviour. For example, consider a user that explores the links in a web site to find every bit of information of potential interest and a user that prefers keyword search. Those two users need fundamentally different support, even if both of them are interested in solar energy collectors, chess and medieval sculpture. In this study, we concentrate on the second aspect of user support, namely on the analysis of user navigational behaviour.

There is a plethora of commercial tools performing some basic analysis of web log files. They mostly provide statistical results on traffic load and access to pages or small page sequences. A

[¶]Institut für Wirtschaftsinformatik, Humboldt-Universität zu Berlin, Spandauer Str. 1, D-10178 Berlin. myra@wiwi.hu-berlin.de

[*]Institut für Informatik, Freie Universität Berlin, Takusstr. 9, D-14195 Berlin, Germany. faulstic@inf.fu-berlin.de

[**]Supported by the German Research Society, Berlin-Brandenburg Graduate School on Distributed Information Systems (DFG grant no. GRK 316).

[‖]Institut für Wirtschaftsinformatik, Humboldt-Universität zu Berlin, Spandauer Str. 1, D-10178 Berlin. kwinkler@wiwi.hu-berlin.de

detailed discussion can be found e.g. in [Zaïane et al., 1998], concluding to their inappropriateness for thorough analysis of access patterns.

In this study, we propose the exploitation of mining technology to discover access patterns with "interesting" statistical properties and present our Web Utilization Miner (WUM) designed to this purpose. The problem we aim to resolve is: *Given a number of traversed paths, discover subpaths with structural or statistical properties of interest.* This includes subpaths going through pages with given properties, being traversed by a minimum or maximum number of users and showing a reliable statistical confidence between any two or only some of the pages in the path.

We purposely use terms like "some" and "important" instead of "frequent", to stress the fact that statistical dominance is not always of interest. On the contrary, statistically dominant patterns rarely bring new knowledge to the expert. The characteristics that make a pattern interesting are of more general nature. For instance, the web owner may be interested in paths rarely followed or at paths crossing pages that refer to a common subject. Such paths would give more indications of pitfalls in the design of the site than statistically dominant ones. WUM supports the specification of such generic characteristics and identifies only patterns conforming to them.

The mining model of WUM is innovative in two aspects. First, it anticipates the fact that the "importance" indicators in user behaviour go far beyond than frequent access to some pages. So, it can be instructed to discover patterns of statistical dominance, as conventional miners do, but also supports the specification of subjective criteria. Finally, it can achieve high performance improvements over conventional miners by processing aggregated sequences instead of raw web log data and by applying optimization steps during the mining process.

In the next section, we discuss data preparation in WUM and describe how visitor transactions are modelled as sequences and aggregated into an internal storage structure. Our pattern discovery mechanism is described thereafter. In the sequel, we present the current status of our implementation and its visualization module. We then discuss related work on web usage mining and sequence mining before coming to conclusions and outlook.

AGGREGATION OF THE RECORDED VISITOR DATA

WUM is applied on aggregated data. Hence, after performing the classical preparation steps [Cooley et al., 1997b], we merge the data into an "Aggregated Log", as described below.

Data Preparation

An entry in a web server log contains the timestamp of a traversal from a source to a target page, the IP address of the originating host, the type of access (GET or POST) and other data. Many entries are considered uninteresting for mining and removed. This filtering is application dependent. However, in most cases (and in WUM) accesses to images are filtered out.

The remaining entries must be grouped by the visitor that performed them. Cookies or similar mechanisms can be exploited to identify a visitor a priori. Alternatively, we can make a posteriori assumptions on how close in time two consecutive entries might be or what pages should they refer to in order to come from the same person. An investigation on such approaches can be found in [Cooley et al., 1997b]. In WUM, we currently assume that consecutive accesses from the same host during a certain time interval come from the same user. However, a more sophisticated mechanism can be used instead, without affecting the subsequent steps of data preparation.

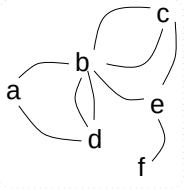
Once we assess the originator of each entry, we group consecutive entries to a user session or "transaction". Different grouping criteria are modelled and compared in [Cooley et al., 1997b]. We support two criteria: A new session starts when (i) the duration of the whole group of traversals exceeds a time threshold, similarly to [Cooley et al., 1997b], or (ii) the elapsed time between two consecutive traversals exceeds a threshold. Dissimilarly to [Cooley et al., 1997b], [Chen et al., 1996], we do not observe a backward traversal as the start of a new session, because it may be part of a guided or explorative tour [Tauscher and Greenberg, 1997].

Transforming Transactions to Trails

A transaction output at the end of the preparation phase is a group of entries. Unnecessary data have been removed, so that only the timestamp, the source and the target page are retained. We now compress each group into the sequence of accessed pages. We call this sequence a "trail" or "path".

A set of visitor transactions produces a *collection* (multiset) of trails, since multiple visitors may access the same pages in the same order. We group them into subcollections of identical trails. The "traffic" of a trail is then the cardinality of the subcollection to which it belongs.

Example 1: Let a, b, c, d, e, f be the pages of a web site and $T1$ - $T4$ be transactions on it, with entries of the form `(timestamp, sourcePage, targetPage)`.



A tiny web site

$T1$	$T2$	$T3$	$T4$
$(11{:}01, a, b)$	$(13{:}15, a, b)$	$(11{:}02, b, d)$	$(11{:}47, a, b)$
$(11{:}03, b, e)$	$(13{:}18, b, e)$	$(11{:}03, d, b)$	$(11{:}49, b, e)$
		$(11{:}04, b, e)$	$(11{:}50, e, f)$
$< a, b, e >$	$< a, b, e >$	$< b, d, b, e >$	$< a, b, e, f >$

The traffic of trail $< a, b, e >$ is 2, while the traffic of $< b, d, b, e >$ and of $< a, b, e, f >$ is 1. Note that the trail $< b, c, b, e >$ contains a circle; page b has been visited twice. △

Aggregating Trails

In Example 1, we see that some trails have common prefixes. For instance, trail $< a, b, e, f >$ occurs only once, but its prefix $< a, b, e >$ has been accessed 3 times. If we merge such trails together, we can put more statistical information in a compact storage structure.

An "aggregate tree" is a tree constructed by merging trails with the same prefix. A tree node corresponds to a page in a trail and *is annotated with the number of visitors having reached this page across the same trail prefix*. This value is the "support" of a node, computed in the context of the node's predecessors up to the aggregate tree's root. The traffic of a trail is then implemented as the support of the tree leaf corresponding to the last node of the trail.

Example 2: We now extend our previous example to model a larger number of trails into aggregate trees. At the left side of Fig. 1 we show the visitor trails with their traffic.

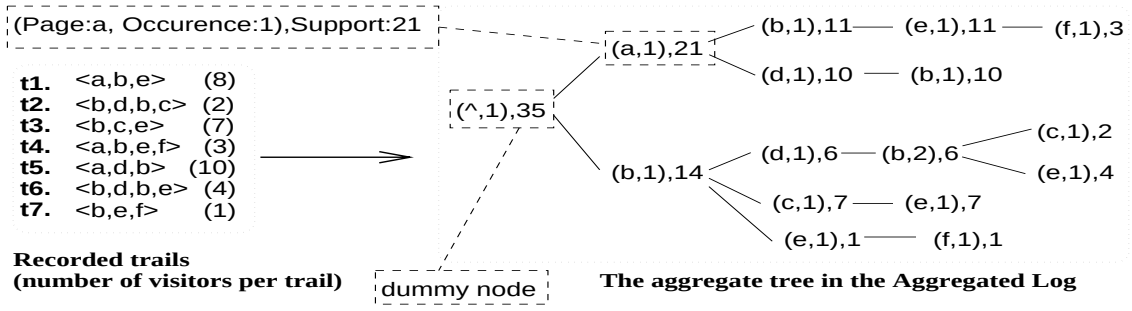


Figure 1: Constructing aggregate trees

b is the first page of trails $t2, t3, t6$ and $t7$. $(b, 1)$ indicates the first occurence of b; $(b, 2)$ denotes a revisit in $t2, t6$. We retain page occurences explicitly on reasons of efficiency.

By adding up the visitors of trails $t2, t3, t6, t7$, we compute 14 as the support of $(b, 1)$. Similarly, the support of $(a, 1)$ is 21. In $t1, t4$, page b was visited after a; the respective aggregate tree node has a support of 11. The root node ^ is a dummy one, needed to merge all trails into one aggregate tree. Its support is the total number of recorded traversals.

Note that the trails starting at a cannot be merged with those starting at b. If we are interested in the number of visitors having traversed $< b, e, f >$ *irrespectively of their starting node*, we encounter 4 visitors, 3 of which started at a before going to b. To build such a pattern we must merge subbranches. This is done by the miner module described in the next section. △

Aggregate Trees in Persistent Storage

WUM is not applied to the collection of trails but to the large aggregate tree, into which they are merged. We refer to this tree as the "Aggregated Log" hereafter. Clearly, a trail that occurs ν times, appears in the Aggregated Log only once. Moreover, for any two or more trails with common prefix, the prefix is stored in the Aggregated Log only once, the supports of its nodes adjusted respectively. This does not only reduce the storage requirements. It also accelerates the mining procedure by reducing the size of the input log.

The Aggregated Log can be built once from a web log or be expanded periodically by the new web log entries. In the latter case, the trails are formed as described above and merged into a small aggregate tree. This tree is then merged with the existing Aggregated Log by merging branches with common prefix, adjusting the supports of the common prefix nodes and extending the branches with new subbranches to accommodate the non-common nodes.

PATTERN DISCOVERY OVER AGGREGATED DATA

The Aggregated Log contains statistically and structurally aggregated information on web traversals in a tree form. The navigation patterns to be discovered are themselves graph structures. Hence, instead of invoking a mechanism similar to those proposed in [Srikant and Agrawal, 1996], [Mannila and Toivonen, 1996], we build a new mechanism discovering, matching and merging subbranches of the Aggregated Log into a navigation pattern.

A *"navigation pattern"* is a generalization of the aggregate tree introduced in the previous section. In particular, it is a graph built according to a pattern descriptor. A "pattern descriptor" is a sequence of identifiers and wildcards, where an identifier refers to an occurence of a web page. The complete formalism on navigation patterns can be found in [Spiliopoulou, 1999].

The navigation pattern is built by discovering the tree branches in the Aggregated Log that conform to the pattern descriptor and merging them at (i) the common prefixes and (ii) the nodes bearing the identifiers referenced in the descriptor. The supports of those new nodes are computed by adding up the supports of the merged nodes; the supports of the other nodes remain intact.

Example 3: Let the aggregate tree of Fig. 1 be our Aggregated Log. Let $< (b, 1), *, (e, 1) >$ be a pattern descriptor. It specifies that we select the (sub)branches containing the first occurence of page b, i.e. $(b, 1)$, and then look among them for those containing also a first occurence of e, i.e. $(e, 1)$. There may be any number of nodes in-between those two. In the Aggregated Log shown at the left side of Fig. 2, we have marked the subbranches thus selected. All but one of them also contain $(e, 1)$.

The navigation pattern is built by copying those branches into a new graph and merging: (i) all nodes referring to $(b, 1)$; (ii) all nodes referring to $(e, 1)$ *preceded by* $(b, 1)$, so that the order specified by the descriptor is preserved; (iii) all nodes belonging to common prefixes between the two nodes (none in this example). Then, we compute the support of each such new node by adding the supports of the merged nodes.

In Fig. 2, we see how the navigation pattern on the right is produced from the Aggregated Log via an intermediate structure. This structure consists of the subbranches conforming to the pattern descriptor. The nodes to be merged are marked. △

The reader might ask whether the nodes corresponding to the wildcards of the pattern descriptor should be part of the navigation pattern. Indeed, maintaining those nodes implies a more complex theoretical framework than in other sequence miners [Agrawal and Srikant, 1995],

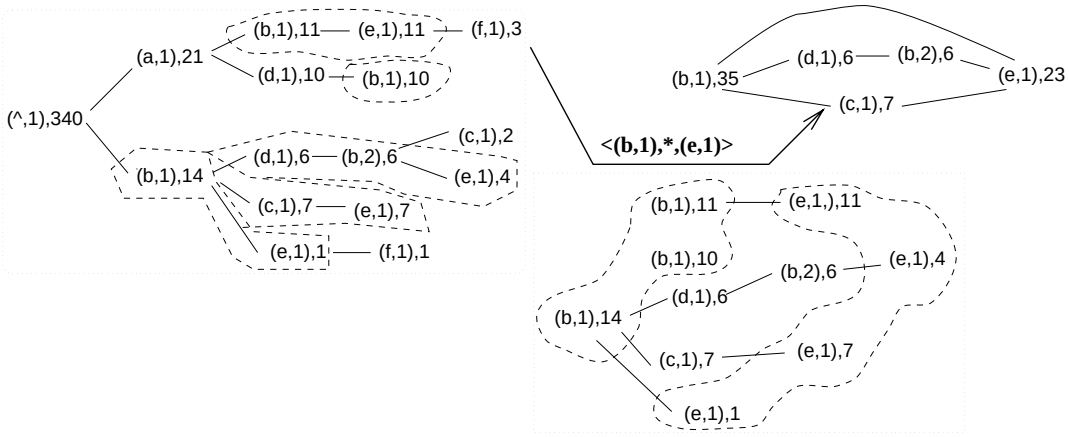


Figure 2: Constructing navigation patterns from the Aggregated Log

[Mannila and Toivonen, 1996], [Srikant and Agrawal, 1996], [Zaki, 1997] and an increase in processing cost. However, we argue that the complete navigation pattern bears invaluable information for the mining expert and the web site designer. The example query of Fig. 4 in the section on the current WUM prototype is intended to depict this

Which Navigation Patterns are Important to Discover?

We have defined a navigation pattern as a graph built according to a pattern descriptor. Obviously, the patterns to be discovered must be described according to more general criteria. In particular, we need a way of specifying the "interestingness" (term in [Piateski-Shapiro and Matheus, 1994]) of navigation patterns, as subjectively conceived by the mining expert.

We suggest that, informally, "interestingness" is a specification concerning (A) the content, (B) the structure and (C) the statistics of navigation patterns. Given an "interestingness descriptor", we must build all conformant navigation patterns by assigning appropriate values to all components of the statement not explicitly specified.

In WUM, an "interestingness descriptor" is a *query* in our mining language, MINT described in [Spiliopoulou and Faulstich, 1998]. MINT supports predicates of type (A) on the properties of web pages (currently, URL string only) and on their occurences, of type (B) on the relative positions of the pages in the pattern, as dictated by a "template", and of type (C) on the statistics of the pattern nodes.

Example 4: A simple query on web usage would be: *Where do visitors accessing pages of a certain group usually go afterwards?* Or equivalently: *Given a certain group of already visited pages, what are the pages preferred thereafter and how are they reached?*

This statement produces a "template" `B*X`, where `B` is a variable standing for the pages in the group and `X` for the preferred pages. If preference is conceived as statistical probability of at least 60%, a predicate `X.support / B.support >= 0.6` is needed.

The expert might further require that the members of the `B` group should have been accessed a significant number of times in this pattern, e.g. 25., i.e. `B.support >= 25`.

Finally, if only the first access is of interest for `B`, the occurence number of `B` must be 1, i.e. `B.occurence=1`. The complete query **Q1** in MINT is:

```
select T
from node as B X,
template B*X as T
and B.support >= 25
and B.occurence = 1
and X.support / B.support >= 0.6
```

In the next subsection, we will show how this query is executed by our miner against the Aggregated Log of Fig. 1. △

What is the difference between the above query and the association rule $B \rightarrow X$ with confidence 0.8 and support of B at least 25? First of all, association rules have no order. To see the impact of this, consider the rule $b \rightarrow a$. We can see from Fig. 1 that the first occurence of b has a support of 35. Moreover, the 21 visitors to a also visited b across two different paths. For an association rule, this is equivalent to saying that 21 of the visitors of b also visited a. In fact, the probability $P(a|b) = 21/35 = 0.6$. Thus, the association rule $b \rightarrow a$ is true with confidence 0.6, although nobody followed a path from b to a.

Second, the output of a MINT query is a navigation pattern, in which also the intermediate nodes are contained. Hence, the expert can observe the common and different parts of the paths between the nodes of interest.

More examples on the formulation of MINT queries, as well as the MINT syntax, can be found in [Spiliopoulou and Faulstich, 1998]. Note that despite the simplicity of those examples, there is no limitation on the number of variables in the template, nor on the predicates combining their statistical characteristics.

Processing a Mining Query

To process queries in MINT, we distinguish the three types of predicates, as described in the subsection "Which navigation patterns are Important to Discover?". Recall that type A predicates refer to invariant properties of the web pages in the log, type B predicates are dictated by the query template, and type C are predicates on statistics.

A simplified version of our mining algorithm in pseudocode is shown in Fig. 3 for a generic template $< *, v_1, *, \ldots, *, v_k, * >$. A faster but more resource consuming variation is proposed in [Spiliopoulou, 1999]. Obviously, the algorithm does not change if some wildcards are omitted. We explain those steps below.

1. Generating the set of all descriptors. In the first step of our algorithm, we traverse the Aggregated Log and generate the candidate descriptors by binding the template variables. These descriptors involve pages and occurence numbers that satisfy the type A and type B predicates.

Example 5: To explain the first step of our algorithm, we consider the following query **Q2**:

```
select T
nodes as X Y, template XY as T
where X.url = "M.html"
and X.support >= 100
and Y.occurence = 2
```

Q2 finds all patterns of two adjacent nodes, the first of which refers to the page "M.html" and has a support of at least 100, while the second one refers to the second occurence of some page. Note that **Q2** is purposely very restrictive, so that we can show how the first step of our algorithm works.

Let $< xywz >$ be a branch in the Aggregated Log, such that $x = (("M.html", 1), 20)$, $y = (("Z.html", 1), 10)$, $w = (("M.html", 2), 9)$ and $z = (("Z.html", 2), 8)$. There are 16 possible descriptors. From the predicate on the url of X, we see that X can only be assigned to x or w. Y can only be assigned to z because of the type B predicate on the occurence number. This leaves the variable bindings $< x, z >$, $< w, z >$. The first one violates the template, because the two variables must be in adjacent positions. Hence, there is only one possible binding of variables, $< w, z >$. △

One might object that most predicates given by the expert are of statistical nature, i.e. of type C. However, we can still use C predicates to generate type A *constraints*: if the support of a node should be at least 100, a page visited less than 100 times in total can be safely ignored. By keeping this property, called accesses, along with each web page, we can efficiently skip uninteresting pages.

Input: Template $< *, v_1, *, v_2, \ldots, v_k, * >$ and predicates of type A, B, C
Output: A set of navigation patterns.

1. *Generate the set of* All_Descriptors *by traversing the Aggregated Log:*
   (a) *For each order-preserving sequence of nodes* $< n_1, *, \ldots, *, n_k >$ *in a branch produce the descriptor* $d =< *, d_1, *, \ldots, *, d_k, * >$, *where* $d_i = (n_i.page, n_i.occurence)$.
   (b) **if** d *is already in* All_Descriptors, **then** *skip it.*
   (c) **else if** *for all* $i = 1, \ldots, k$:
      i. *The web page referred to in* n_i *satisfies the type A predicates for variable* v_i.
      ii. *The position of* n_i *in the sequence is allowed by the template.*
      iii. *The occurence number in* n_i *is permitted for* v_i.

      then *add* d *to* All_Descriptors.
2. *Construct the navigation pattern for each descriptor* d *in* All_Descriptors:
   (a) *Compare* d *with the (sub)descriptors already in the set* Tested_Descriptors *and test if it can be rejected without building the navigation pattern.*
   (b) *If* d *is not rejected, construct the navigation pattern for it:*
      i. *Find all branches of the Aggregated Log that conform to* d.
      ii. *Merge at each element of* d.
      iii. *Compute the supports of the nodes produced by merging.*
      iv. *Test the C predicates against the navigation pattern.*
      v. *If* d *is rejected*
         - **then** *store the smallest prefix that caused the rejection in the set* Tested_Descriptors, *marking it as* R(ejected).
         - **else** *store* d *in* Tested_Descriptors, *marking it as* S(uccessful).
   (c) *If* d *is not rejected,* **then** *output its navigation pattern.*

Figure 3: The mining algorithm of WUM

Example 4 cntd. We can see the impact of this optimization in the processing of query **Q1** on the Aggregated Log of Fig. 1: Since the support of `B` should be at least 25, we can add the constraint `B.accesses >= 25` in the original query.

Now, assuming that a, b both refer to middleware, we can reject $(a, 1)$ without further testing, because the value of its `accesses` is 21 only. Hence, the descriptors generated in the first step have the form $< (b, 1), *, X >$, where `X` is one of $(e, 1), (f, 1), (d, 1), (b, 2), (c, 1)$. △

2. Constructing the navigation pattern of each descriptor In the next step of our algorithm, we build the navigation pattern of each descriptor. Actually, this step can be combined with the previous one to test the descriptors as they are built. We make the separation to simplify the analysis.

The first and last actions of this phase are optimizations. In particular, we keep all descriptors produced thus far in a set Tested_Descriptors, marking them as Successful or Rejected.

1. *Already computed support values:*
   Once a descriptor is computed, we know the support values of all its elements. If a new descriptor has the same values for the first i variables with an already computed one, then we already know their support values.

2. *Additional type A predicates:*
   By knowing the support values of the first i descriptor elements, we can assess constraints on the minimum total number of accesses for the values of the subsequent elements.

3. *Rejected subdescriptors:*
   Assume a descriptor is rejected because its i^{th} element failed in a comparison with the statistics of the j^{th} element (for $i > j$). Then we say that the i^{th} element caused the rejection.

   No descriptor with the same first i elements can satisfy the query. By keeping (only) the subdescriptor with those i elements and comparing it to each new descriptor, we can prune out unsuccessful descriptors without building their navigation patterns.

Example 4 contd. After building all descriptors for query **Q1**, we build the navigation pattern for the first one, say $< (b,1), *, (f,1) >$. This descriptor is added to the Tested_Bindings as R(ejected). The rejection was caused by the second element, the support of which was too low.

The next descriptor has the form $< (b,1), *, X >$ for some other value of `X`. From the contents of Tested_Bindings, we assess that:

1. `B.support = 35`
2. If `X` is equal to $(f,1)$, the descriptor should be rejected.
3. The value of `X.accesses` should be no lower than `B.support * 0.6`, i.e. 21.

The last observation allows us to reject all bindings of `X` but $(e,1)$ without building their navigation patterns.

The careful reader may observe that by a template of the form `B*X`, the values of `B.support` and `B.accesses` are always equal. Hence, we could prune out all candidates but $(e,1)$ at step 1. This is indeed so for the first variable of any template and only for it. We have ignored this fact on purpose though, in order to explain the optimization steps possible at step 2, without introducing a more complex template. △

Test the C predicates [Step 2.(b)iv]. Type C predicates are evaluated after constructing the navigation pattern. This has the disadvantage of producing many navigation patterns, whose statistics do not satisfy the query predicates. However, the optimization steps described above show how we can exploit those predicates: if part of a navigation pattern does not satisfy a type C predicate, then all other navigation patterns having this same part will also fail and can be pruned without testing.

THE CURRENT WUM PROTOTYPE

The current version of WUM prototype is implemented in Java^{TM}. A graphical user interface is available for entering queries in MINT and for presenting the results.

At the left side of Fig. 4, we show a MINT query executed against a web log of one week, kindly provided by a german company for testing. The query finds patterns leading to (and going beyond) the page with the contact persons of the company. Only patterns starting at a node with support at least 40 are of interest. One URL is explicitly excluded.

The query produces two navigation patterns, of which we show one at the right side of Fig. 4. The upper window shows one part of the pattern, namely `X*Y`, while the lower window shows the second part `Y*`. Our visualization module currently displays patterns as trees; this is why `X*Y` is a tree, all leaf nodes of which refer to the same page. This page is the value bound to the variable `Y`.

```
select T
from node as X Y, template X*Y* as T
where X.url != "/balk.html"
and X.support > 40
and Y.url = "/kontakt.html"
```

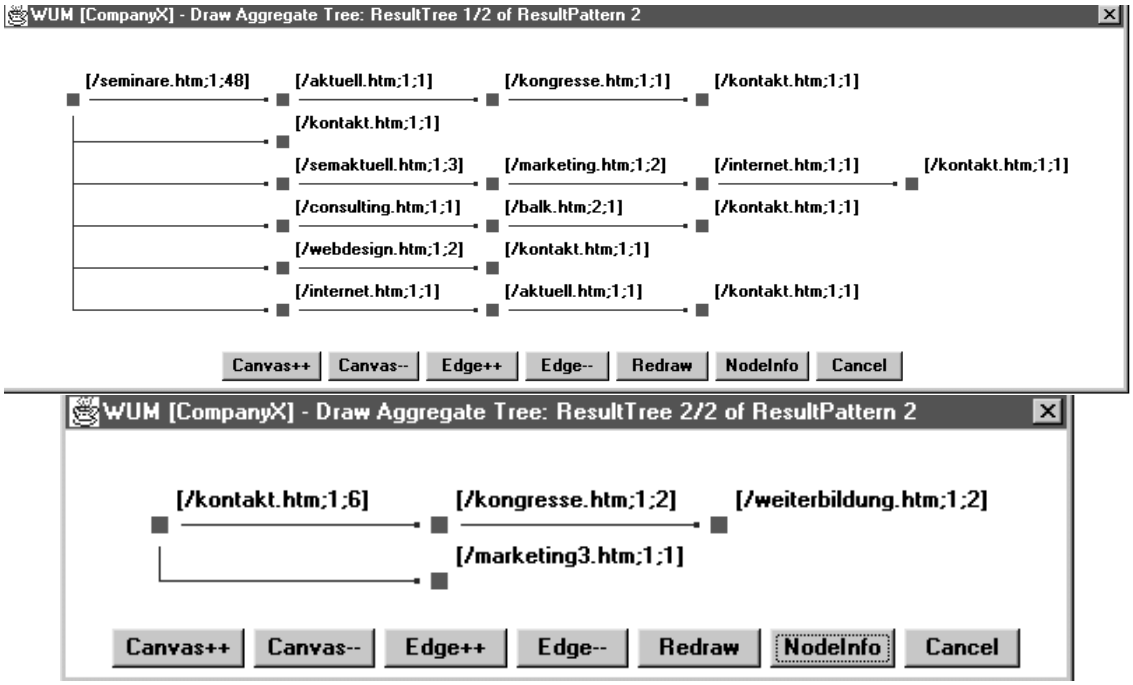


Figure 4: Presentation of navigation patterns

The displayed navigation pattern shows one of the (two) patterns. In this pattern, people reaching the contacts-page go through the page on the offered seminars. Although there is a link from this page to the contacts-page, most people choose different links, but eventually reach the contacts-page. The second pattern indicates that 50% of the visitors stop at this page, while the others continue navigation. Interpretation of those trends requires knowledge of the contents of all pages appearing in the patterns.

This example reveals why it is important to display the whole navigation pattern, instead of the frequent nodes in it only. The first pattern leads from the seminars-page to the contacts-page. Since this pattern is frequent and since there is a link from the former page to the latter, the web page designer would erroneously assume that the traversal is made by means of the link. Thus, a design pitfall (e.g. a misleadingly labelled link) would remain undiscovered.

The suspicious designer would discover the pitfall by issuing the above query *and* a similar one, with the template set to $X[1;\infty]Y*$ instead. This would imply that at least one web page is accessed between `X` and `Y`. The less suspicious designer might skip this test, though. The goal of WUM is to help the application expert in discovering knowledge with the least effort.

RELATED WORK

Mining on the navigational behaviour of web users has gained significant importance in the last years, as the value of the web for commerce and for information dissemination has been recognized.

The web miner proposed in [Chen et al., 1996] focusses only on statistically dominant paths using a methodology for the discovery association rules.

The "web log miner" of [Zaïane et al., 1998] uses OLAP technology for prediction, classification and time-series analysis of web log data. Interesting results are obtained on web traffic analysis and on the evolution of user behaviour (e.g. preferred pages) over time. The orthogonal issue of assessing the users' behaviour and organizing the web site in conformance with it, is left open.

The "PageGather" tool of [Perkowitz and Etzioni, 1998] uses a clustering methodology to discover web pages visited together and to place them in the same group. This work concentrates on

the discovery of pages that are of potential interest to the same group of users. The issue of how those pages were reached is not being considered.

The "Footprints" tool of [Wexelblat, 1996] records the footprints left behind by web site visitors and accumulates them into frequently accessed paths. The new visitor of the web site can profit from the recorded behaviour of previous users.

The "WEBMINER" of [Cooley et al., 1997a] provides a query language, with which the user can initiate searches for paths conforming to more sophisticated criteria than high frequency of access. However, the miners invoked to process such queries have not been designed to cope with such criteria: According to [Cooley et al., 1997a], the miner for association rules and the miner for sequential access patterns incorporated to the WEBMINER are conventional tools, of which the former is slightly customized to improve its performance. The disadvantage of this approach is the necessity of a postprocessing module, which computes the information required to verify the criteria of the language not supported by the miner itself.

In particular, conventional sequence miners are designed and optimized for the discovery of *frequent* sequences, i.e. sequences appearing more often than a threshold [Agrawal and Srikant, 1995], [Mannila and Toivonen, 1996], [Srikant and Agrawal, 1996], [Zaki, 1997]. There is no trivial way of instructing such a miner to find rare paths of high confidence. In [Zaki et al., 1998], the approach proposed to cope with this problem was the gradual removal of frequent but uninteresting sequences in a loop of mining and post-mining sessions. WUM provides instead an elegant query language for specifying the frequence/rarity thresholds to be satisfied by the mining results.

All miners described above process the raw data in the web log file. Recently, a data aggregation method has been proposed in [Amir et al., 1997] and exploited for the discovery of association rules. They observe the transactions recorded in the original dataset as sequences of items according to a some arbitrary order. They aggregate them into a trie structure, so that sequences with common prefixes are merged together. Then, a rule $A \rightarrow B$ holds iff events A, B appear in the same branch of the tree. Since association rules have no ordering, they will appear in exactly one aggregated sequence. The efficiency of this approach has been demonstrated e.g. in [Feldman et al., 1997].

We have independently come to the idea of aggregating web paths: since paths are already sequences, this is intuitive. Dissimilarly to association rules, though, paths have an order. So, two events A, B may appear in several aggregated sequences at any positions. This makes mining over aggregated sequences more complicated than the discovery of association rules, as could be seen from the description of the WUM algorithm.

CONCLUSIONS

We have presented a mining mechanism for the analysis of the navigational behaviour of web users. Our miner is part of the WUM tool for the specification, discovery and visualization of interesting patterns. Its first aim is flexibility towards application-dependent and even pattern-dependent interestingness criteria.

To cope with very generic and with complicated specifications of the desired patterns, WUM uses an efficient discovery mechanism. By mining on aggregated instead of raw data we ensure a performance improvement, since the size of the dataset to be processed is reduced considerably. In our description of the mining mechanism, we have shown that further performance gains are achieved by generating constraints from the interestingness criteria specified by the expert.

We are currently working on formalizing the optimization mechanism and on the establishment of index structures to further increase efficiency. To determine the accuracy of predictions on navigational behaviour, we perform experiments on web logs of our site. A concrete goal is to improve the design of the lecture notes' pages we provide.

WUM has been originally designed for the discovery of navigation patterns in web sites. However, its miner is appropriate to discover patterns over sequences of any type. Hence, we are particularly interested in testing WUM for arbitrary event patterns from other application areas.

References

[Agrawal and Srikant, 1995] Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In *ICDE*, Taipei, Taiwan.

[Amir et al., 1997] Amir, A., Feldman, R., and Kashi, R. (1997). A new and versatile method for association generation. *Information Systems*, 22:333–347.

[Chen et al., 1996] Chen, M.-S., Park, J. S., and Yu, P. S. (1996). Data mining for path traversal patterns in a web environment. In *ICDCS*, pages 385–392.

[Cooley et al., 1997a] Cooley, R., Mobasher, B., and Srivastava, J. (1997a). Grouping web page references into transactions for mining world wide web browsing patterns. Technical Report TR 97-021, Dept. of Computer Science, Univ. of Minnesota, Minneapolis, USA.

[Cooley et al., 1997b] Cooley, R., Mobasher, B., and Srivastava, J. (1997b). Web mining: Information and pattern discovery on the world wide web. In *ICTAI'97*.

[Feldman et al., 1997] Feldman, R., Klösgen, W., and Zilberstein, A. (1997). Visualization techniques to explore data mining results for document collections. In *KDD'97*, pages 16–23, Newport Beach, CA. AAAI Press.

[Mannila and Toivonen, 1996] Mannila, H. and Toivonen, H. (1996). Discovering generalized episodes using minimal occurences. In *KDD'96*, pages 146–151.

[Perkowitz and Etzioni, 1998] Perkowitz, M. and Etzioni, O. (1998). Adaptive web pages: Automatically synthesizing web pages. In *submitted to AAAI'98*.

[Piateski-Shapiro and Matheus, 1994] Piateski-Shapiro, G. and Matheus, C. J. (1994). The interestingness of deviations. In *AAAI'94 Workshop Knowledge Discocery in Databases*, pages 25–36. AAAI Press.

[Spiliopoulou, 1999] Spiliopoulou, M. (1999). The laborious way from data mining to web mining. *Int. Journal of Comp. Sys., Sci. & Eng., Special Issue on "Semantics of the Web"*. to appear.

[Spiliopoulou and Faulstich, 1998] Spiliopoulou, M. and Faulstich, L. C. (1998). WUM: A Tool for Web Utilization Analysis. In *EDBT Workshop WebDB'98*, Valencia, Spain. Springer Verlag. extended version to appear in LNCS 1590.

[Srikant and Agrawal, 1996] Srikant, R. and Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In *EDBT*, Avignon, France.

[Tauscher and Greenberg, 1997] Tauscher, L. and Greenberg, S. (1997). Revisitation patterns in world wide web navigation. In *CHI'97*, Atlanta, Georgia.

[Wexelblat, 1996] Wexelblat, A. (1996). An environment for aiding information-browsing tasks. In *Proc. of AAAI Spring Symposium on Acquisition, Learning and Demonstration: Automating Tasks for Users*, Birmingham, UK. AAAI Press.

[Zaïane et al., 1998] Zaïane, O., Xin, M., and Han, J. (1998). Discovering web access patterns and trends by applying OLAP and data mining technology on web logs. In *Advances in Digital Libraries*, pages 19–29, Santa Barbara, CA.

[Zaki, 1997] Zaki, M. J. (1997). Fast mining of sequential patterns in very large databases. Technical Report 668, University of Rochester.

[Zaki et al., 1998] Zaki, M. J., Lesh, N., and Ogihara, M. (1998). PLANMINE: Sequence mining for plan failures. In Agrawal, R., Stolorz, P., and Piatesky-Shapiro, G., editors, *Proc. of 4th Int. Conf. KDD*, pages 369–373, New York, NY.