Mining Web Logs to Improve Website Organization

Ramakrishnan Srikant
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120

Yinghui Yang*
Dept. of Operations & Information Management
Wharton Business School
University of Pennsylvania
3620 Locust Walk, Suite 1300
Philadelphia, PA 19104

ABSTRACT

Many websites have a hierarchical organization of content. This organization may be quite different from the organization expected by visitors to the website. In particular, it is often unclear where a specific document is located. In this paper, we propose an algorithm to automatically find pages in a website whose location is different from where visitors expect to find them. The key insight is that visitors will backtrack if they do not find the information where they expect it: the point from where they backtrack is the expected location for the page. We present an algorithm for discovering such expected locations that can handle page caching by the browser. Expected locations with a significant number of hits are then presented to the website administrator. We also present algorithms for selecting expected locations (for adding navigation links) to optimize the benefit to the website or the visitor. We ran our algorithm on the Wharton business school website and found that even on this small website, there were many pages with expected locations different from their actual location.

1. INTRODUCTION

Consider the quandary of visitors to Yahoo!: will a listing of computer stores be in the "Business & Economy" directory (on Yahoo!) or in the "Computers & Internet" directory? In this case, Yahoo! has a link from both locations. However, in general, it is hard to organize a website such that pages are located where visitors expect to find them. This problem occurs across all kinds of websites, including B2C shops, B2B marketplaces, corporate websites and content websites.

We propose a novel algorithm to solve this problem by automatically discovering all pages in a website whose location is different from the location where visitors expect to find them. The key insight is that visitors will backtrack if they do not find the page where they expect it: the point from where they backtrack is the expected location for the page. Expected locations with a significant number of hits are presented to the website administrator for adding navigation links from the expected location to the target page. We also present algorithms for selecting the set of navigation links to optimize the benefit to the website or the visitor, taking into account that users might try multiple expected locations for a target page.

Paper Organization We discuss related work in Section 1.1. We describe the algorithm for finding expected locations in Section 2, and show how to optimize the set of navigation links in Section 3. We present results on the Wharton website in Section 4, and conclude in Section 5.

1.1 Related Work

There has been considerable work on mining web logs; however, none of them include the idea of using backtracks to find expected locations of web pages.

Perkowitz et al. [4] [5] investigate the problem of index page synthesis, which is the automatic creation of pages that facilitate a visitor's navigation of a website. By analyzing the web log, their cluster mining algorithm finds collections of pages that tend to co-occur in visits and puts them under one topic. They then generate index pages consisting of links to pages pertaining to a particular topic.

Nakayama et al. [2] also try to discover the gap between the website designer's expectations and visitor behavior. Their approach uses the inter-page conceptual relevance to estimate the former, and the inter-page access co-occurrence to estimate the latter. They focus on website design improvement by using multiple regression to predict hyperlink traversal frequency from page layout features.

Spiliopoulou et al. [7] [8] propose a "web utilization miner" (WUM) to find interesting navigation patterns. The interestingness criteria for navigation patterns are dynamically specified by the human expert using WUM's mining language which supports the specification of statistical, structural and textual criteria.

Other related work includes the following: Chen et al. [1] present an algorithm for converting the original sequence of log data into a set of maximal forward references and filtering out the effect of some backward references which are mainly made for ease of traveling. Pei et al. [3] propose a novel data structure, called Web ac-

*This work was partly done while the author was at IBM Almaden.

WWW10, May 1-5, 2001, Hong Kong.
ACM 1-58113-348-0/01/0005.

cess pattern tree for efficient mining of access patterns from pieces of logs. Shahabi et al. [6] capture the client's selected links, page order, page viewing time, and cache references. The information is then utilized by a knowledge discovery technique to cluster visitors with similar interests.

2. FINDING EXPECTED LOCATIONS

We describe our model of visitor search patterns in Section 2.1, and discuss the problem of identifying target pages in Section 2.2. Section 2.3 gives the algorithm for finding expected locations. We discuss the limitations of our approach in Section 2.4. Throughout this section, we use "search" to denote that the visitor is browsing for a specific page or set of pages, and do not imply the use of a search engine.

2.1 Model of Visitor Search Patterns

Single Target Consider the case where the visitor is looking for a single specific target page T. We expect the visitor to execute the following search strategy:

1. Start from the root.
2. While (current location C is not the target page T) do
   (a) If any of the links from C seem likely to lead to T, follow the link that appears most likely to lead to T.
   (b) Else, either backtrack and go to the parent of C with some (unknown) probability, or give up with some probability.

Note that when the visitor reaches the same page again in step 2(a), she will follow a different link since the estimates of whether a link is likely to lead to will have been updated.

Set of Targets Now consider the scenario where the visitor wants to find a set of target pages $T_1, T_2, \ldots, T_n$. The search pattern is similar, except that after finding (or giving up on) T_i, the visitor then starts looking for T_{i+1}:

1. For i := 1 to n
   (a) If $i = 1$, start from the root, else from the current location C.
   (b) While (current location C is not the target page T_i) do
   - If any of the links from C seem likely to lead to T_i, follow the link that appears most likely to lead to T_i.
   - Else, either backtrack and go to the parent of C with some probability, or give up on T_i and start looking for T_{i+1} at step 1(a) with some probability.

In this scenario, it may be hard to distinguish the target pages from the other pages by simply looking at the web log. We discuss this issue after first giving an example.

Example 1 Figure 1 shows a hierarchically organized website, and a traversal path {1A, 2A, 3A, 2A, 3B, 2A, 1A, 2C, 9}. If pages are cached in the browser, the web log will only contain {1A, 2A, 3A, 3B, 2C, 9}, not the entire path. We give three possible interpretations of this path:

- The visitor is looking for a single target page 9. She starts at the root, expects to find page 9 under 2A, goes to 2A and then 3A, realizes that 9 is not under 3A, backtracks to 2A, tries 3B, realizes that 3B does not contain 9 either, backtracks all the way to the root, and finally finds 9 under 2C. The expected locations for target page 9 are 3A and 3B, and the actual location is 2C.
- The visitor is looking for the target pages {3A, 9}. She starts at the root, expects to find 3A under 2A, and successfully reaches 3A. She then starts looking for 9. She goes back to 2A, tries 3B, backtracks to the root, and then finds 9 under 2C. The expected location for the target page 9 is 3B, and the actual location is again 2C.
- The set of target pages was {3A, 3B, 9}. In this case, all the target pages were located where the visitor expected them.

Notice that it is not easy to figure out which is the correct interpretation, *unless* 3A and 3B are index or navigation pages, while 9 is a content page. In this case, the first interpretation is the correct one, since 3A and 3B cannot be target pages.

2.2 Identifying Target Pages

For some websites like Amazon and Ebay, there is a clear separation between content pages and index (or navigation) pages; product pages on these websites are content pages, and category pages are index or navigation pages. In such cases, we can consider the target pages for a visitor to be exactly the set of content pages requested by the visitor. Other websites such as information portals or corporate websites may not have a clear separation between content and index pages. For example, Yahoo! lists websites on the internal nodes of its hierarchy, not just on the leaf nodes. In this case, we can use a time threshold to distinguish whether or not a page is a target page. Pages where the visitor spent more time than the threshold are considered target pages. We can also combine these two methods, and have different time thresholds for different classes of pages.

2.3 Algorithm

If there is no browser caching, it is conceptually trivial to find a backtrack point: it is simply the page where the previous and next pages in the web log (for this visitor) are the same. The HTTP standard states that the browser should not request the page again when using the browser's history mechanism. In practice, some browsers use the cached page when the visitor hits the "back" button, while others incorrectly request the page again. It is possible to disable caching by setting an expiration date (in the meta tag in the page), but this can significantly increase the load on the website.

Rather than rely on disabling browser caching, we use the fact that if there is no link between pages P_1 and P_2, the visitor must have hit the "back" button in the browser to go from P_1 to P_2. Thus to find backtrack points, we need to check if there is a link between two successive pages in the web log. We build a hash table of the

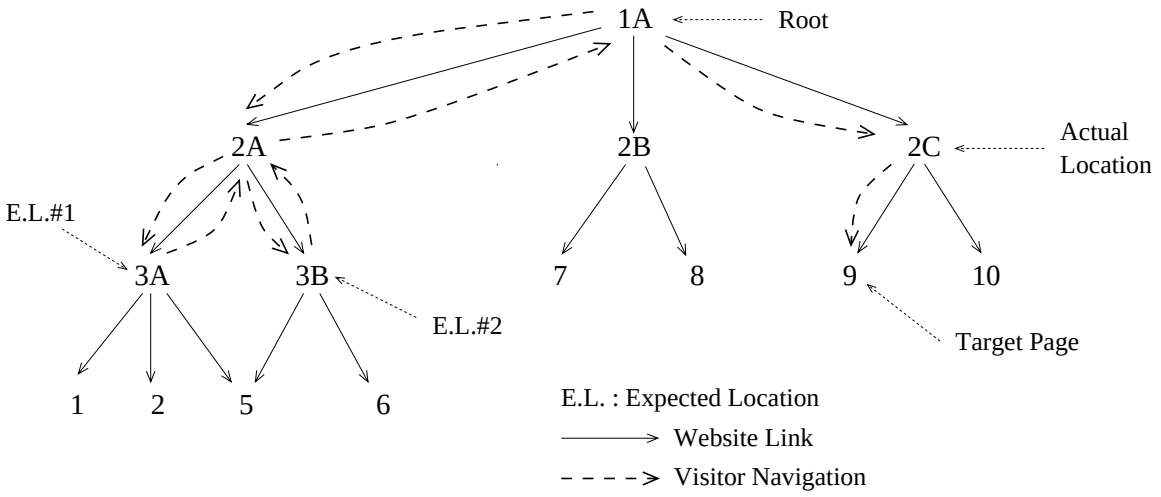

Figure 1: Website & Search Pattern

1. Build hash table of links in the website.
2. Partition web log by visitor:
   - Sort the web log file by visitor ID as the primary key and time as the secondary key, or
   - Partition the web log file by hashing on visitor ID and sort each partition separately, or
   - Scan web log and extract sequence of pages for each visitor ID, passing them onto step 2.
3. For each visitor, partition web log such that each subsequence terminates in a target page.
4. For each visitor and target page, find any expected locations for that target page:

Let $\{P_1, P_2, \ldots, P_n\}$ be the set of visited pages, where P_n is a target page.
Let $B := \phi$ denote the list of backtrack pages.
a) `for` i := 2 `to` $n-2$ `begin`
b) `if` (($P_{i-1} = P_{i+1}$) `or` (no link from P_i to P_{i+1}))
c) Add P_i to B. // P_i is a backtrack point.
`end`
`if` (B not empty)
Add $\langle P_n, B, P_{n-1} \rangle$ to ⟨current URL, backtrack list, Actual Location⟩ table;

Figure 2: Algorithm: Find Expected Location

edges in the website to efficiently check if there is a link from one page to another.

Figure 2 shows the algorithm. In Step 1, we build the hash table. We partition the web log by visitor in Step 2. In Step 3, we split the sequence of accesses for each visitor by the target pages they visit. We assume that the website administrator either specifies the set of possible target pages, or specifies a time threshold to distinguish between target pages and other pages. In Step 4, we find all expected locations (if any) for that target page, and add it to a table for use by the next step of the algorithm. The detection of backtracks occurs in Step 4(b). In addition to checking for the absence of a link from the current to the next page, we also check if the previous and next pages are the same. The latter check takes care of the case where visitors use a navigation link to go to the previous page instead of using the "back" button in the browser.

2.4 Limitations

As we discussed in Example 1 and in Section 2.2, it can be hard to distinguish between target pages and other pages when the website does not have a clear separation between content and index pages. Hence the algorithm may generate false expected locations if it treats target pages as backtrack points, and may miss expected locations if it treats backtrack points as target pages. Increasing the time threshold will result in fewer missed expected locations at the cost of more false expected locations, while decreasing the threshold will have the opposite effect. Hence for websites without a clear separation between content and navigation, the administrator will have to spend some time to determine a good value for the time threshold, as well as sift through the discovered expected locations to drop any false patterns.

Another limitation is that only people who successfully find a target page will generate an expected location for that page. We cannot track people who tried the expected location and gave up after not finding the target page.

3. OPTIMIZING THE SET OF NAVIGATION LINKS

We consider three approaches for recommending additional links to the web site administrator (or automatically adding links):

1. **FirstOnly**: Recommend all the pages whose frequency of occurrence in the first expected location is above an administrator-specified threshold.

2. **OptimizeBenefit**: Recommend the set of pages that optimize benefit to the website, where benefit is estimated based on the fraction of people who might give up on not finding a page.

3. **OptimizeTime**: Recommend the set of pages that minimize the number of times the visitor has to backtrack, i.e., the number of times the visitor does not find the page in an expected location.

The three approaches can generate very different answers, as we illustrate in Examples 2 through 4.

From the algorithm in Figure 2, we get a table with the following columns: $\hat{T}$, $\hat{E}_1$, $\hat{E}_2$, ..., $\hat{E}_n$, $\hat{A}$, where $\hat{T}$ contains the target page, $\hat{A}$ the actual location (parent) of the target page, and $\hat{E}_1$ through $\hat{E}_n$ the first n expected locations for the page for a specific visitor. Note that only $\hat{E}_1$ is guaranteed to be non-empty: if the visitor found the page after the kth expected location, $\hat{E}_{k+1}$ through $\hat{E}_n$ will be empty in the corresponding record.

For ease of exposition, we assume that the set of records has been partitioned based on the value of $\hat{T}$. Let $T_1, T_2, \ldots, T_r$ denote the r unique target pages (in the $\hat{T}$ column). Then the algorithms in 3.1 through 3.3 will be called once for each T_i, on the set of records where value($\hat{T}$) equals T_i.

Count the support for all the pages in $\hat{E}_1$.
Sort the pages by their support.
Present all pages whose support is greater than or equal to S to the administrator.

Figure 3: Algorithm: FirstOnly

3.1 FirstOnly

Figure 3 describes the FirstOnly algorithm. This algorithm recommends the frequent first expected locations (the pages that occur frequently in $\hat{E}_1$) to the website administrator, ignoring any second or subsequent expected locations the visitor may have considered.

We define the support for a page as the count of the page. The administrator specifies a minimum support S for the first choice for the expected location. All expected locations whose support is greater than or equal to S are presented to the administrator.

Example 2 We will use the following example to illustrate the differences between the three approaches. There are five visitors who generated expected locations for the target page T_1. For instance, record 1 says that the visitor first looked at page P1, backtracked and tried P2 before finally locating T_1 in AL.

	$\hat{T}$	$\hat{E}_1, \hat{E}_2, \hat{E}_3, \hat{E}_4$	$\hat{A}$
Record 1	T_1	P1, P2	AL
Record 2	T_1	P1	AL
Record 3	T_1	P2, P3, P4, P1	AL
Record 4	T_1	P3, P2	AL
Record 5	T_1	P4, P2	AL

Let $S = 2$. With FirstOnly, we only look at the frequency in $\hat{E}_1$. Hence P1 is the only recommendation, with a count of 2.

3.2 OptimizeBenefit

Let b_k represent the benefit to the website of visitors finding their target web page in their kth attempt, i.e., in the kth expected location. For instance, b_k could represent the fraction of visitors who give up if they do not find their target page in their kth attempt. We can then order pages based on the benefit to the website of not losing a visitor. Let S_b be the minimum benefit threshold specified by the website administrator.

Figure 4 describes the OptimizeBenefit algorithm. This is a greedy algorithm that attempts to maximize the benefit to the website of adding additional links. In each pass, it finds the page with the maximum benefit, adds it to the set of recommendations, nulls out

// b_k: benefit of finding the target page in the kth expected location.
repeat
  foreach record begin
    for j := 1 to n
      Increment support of value($\hat{E}_j$) by b_j.
  end
  Sort pages by support.
  P := Page with highest support (break ties at random).
  if support(P) $\geq S_b$ begin
    Add $\langle P$, support(P)$\rangle$ to list of recommended pages.
    foreach record begin
      for k := 1 to n begin
        if value($\hat{E}_k$) = P
          Set $\hat{E}_k, \hat{E}_{k+1}, \ldots, \hat{E}_n$ to null;
      end
    end
  end
until (support(P) $< S_b$);

Figure 4: Algorithm: OptimizeBenefit

repeat
  foreach record begin
    Let m be the number of expected locations in this record.
    for j := 1 to m
      Increment support of value($\hat{E}_j$) by $m+1-j$.
  end
  Sort pages by support.
  P := Page with highest support (break ties at random).
  if support(P) $\geq S_t$ begin
    Add $\langle P$, support(P)$\rangle$ to list of recommended pages.
    foreach record begin
      for k := 1 to n begin
        if value($\hat{E}_k$) = P
          Set $\hat{E}_k, \hat{E}_{k+1}, \ldots, \hat{E}_n$ to null;
      end
    end
  end
until (support(P) $< S_t$);

Figure 5: Algorithm: OptimizeTime

all instances of this page and succeeding pages, and recomputes the benefit.

Example 3 We use the same table from Example 2:

	$\hat{T}$	$\hat{E}_1, \hat{E}_2, \hat{E}_3, \hat{E}_4$	$\hat{A}$
Record 1	T_1	P1, P2	AL
Record 2	T_1	P1	AL
Record 3	T_1	P2, P3, P4, P1	AL
Record 4	T_1	P3, P2	AL
Record 5	T_1	P4, P2	AL

Let $b_1 = 1$, $b_2 = 0.5$, $b_3 = b_4 = 0.25$, and $S_b = 2$. Then benefit(P2) = $1 \times 1 + 3 \times 0.5$ = 2.5, benefit(P1) = 2.25, benefit(P3) = 1.5 and benefit(P4) = 1.25. We add P2 to the list of recommendations, and drop P2 and all items that follow P2, resulting in the following table:

	$\hat{T}$	$\hat{E}_1, \hat{E}_2, \hat{E}_3$	$\hat{A}$
Record 1	T_1	P1	AL
Record 2	T_1	P1	AL
Record 3	T_1		AL
Record 4	T_1	P3	AL
Record 5	T_1	P4	AL

In the next pass, benefit(P1) = 2, benefit(P3) = 1, and benefit(P4) = 1. We add $\langle$P1,2$\rangle$ to the table.[1] At this point, none of the remaining pages have support above S_b. Hence P1 and P2 are the only two recommendations.

3.3 OptimizeTime

The goal of this algorithm is to minimize the number of backtracks the visitor has to make. (We use the number of backtracks as a proxy for the search time.) Let S_t denote the threshold for the number of backtracks we must save in order to add a link. Figure 5 describes the OptimizeTime algorithm. This algorithm also does a greedy search, and is quite similar to the OptimizeBenefit algorithm, except for how we count support.

Example 4 We apply the algorithm to the dataset from Example 2:

	$\hat{T}$	$\hat{E}_1, \hat{E}_2, \hat{E}_3, \hat{E}_4$	$\hat{A}$
Record 1	T_1	P1, P2	AL
Record 2	T_1	P1	AL
Record 3	T_1	P2, P3, P4, P1	AL
Record 4	T_1	P3, P2	AL
Record 5	T_1	P4, P2	AL

Let $S_t = 4$. Then TimeSaved(P2) = $1 + 4 + 1 + 1$ = 7, TimeSaved(P1) = 4, TimeSaved(P3) = 4, and TimeSaved(P4) = 3. We add $\langle$P2,7$\rangle$ to the list of recommendations, and drop all items that follow P2, resulting in the following table:

	$\hat{T}$	$\hat{E}_1, \hat{E}_2, \hat{E}_3$	$\hat{A}$
Record 1	T_1	P1	AL
Record 2	T_1	P1	AL
Record 3	T_1		AL
Record 4	T_1	P3	AL
Record 5	T_1	P4	AL

TimeSaved(P1) = 2, TimeSaved(P3) = 1, and TimeSaved(P3) = 1. Hence P2 is the only recommendation.

4. EXPERIMENTS

We used the web server log from http://www.wharton.upenn.edu (Wharton Business School, University of Pennsylvania) to evaluate our algorithm. Figure 6 shows the structure of the website. There are 7 levels in this website hierarchy. Starting with the root as level 1, there are 7 directories (interior nodes in the hierarchy) in level 2, 20 directories in level 3, 21 directories in level 4, 13 directories in level 5, 2 directories in level 6 and 2 directories in level 7. The

[1] Notice that the benefit of P2 would have been 2 and not 2.5 if we had added P1 first. We could optionally go back and recount supports to avoid this double counting, but we do not expect this to make a significant difference to the final answer.

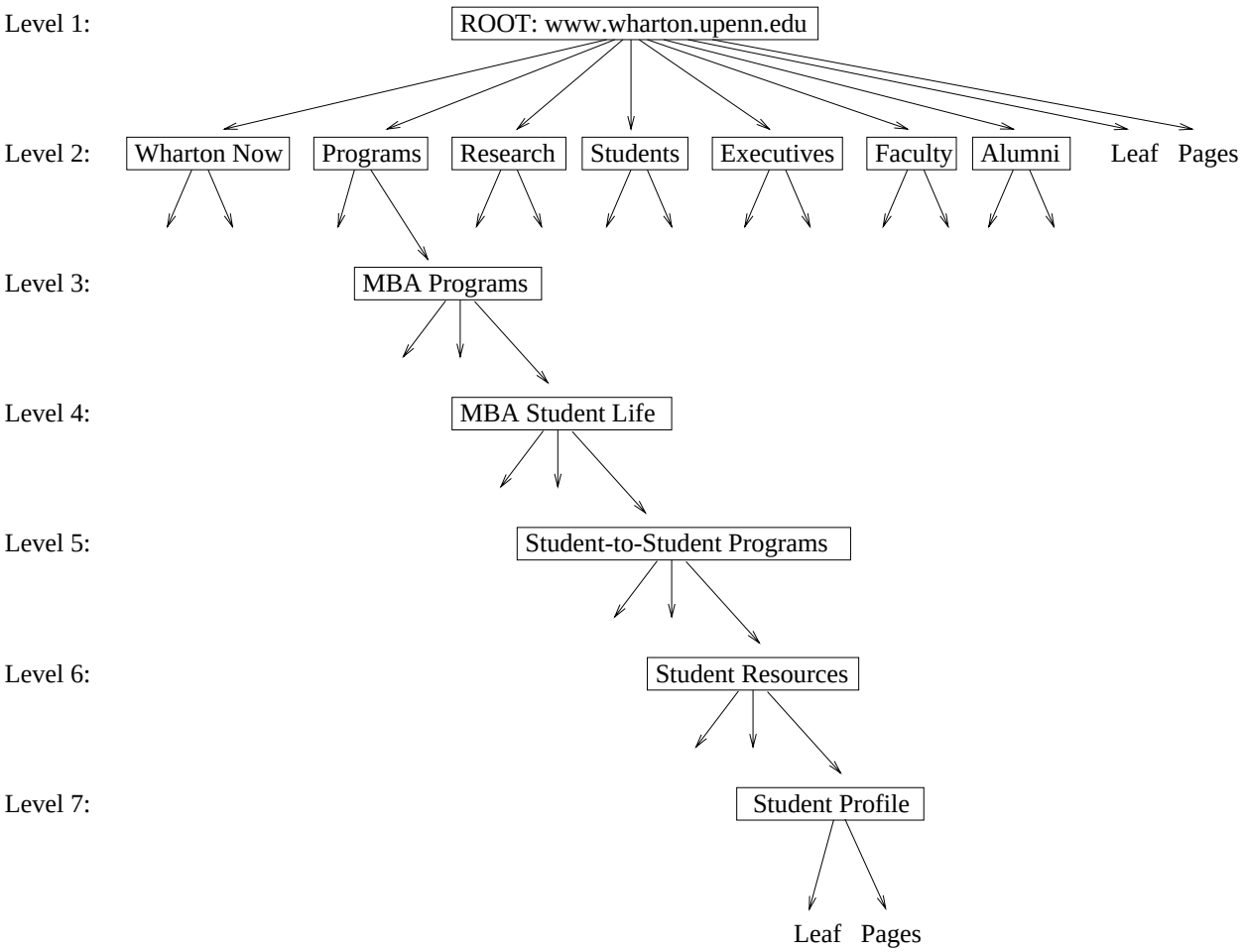

Figure 6: Wharton Website Structure

website has 240 leaf pages. We defined any leaf pages in the log to be a target page, and did not use a time threshold. We made some judgment calls about what was a leaf page versus a directory. For instance, we treated the index of faculty web pages as a leaf page, since our primary interest was in the structure of the Wharton site rather than individual faculty member pages.

We obtained a web log covering 6 days. There were 15,575 unique visitors, and 2,988,206 records (hits) in the log. For each hit, we extracted the IP address, time, and URL from the log. A sample record might then be:

202.104.29.xxx, 00:15:10, /mba/admissions/index.html

If the page the visitor requested contains images, a record is generated for each image. We dropped all image request records, since they are not relevant to our algorithm. In addition, two other Wharton sites, http://inside.wharton.upenn.edu and http://spike.wharton.upenn.edu/spike6/interface, are independent of http://www.wharton.upenn.edu and have their own structure. So we also dropped records for pages in these two sites. This left us with 102,540 records.

According to our experimental results, 25 leaf pages (out of a total of 240) have expected locations different from their actual location at a support threshold of 5. Figure 7 shows five examples from this set. "Support" refers to the number of visitors who expected to find the page at the given expected location, while "Total Hits" is the total number of hits for the leaf page. While some of the supports may seem low (relative to the hits), only people who successfully found the page contribute to the support. We cannot track people who tried the expected location and gave up after not finding the leaf page.

Example W1 (in Figure 7) is perfect: it's obvious that the page should be relocated or an additional link added. Examples W2 and W3 are also straightforward: clearly the target page could be in either location. Examples W4 and W5 are more ambiguous, and we present two possible explanations:

- Visitors may be looking at the student pages, and then backtracking to look at the curriculum.
- Visitors may really expect to find "Curriculum" under "Students" rather than "Programs".

This brings us back to the issue we discussed in Section 2.4: without a clear separation between content and navigation, it can be difficult to differentiate between people browsing a set of target pages versus people who are searching for a single target page. Recall that we had proposed using a time threshold to distinguish between

Example W1	Target page:	http://www.wharton.upenn.edu/mba/s2s/why_wharton.html
	Actual Location:	http://www.wharton.upenn.edu/mba/s2s/s2s.html
	Expected Location:	http://www.wharton.upenn.edu/mba/s2s/s_qa.html
	Support:	6
	Total Hits:	81
	Explanation:	Visitors expect to find the answer to "Why choose Wharton?" in the "Student-to-Student Program's Question & Answer Session" directory instead of the "Student-to-Student Program's General Description" directory.
Example W2	Target page:	http://www.wharton.upenn.edu/mba/admissions/profile.html
	Actual Location:	http://www.wharton.upenn.edu/mba/admissions/index.html
	Expected Location:	http://www.wharton.upenn.edu/students.html
	Support:	6
	Total Hits:	882
	Explanation:	Visitors expect to find "MBA Student Profiles" under "Student" instead of "MBA Admission".
Example W3	Target page:	http://www.wharton.upenn.edu/whartonnow/calendars.html
	Actual Location:	http://www.wharton.upenn.edu/whartonnow.html
	Expected Location:	http://www.wharton.upenn.edu/programs.html
	Support:	7
	Total Hits:	292
	Explanation:	Visitors expect to find "Calendar" under "Programs" instead of the "WhartonNow" directory.
Example W4	Target page:	http://www.wharton.upenn.edu/undergrad/curriculum/concentrations.html
	Actual Location:	http://www.wharton.upenn.edu/undergrad/curriculum/index.html
	Expected Location:	http://www.wharton.upenn.edu/students.html
	Support:	6
	Total Hits:	293
	Explanation:	Visitors expect to find "Curriculum" under "Students" instead of "Programs".
Example W5	Target page:	http://www.wharton.upenn.edu/mba/curriculum/curriculum.html
	Actual Location:	http://www.wharton.upenn.edu/mba/curriculum/index.html
	Expected Location:	http://www.wharton.upenn.edu/students.html
	Support:	6
	Total Hits:	555
	Explanation:	Visitors expect to find "Curriculum" under "Students" instead of "Programs".

Figure 7: Examples from the Wharton Website Results

the two cases. For W4, the 6 visitors spent 3, 120, 25, 21, 4, and 11 seconds on the students.html page. For W5, they spent 5, 4, 68, 55, 19, and 35 seconds. With a time threshold of 30 seconds, the support for W4 would drop to 5, i.e., the algorithm would consider students.html to be a backtrack point in 5 of the 6 cases, and as a target page in the remaining case. Similarly, the support for W5 would drop to 3. Notice that it is hard to get the time threshold exactly right. A time of less than 10 seconds clearly indicates that the page is a backtrack point, while a time of more than 1 minute clearly indicates that the page is a target page. Any value between 10 seconds and 1 minute would be a reasonable choice for the threshold (for this domain), and the administrator can choose a value in this range based on whether she wants to bias the algorithm toward missed expected locations or false expected locations.

The Wharton website is well structured, and many leaf pages are put under multiple directories (as the website designer took into consideration the different expectations of different visitors), yet we still discovered many missed expectations by analyzing the actual visitor traversal pattern. We expect our algorithm to work even better on more structured websites, such as e-commerce websites that have product hierarchies.

5. SUMMARY AND FUTURE DIRECTIONS

We proposed a novel algorithm to automatically discover pages in a website whose location is different from where visitors expect to find them. This problem of matching website organization with visitor expectations is pervasive across most websites.

Our key insight is that visitors will backtrack if they do not find information where they expect it. The point from where they backtrack is the expected locations for the page. We presented an algorithm for discovering such backtracks that also handles browser caching, and discussed the limitations of our approach. We also presented algorithms that select the set of navigation links (to add to expected locations) to optimize visitor time or benefit to the website. We applied our algorithm on the Wharton website, and found many pages that were located differently from where visitors expected to find them.

An interesting problem for future research is that in websites without a clear separation of content and navigation, it can be hard to differentiate between visitors who backtrack because they are browsing a set of target pages, and visitors who backtrack because they are searching for a single target page. While we have proposed using a time threshold to distinguish between the two activities, it will be interesting to explore if there are better approaches to solve this problem.

Acknowledgments We would like to thank Rakesh Agrawal, Roberto Bayardo and Balaji Padmanabhan for their comments and suggestions.

6. REFERENCES

[1] M.-S. Chen, J. S. Park, and P. S. Yu. Data mining for path traversal patterns in a web environment. In *Proc. of the 16th International Conference on Distributed Computing Systems*, pages 385–392, May 1996.

[2] T. Nakayama, H. Kato, and Y. Yamane. Discovering the gap between web site designers' expectations and users' behavior. In *Proc. of the Ninth Int'l World Wide Web Conference*, Amsterdam, May 2000.

[3] J. Pei, J. Han, B. Mortazavi-asl, and H. Zhu. Mining access patterns efficiently from web logs. In *Proc. of the 4th Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, pages 396–407, April 2000.

[4] M. Perkowitz and O. Etzioni. Adaptive web sites: Automatically synthesizing web pages. In *Proc. of the Fifteenth National Conf. on Artificial Intelligence (AAAI)*, pages 727–732, 1998.

[5] M. Perkowitz and O. Etzioni. Towards adaptive sites: Conceptual framework and case study. In *Proc. of the Eighth Int'l World Wide Web Conf*, Toronto, Canada, May 1999.

[6] C. Shahabi, A. M. Zarkesh, J. Abidi, and V. Shah. Knowledge discovery from users web-page naviagtion. In *Proc. of the 7th IEEE Intl. Workshop on Research Issues in Data Engineering (RIDE)*, pages 20–29, 1997.

[7] M. Spiliopoulou and L. C. Faulstich. Wum: A web utilization miner. In *Proc. of EDBT Workshop WebDB98*, Valencia, Spain, March 1998.

[8] M. Spiliopoulou, L. C. Faulstich, and K. Wilkler. A data miner analyzing the navigational behaviour of web users. In *Proc. of the Workshop on Machine Learning in User Modelling of the ACAI99*, Greece, July 1999.