User Browsing Behavior-driven Web Crawling

Minghai Liu[†‡]*, Rui Cai[†], Ming Zhang[‡], and Lei Zhang[†]
[†]Microsoft Research Asia [‡]School of EECS, Peking University
[†]{ruicai, leizhang}@microsoft.com [‡]{lmh, mzhang}@net.pku.edu.cn

ABSTRACT

To optimize the performance of web crawlers, various measures of page importance have been studied to select and order URLs in crawling. Most sophisticated measures (*e.g. breadth-first* and *PageRank*) are based on link structure. In this paper, we treat the problem from another perspective and propose to directly measure page importance through mining user interest and behaviors from web browse logs. Unlike most existing approaches which work on single URL, in this paper, both the log mining and the crawl ordering are performed at the granularity of URL pattern. The proposed URL pattern-based crawl orderings are capable to properly predict the importance of newly created (unseen) URLs. Promising experimental results proved the feasibility of our approach.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*information filtering*

General Terms

Algorithms, Performance, Experimentation.

Keywords

web crawling, URL pattern discovery, web log mining

1. INTRODUCTION

A web crawler cannot download everything but should be selective. The strategy of a crawler to choose URLs from its crawling queue is called crawl ordering policy [14]. The essence of designing crawl ordering policy is to quantitatively measure the "importance" of a page.

In the past decade, a series of ordering policies have been proposed, based on various importance measures. Most popular measures are based on link structure. For example, the breadth-first policy [19] believes pages located within a few links to a website's portal are important. More sophisticated measures like in-degree [7], PageRank [15], and their derivatives [2, 3, 8, 9] take more complicated link structures into account to estimate page importance. Besides link structure, some new measures were proposed recently to leverage other information. For example, topical crawler [13,18] (also called focused crawler [6]) was introduced to prioritize pages according to their content relevance to some pre-defined semantic topics. And in [16, 17], URLs with higher search impact (in terms of click-through rates) are prompted in crawling. Another noticeable trend is to design site-specific importance measures. For example, the structure-driven approach [21] is interested in selecting some certain kinds of target pages in a website; and for forum crawling, only pages containing user-generated content are considered to be important [5,22]. In the literature, promising results have been reported using all these approaches. Some of them (*e.g.* PageRank) have been proven to work well in industry [4].

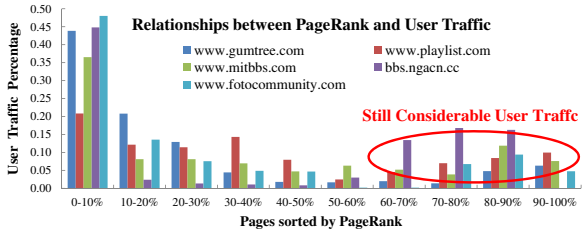

Figure 1: The relationships between PageRank and user traffic distribution on several example websites.

In nature, these importance measures adopt different hypotheses to predict how a page could be interesting to users [19]. All these hypotheses are reasonable but just characterize user interest indirectly and incompletely. Fig. 1 shows that even the most successful PageRank still lose considerable user traffic. This is because user interest is influenced by multifarious factors when the Web becomes more dynamic and complex. Therefore, we propose in this paper to directly measure page importance through mining user interest and behaviors from web browse logs.

Simply prioritizing a URL according to its frequency being recorded is impractical as the log data is very sparse in comparison with the Web. In addition, user behaviors on a single URL are usually noisy and unstable. We propose to summarize log data with URL patterns, and design crawl ordering policies at pattern-level. The motivation is that in a website, URLs generally follow some syntax schemes (patterns) pre-defined by the designers, and URLs belonging to the same pattern usually act similar roles [12]. Moreover, pattern-level statistics are more reliable and robust. In practice, our approach has two additional advantages:

*This work was performed at Microsoft Research Asia.

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

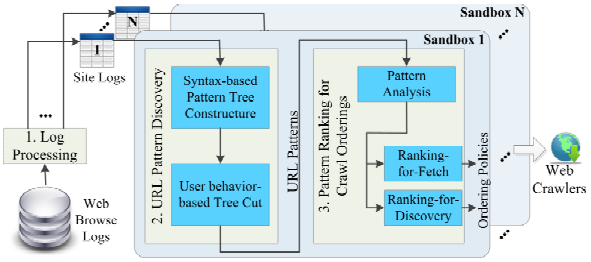

Figure 2: The overview of the proposed approach.

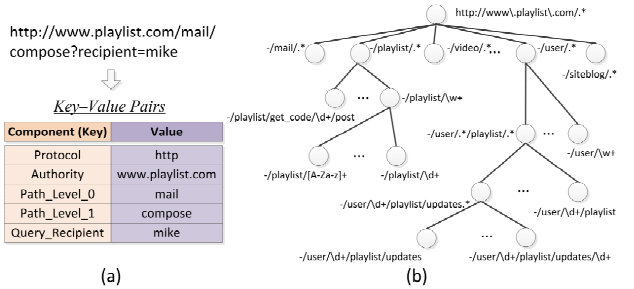

Figure 3: Illustrations of (a) URL decomposition and (b) the pattern tree (only shows a partial part for a clear view) for URLs from www.playlist.com.

– First, our approach *is capable of predicting the importance of unseen URLs*. URL patterns of a website are steady in a relatively long period, and the user behaviors related to a certain pattern won't change too much.
– Second, our approach *integrates the merits of both general and site-level crawl ordering policies*. General policies like PageRank deal with a majority of websites but cannot optimize the efficiency of a particular one; while site-level policies [21, 22] are designed for specific websites but are hard to be scaled up to the whole Web. By contrast, our approach can go deep to optimize site-level crawl efficiency, as well as go wide to provide a general solution.

Two technical obstacles are solved to make the proposed approach work:
– *How to determine the granularities of URL patterns*? General patterns are apt to mix up URLs with different characteristics and cannot help crawling; while subtle patterns have the risk of over-fitting which leads to poor generalization ability on unseen data. We propose to organize URL patterns in a tree structure, and select appropriate patterns from the tree through investigating the corresponding user behaviors.
– *How to properly leverage URL patterns to design crawl ordering policies*? URLs from various patterns act different roles in a website. We propose a behavior graph-based solution to rank URL patterns for two common crawling scenarios–*comprehensive fetch a website* and *timely discover new pages*.

Elaborate experiments have been carried out and the results are quite promising. First, the discovered URL patterns can describe user behaviors very well. Furthermore, the patterns are temporally reliable and with good generalization ability to unseen data. Second, the crawling efficiencies are noticeably improved. Under the same download throughput, the proposed approach discovers and fetches more informative pages than several traditional methods.

2. DATA AND FRAMEWORK OVERVIEW

The framework overview of the proposed approach is shown in Fig. 2, which mainly consists of three components: (1) *log processing*, (2) *URL pattern discovery*, and (3) *pattern ranking for crawl ordering*.

We work on browse log collected from anonymous users who have agreed to contribute their surfing history. Records in the raw log data are in the form of a triple

$$(URL_t, URL_r, GUID)$$

where URL_t is the target being browsed and URL_r is the referrer the user comes from. $GUID$ is a hexadecimal string to identify anonymous users. Duplicate records (*i.e.* the same user transit from URL_r to URL_t for multiple times) are removed to avoid bias brought by individual users. Several measurements are statistically defined to characterize user browsing behaviors:
– $\mathcal{F}_{in}(u)$. The frequency of a URL u being a visit target. It is defined as the number of records whose URL_t is u.
– $\mathcal{F}_{out}(u)$. The frequency of a URL u being a referrer. It is defined as the number of records whose URL_r is u.
– $\mathcal{F}_{trans}(u, v)$. The frequency of transiting from a URL u to v. It is defined as the number of records whose URL_t is v and URL_r is u.

To discover appropriate URL patterns, as shown in Fig. 2, we first construct a pattern tree to organize URLs based on their syntax structures. The "parent-child" relationship between tree nodes characterizes the syntax similarity very well at various granularities. Then we tailor the tree through merging child nodes to their parent if they have consistent user behaviors.

Next, we analyze the statistical characteristics of user behaviors on each URL pattern. For example, how many URLs in a pattern act as referrer in browsing? And what is the frequency that users transiting from URLs in one pattern to URLs in another? From such statistics we can estimate how users browse a website, and analogically design crawl ordering policies. Currently two crawling scenarios – *comprehensive fetch* and *timely discovery* – are considered and their ordering policies are discussed in the modules *ranking-for-fetch* and *ranking-for-discovery* in Fig. 2.

At last, we would like to emphasize that the proposed approach can be run in parallel in multiple sandboxes, each of which handles one website, as shown in Fig. 2. The only input of a sandbox is the site-level logs. It is easy to deploy this approach on a cluster to deal with millions of websites.

3. DISCOVERING URL PATTERNS

In this section, we introduce how to discover appropriate URL patterns from web browse logs.

3.1 Syntax-based Pattern Tree Construction

A URL is not an ordinary string but complies with the syntax scheme strictly defined in [1]. Based on the syntax scheme, a URL can be decomposed into a series of "key–value" pairs, as shown in Fig. 3 (a). In addition, URLs in the same website usually follow some designing principles. Specifically, different keys usually have different functions and play different roles.

Following the recursive split process in [12], we construct a pattern tree in a top-down manner. That is, we start from the root (which contains all the input URLs), and iteratively divide URLs into subgroups according to their values under a particular key. The selected key in each iteration is the one

Table 1: Pattern-level statistics of user behaviors.

Statistics	Description
$\|\|\mathbb{P}\|\|$	$\|\|\{u, u \in \mathbb{P}\}\|\|$
$\mathcal{F}_{in}(\mathbb{P})$	$\mathcal{F}_{in}(\mathbb{P}) = \sum_{u\in\mathbb{P}} \mathcal{F}_{in}(u)$
$\mathcal{F}_{out}(\mathbb{P})$	$\mathcal{F}_{out}(\mathbb{P}) = \sum_{u\in\mathbb{P}} \mathcal{F}_{out}(u)$
$\mathcal{F}_{trans}(\mathbb{P}_a, \mathbb{P}_b)$	$\mathcal{F}_{trans}(\mathbb{P}_a, \mathbb{P}_b) = \sum_{u\in\mathbb{P}_a, v\in\mathbb{P}_b} \mathcal{F}_{trans}(u, v)$

who has the most concentrated distribution of values. For more details please refer to [12]. Fig. 3 (b) shows an example pattern tree constructed for the website `www.playlist.com`.

3.2 User Behavior-based Pattern Selection

Table 1 defines several pattern-level statistics, which are statistically significant than URL-level ones. Two browsing behaviors, *visit* and *transit*, are then considered in sequence to cut down the syntax-based tree, as shown in Fig. 4.

3.2.1 Visit-based Tree-Cut

Visit-based tree-cut is to remove sibling nodes having similar visit traffic (*e.g.* the nodes in the red dashed ellipse in Fig. 4 (b)). The process can be considered as using a model (a tree-cut plan) to characterize a dataset (URL-based user traffic). A model with lower complexity is preferred if it can describe the data well. In this paper, the well-known minimum description length (MDL) principle [20] is adopted to balance the model complexity and the fit to the data. Let $\mathbb{U}$ be the set of input URLs. A tree-cut plan is denoted by a set $\Gamma = \{\mathbb{P}_1, \cdots, \mathbb{P}_K\}$ in which each $\mathbb{P}_k (1 \le k \le K)$ is a leaf node survived after the cut. The best plan Γ_{best} is the one that minimizes the sum of *model description length* $L(\Gamma)$ and *data description length* $L(\mathbb{U}|\Gamma)$, as

$$\Gamma_{best} = \arg\min_{\Gamma} L(\Gamma) + L(\mathbb{U}|\Gamma). \quad (1)$$

According to [20], $L(\Gamma)$ can be estimated by

$$L(\Gamma) = \frac{K-1}{2} \log \mathcal{F}_{in}(\mathbb{P}_{root}), \quad (2)$$

where $\mathcal{F}_{in}(\mathbb{P}_{root}) = \sum_{u\in\mathbb{U}} \mathcal{F}_{in}(u)$ is the total traffic on all the URLs in $\mathbb{U}$, and K is the size of Γ. $L(\mathbb{U}|\Gamma)$ can be described by the negative log-likelihood of the data given the model [10],

$$L(\mathbb{U}|\Gamma) = -\log p(\mathbb{U}|\Gamma) = -\sum_{u\in\mathbb{U}} p(u|\Gamma), \quad (3)$$

where $p(u|\Gamma)$ is defined as

$$p(u|\Gamma) = \frac{1}{||\mathbb{P}_k||} \frac{\mathcal{F}_{in}(\mathbb{P}_k)}{\mathcal{F}_{in}(\mathbb{P}_{root})}. \quad (4)$$

In other words, the probability of a single URL being visited is smoothed by the corresponding pattern-level average.

3.2.2 Transit-based Tree-Cut

As user behaviors are easy to be disturbed by temporal interruptions, child nodes belonging to the same functional role are still possible to have different visit frequencies. In such a case, transit behavior can help remove redundant nodes. For example, it is reasonable to remove the two nodes in the blue rectangle in Fig. 4 (b) if they have identical in-links and out-links, as shown in Fig. 4 (c).

Given a pattern $\mathbb{P}_k$, it is straightforward to characterize its in-link distribution $\mathbf{p}_k^{in}$ and out-link distribution $\mathbf{p}_k^{out}$ as

$$\begin{aligned} \mathbf{p}_k^{in} &= \{p_{i\to k}^{in}, 1 \le i \le K\}, \quad p_{i\to k}^{in} = \frac{\mathcal{F}_{trans}(\mathbb{P}_i, \mathbb{P}_k)}{\mathcal{F}_{in}(\mathbb{P}_k)} \\ \mathbf{p}_k^{out} &= \{p_{k\to i}^{out}, 1 \le i \le K\}, \quad p_{k\to i}^{out} = \frac{\mathcal{F}_{trans}(\mathbb{P}_k, \mathbb{P}_i)}{\mathcal{F}_{out}(\mathbb{P}_k)} \end{aligned}. \quad (5)$$

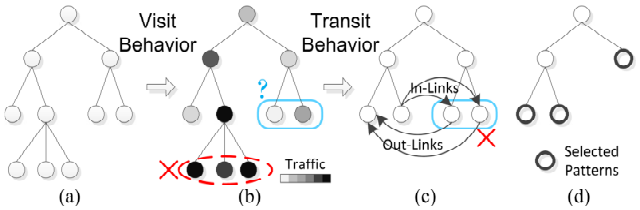

Figure 4: User behavior-based pattern selection.

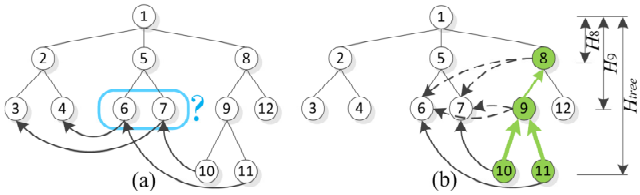

Figure 5: Illustrations of (a) a complex case to judge the transit similarity of node 6 and 7; and (b) propagate transit possibilities on the tree to smooth the in-link distributions of 6 and 7.

Such a definition can handle the simple case in Fig. 4 (c), while real cases are more complicated. As shown in Fig. 5 (a), the nodes 6 and 7 have different in-link and out-link nodes. However, it is noticed that their in-link nodes (10 and 11) and out-link nodes (3 and 4) are siblings of each other. The nodes 6 and 7 still have similar transit relationships if node 9 and 2 are respectively treated as their in-link and out-link nodes. To smooth the in-link and out-link distributions, the transits are propagated on the tree, as shown in Fig. 5 (b). The propagation is defined by a $K \times N$ matrix whose element$\mathbf{P}_{k,n}$is defined as

$$\mathbf{P}_{k,n} = \begin{cases} \prod_{h=H_n}^{H_k-1} \frac{h}{H_{tree}} & k \text{ is a leaf, } n \text{ is its ancestor} \\ 1 & k = n \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

where H_k and H_n are the heights of k and n on the tree, H_{tree} is the total tree height, and N is the total number of tree nodes. Eq. (6) is defined following two criteria:

- The transit possibility of a leaf node can only be propagated to its direct ancestors.
- The transit possibility keeps weakening when the propagation goes farther. The decay factor is experimentally set as H_{father}/H_{tree} where H_{father} is the depth of the father node.

With the $\mathbf{P}$ matrix, the K-dimensional sparse vectors in Eq. (5) are converted to N-dimensional dense ones, as

$$\hat{\mathbf{p}}_i^{in} = \mathbf{p}_i^{in} \times \mathbf{P} \quad \text{and} \quad \hat{\mathbf{p}}_i^{out} = \mathbf{p}_i^{out} \times \mathbf{P}. \quad (7)$$

And the transit similarity $sim_{transit}(i, j)$ between $\mathbb{P}_i$ and $\mathbb{P}_j$ is defined as the average of their in-link and out-link cosine similarities on the smoothed distributions, as

$$sim_{transit}(i, j) = \frac{1}{2}[\cos(\hat{\mathbf{p}}_i^{in}, \hat{\mathbf{p}}_j^{in}) + \cos(\hat{\mathbf{p}}_i^{out}, \hat{\mathbf{p}}_j^{out})]. \quad (8)$$

Based on Eq. (8), the cutdown process is carried out in a bottom-up way, as shown in Algorithm 1.

4. PATTERN RANKING FOR CRAWLING

In this section, we introduce how to design crawl ordering policies based on the discovered URL patterns.

Algorithm 1 Transit-based Tree Cut. The input is a pattern tree and the output is a series of URL patterns.

Let $\mathcal{Q}$ be a priority queue in which tree nodes are sorted in descending order according to their depths on the tree.
Initializing $\mathcal{Q}$. For each node, add it to $\mathcal{Q}$ if the node itself is not a leaf but all its children are leaves.
while $\mathcal{Q}$ is not empty **do**
  Pop the first node q from $\mathcal{Q}$, $\mathcal{C}_q$ is the set of q's children.
  $\xi_q = \min\{sim_{transit}(m,n), \forall m,n \in \mathcal{C}_q \ \ \& \ \ m \neq n\}$
  if $\xi_q > \xi_{thresh-to-cut}$ **then**
    Turn q into a leaf by cutting all the nodes in $\mathcal{C}_q$.
    Update $\mathbf{p}_i^{in}$, $\mathbf{p}_i^{out}$ and $\mathbf{P}$ following Eq. (5) and (6).
    if all the siblings of q are leaves **then**
      Add q's parent to $\mathcal{Q}$.
    end if
  end if
end while
return URL patterns of all the remaining leaf nodes.

4.1 Two Crawling Scenarios

Two common crawling scenarios–*batch* and *incremental* modes [14]–are exported to site-level here, as:

- *Comprehensive fetching a website* is similar to the *batch* mode, to periodically snapshot a website. The goal is to fetch as much as possible valuable information from the target website. Importance measurement in this situation should describe the value of a downloaded page itself.
- *Timely discovering new pages* partially realizes the function of the *incremental* mode, to discover newly created URLs via frequently checking a few "hub" pages. This is useful to monitor websites of social network or news. Importance measurement here should consider the total value of URLs linked from the downloaded page.

It is straightforward to design ordering strategies directly based on the pattern-level statistics of browsing behaviors.

Ranking-for-fetch strategies:

- *Rank-by-Volume*. URL patterns are sorted in descending order based on its volume $||\mathbb{P}||$. Large volume means that pattern is a dominant resource in a website, and should provide more information to users.
- *Rank-by-Overall-Visit*. Patterns attracting more overall user visit traffic, *i.e.* $\mathcal{F}_{in}(\mathbb{P})$, are with higher priorities.
- *Rank-by-Average-Visit*. Patterns are ordered according to the averaged visit frequency $\mathcal{F}_{in}(\mathbb{P})/||\mathbb{P}||$.

Ranking-for-discovery strategies:

- *Rank-by-Overall-Referral*. The importance of a pattern is evaluated based on the total amount of times that its URLs act as referrers in browse logs, *i.e.*, $\mathcal{F}_{out}(\mathbb{P})$.
- *Rank-by-Average-Referral*. Patterns are ranked based on the average frequency that their URLs play a referrer role in browsing, *i.e.*, $\mathcal{F}_{out}(\mathbb{P})/||\mathbb{P}||$.

All these strategies are partially reasonable from certain perspectives. However, they just locally consider the characteristics of an individual pattern, but neglect the relationships between patterns. To better combine both the behavior statistics and the relationships between URL patterns, we propose to use graph-based strategy to rank patterns.

4.2 Behavior Graph-based Pattern Ranking

The graph constructed in this paper is called behavior graph as it is different from the traditional link graph in several aspects. First, link graph in existing work is based on URLs, while nodes in the behavior graph are URL patterns. Therefore, the behavior graph is small and the related computation is cheap. Second, the behavior graph is initialized based on statistics at pattern-level, while link graphs are initialized almost randomly. Third, the transition probabilities (*i.e.* weights of graph edges) of the behavior graph are voted by users, while the edge weights of a link graph are based on hyper-links (*e.g.* in-degree and out-degree).

Of course, PageRank-like link analysis strategy is still feasible for the behavior graph. However, it measures the overall importance of a graph node and thus is only suitable for *comprehensive fetch*. In comparison, another popular link analysis method, the HITS algorithm [11], measures page importance from two aspects–*authority* and *hub*. For each node, the *authority* score depicts the value of its content and the *hub* score describes the value of its links. This perfectly matches the requirements of both *comprehensive fetch* and *timely discovery*. Following the instruction of the HITS algorithm, the two scores of a node $\mathbb{P}_i$ are initialized as

$$auth^0(\mathbb{P}_i) = \frac{\mathcal{F}_{in}(\mathbb{P}_i)}{\mathcal{F}_{in}(\mathbb{P}_{root})} \quad hub^0(\mathbb{P}_i) = \frac{\mathcal{F}_{out}(\mathbb{P}_i)}{\mathcal{F}_{out}(\mathbb{P}_{root})}. \quad (9)$$

The two scores are continually updated in a series of iterations. The updating formulas of the k^{th} iteration are

$$\begin{array}{r} auth^k(\mathbb{P}_i) = \sum_j p_{j\to i}^{out} \times hub^{k-1}(\mathbb{P}_j) \\ hub^k(\mathbb{P}_i) = \sum_j p_{i\to j}^{in} \times auth^{k-1}(\mathbb{P}_j) \end{array}. \quad (10)$$

Both $auth^k(\mathbb{P}_i)$ and $hub^k(\mathbb{P}_i)$ are normalized to unit magnitude after the update. The converged *authority* and *hub* of each node can properly estimate the possibilities of URLs in that pattern to be targets or referrers. Consequently, for *comprehensive fetch*, URL patterns are ranked in descending order based on $0.5 \times [auth(\mathbb{P}_i) + hub(\mathbb{P}_i)]$; and for *timely discovery*, patterns are simply ordered by $hub(\mathbb{P}_i)$.

5. EVALUATION AND DISCUSSION

Elaborate experiments have been carried out to demonstrate the feasibility of the proposed approach, for both URL pattern discovery and efficient site-level crawling.

5.1 Experiment Setup

The experiments were carried out on 9 websites, as shown in Table 2. These websites include forums, video websites, and several popular Web 2.0 portals. For each website there are three data resources: (1) the mirror cached at May 2010, using a brute-force crawler; (2) 6-month browse logs from Nov. 2009 to Apr. 2010, collected by web browsers; and (3) search results returned by Google, Bing, and Yahoo!, including URLs from that website and were clicked in web search. Browse logs and search results can be considered to describe user interest from two different perspectives. The scales of these data resources are listed in Table 2. In the experiments, the URL patterns were mined based on one-month log data (Nov. 2009), and the other five months' logs

Table 2: The dataset scales (#URLs) in evaluation.

Website	Mirror	30-day Log	Search Result
www.mitbbs.com	9,903,107	265,464	27,757
www.gumtree.com	9,135,292	973,317	64,663
www.cooks.com	3,555,610	353,492	200,354
6.cn	6,996,186	234,936	53,962
bbs.ngacn.cc	2,901,950	490,818	49,980
www.fotocommunity.de	8,282,845	1,699,434	22,226
ku6.com	17,896,352	1,613,165	214,059
www.playlist.com	10,831,023	964,819	20,783
fantong.com	4,952,562	67,060	6,789
Total	**74,454,927**	**6,662,505**	**660,573**

Table 3: The statistics of pattern trees

Pattern Tree	#Leaf Node			Tree Height		
	Min	Max	Avg	Min	Max	Avg
$\mathbb{T}_{SYN}$	28	1076	283.7	7	17	11.3
$\mathbb{T}_{VISIT}$	20	106	60.9	6	12	8.4
$\mathbb{T}_{TRAN}$	16	100	51.1	6	11	7.8

Table 4: Several evaluations of pattern qualities

Quality	$\mathbb{T}_{SYN}$	$\mathbb{T}_{VISIT}$	$\mathbb{T}_{TRAN}$	Random
Mirror Coverage	91.2%	99.1%	99.9%	–
Layout Similarity	0.915	0.889	0.881	0.322
Traffic Approximation	0.165	0.166	0.166	–

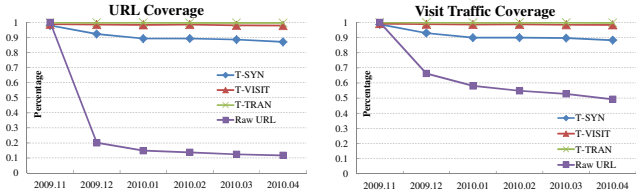

Figure 6: Temporal reliability of URL patterns.

were used for evaluation. All the crawling was simulated on the mirrors, which provide a consistent experimental environment. In the following evaluations, the average performances of all the 9 websites are reported.

5.2 Quality of URL Patterns

According to the steps in Section 3, the generated pattern trees are denoted as $\mathbb{T}_{SYN}$, $\mathbb{T}_{VISIT}$ and $\mathbb{T}_{TRAN}$. Some basic statistics of these pattern trees are listed in Table 3. Several evaluations were designed to measure the URL pattern qualities from four aspects.

Mirror coverage is to measure the *generalization ability* of URL patterns. Specifically, we measure how many URLs in the mirrors can be covered by the discovered patterns. From Table 4, it is noticed that around 10% URLs in the mirrors cannot be covered by the patterns from $\mathbb{T}_{SYN}$. In other words, these patterns somewhat overfit the training log data. In comparison, the patterns from $\mathbb{T}_{VISIT}$ and $\mathbb{T}_{TRAN}$ significantly improve the generalization ability and can cover almost all the URLs in the mirrors.

Page layout similarity is to measure the *distinguishability* of URL patterns. That is to say, a pattern should contain only one certain resource in a website. Here, the distinguishability is evaluated via checking the layout similarity between pages under the same pattern. This makes sense as most modern websites are built based on templates [5, 21]. We computed the pair-wise layout similarities between pages under the same patterns, as well as some random pages from different patterns for comparison. From Table 4, it is clear that pages grouped by URL patterns have better layout consistencies than randomly selected pages.

Visit traffic approximation is to measure the *summarization ability* of URL patterns. The pattern-level summarizations smooth the user browsing behaviors on individual URLs. However, such smoothing shouldn't distort the browsing behaviors too much. We calculated the Jensen-Shannon divergences between the visit traffic distribution on original URLs and the smoothed distributions on pattern-level. From Table 4, it is clear that the smoothing process (*e.g.* $\mathbb{T}_{SYN}$ *vs.* $\mathbb{T}_{TRAN}$) just slightly hurt the distribution of browsing behaviors.

Temporal reliability is to measure the period of validity of URL patterns. Both the URL coverage and the traffic

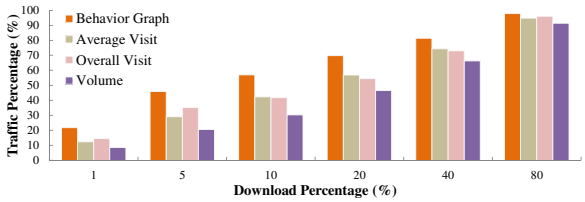

Figure 7: Comparisons of different pattern-based ordering strategies, for *comprehensive fetch*.

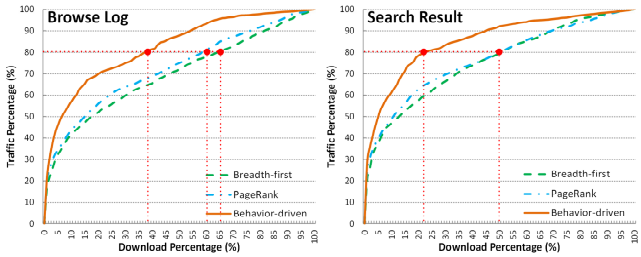

Figure 8: Comparisons of the breadth-first, PageRank, and the proposed behavior-driven approach, for *comprehensive fetch*.

coverage were evaluated along time. For comparison, we also measured the URL-level reliability, that is, the intersection percentage of URLs in logs of different periods. As shown in Fig. 6, the URL-level reliability drops very quickly, which perfectly echoes the statement in [14] that URLs retire rapidly. By contrast, URL patterns (especially those from the final tree-cut $\mathbb{T}_{TRAN}$) are quite reliable. This is a convincing justification to do pattern-level crawl ordering.

5.3 Crawling Efficiency

The evaluations of crawling performance were set up respectively for *comprehensive fetch* and *timely discovery*.

5.3.1 Comprehensive Fetch a Website

Two factors were monitored for the evaluation of the *comprehensive fetch* scenario. One is the throughput which indicates how many web pages have been downloaded; the other is the amount of downloaded information which is described by the sum of traffic[1] of the fetched URLs.

In Fig. 7, we compare the behavior graph-based ranking strategy with the three naïve ranking-for-fetch strategies in Section 4.1. Several different throughput conditions, from 1% to 80%, were investigated. Among the four strategies, rank-by-volume is always the worst. This indicates that the visit traffic is not in proportion to the resource scale (a URL pattern can be considered as a kind of resource in a website). Rank-by-overall-traffic and rank-by-average-traffic are more efficient but still not good enough. In comparison, the behavior graph-based strategy remarkably outperforms all those naïve strategies, especially when the throughput is small. This is a nice advantage as throughput is always the biggest bottleneck in crawling.

Fig. 8 compares the proposed approach with two popular ordering policies, *breadth-first* and *PageRank*, deployed in industry. The link-depth and PageRank scores were calculated based on the mirrored websites. Besides the browse logs, the information amount was also estimated based on the click numbers in the search results. It is noticeable that

[1]For a URL in browse log, the traffic is $\mathcal{F}_{in}(u)$; for a URL in search results, the traffic is its click number; otherwise the traffic is simply set as zero.

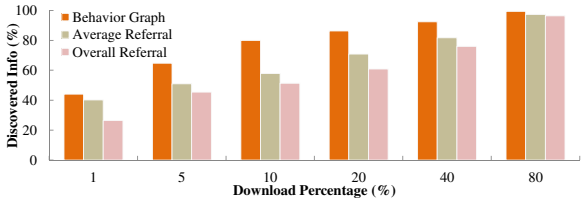

Figure 9: Comparisons of different pattern-based ordering strategies, for *timely discovery*.

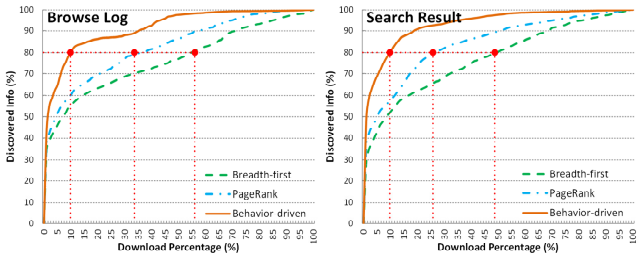

Figure 10: Comparisons of the breadth-first, PageRank, and the proposed behavior-driven approach, for *timely discovery*.

the proposed approach is much better than the breadth-first and PageRank methods. For example, to download 80% visit traffic, our approach needs to fetch less than 40% pages form a website; while the other two methods need to download more than 60% pages to achieve this goal. Under the measurement of click number in search results, the superiority is even more obvious, as shown in the right part of Fig. 8. This is a good news to search engines, as URLs frequently clicked in search results are quite important to search relevance.

5.3.2 *Timely Discover New Pages*

For evaluation of the *timely discovery* scenario, we replace the amount of downloaded information to the amount of discovered information. The discovered information is defined as the sum of traffic of all the seen URLs. A URL is called a seen URL once it was parsed from a fetched page and has been added to the crawling queue. The discovered information provides an expectation of the amount of information can be downloaded in the future.

In Fig. 9, we compare the behavior graph-based ranking strategy with the other two ranking-for-discovery strategies in Section 4.1. Among the three strategies, rank-by-average-referral is better than rank-by-overall-referral. This makes sense as dominant hub patterns in a website usually have only a few URLs which frequently act as referrers in browsing. Although the count of overall referral may be small, the average number of referrals on each page is large. Again, the behavior graph-based strategy is the best under all the throughputs. This suggests the quality of referrals is also important in ranking a referrer.

Finally, the efficiencies of the breadth-first, PageRank, and our approach are compared in Fig. 10. Our approach is still better than the breadth-first and PageRank methods. For instance, to discover 80% traffic, our approach needs to download around 10% pages; while PageRank needs to download about 35% pages and the breadth-first needs to fetch 55%. In other words, a crawler equipped with our approach can monitor 80% information of a website via only downloading and checking 10% URLs. This is especially useful for the crawling of Web 2.0 websites. In addition, our approach also held the superiority under the measurement of click number, as shown in the right part of Fig. 10.

6. CONCLUSIONS

In this paper we proposed a user behavior-driven method for efficient crawling. The contributions are four-fold:

- Propose to leverage web log data to help web crawling. This is more direct than existing technologies to predict user interest and optimize crawl ordering.
- Propose to summarize web logs with URL patterns. Come up with a pattern tree-based method to identify URL patterns in a proper granularity from log data.
- Propose several strategies to rank patterns for two crawling scenarios, comprehensive fetch and timely discovery.
- The proposed approach goes deep to optimize site-level crawl ordering policies, and is capable of going wide to deal with millions of websites in parallel mode.

7. REFERENCES

[1] Uniform resource identifier: Generic syntax. RFC 3986.
[2] S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. In *Proc. WWW*, pages 280–290, 2003.
[3] R. Baeza-Yates, C. Castillo, M. Marin, and A. Rodriguez. Crawling a country: better strategies than breadth-first for web page ordering. In *Proc. WWW*, pages 864–872, 2005.
[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1–7):107–117, 1998.
[5] R. Cai, J.-M. Yang, W. Lai, Y. Wang, and L. Zhang. iRobot: an intelligent crawler for web forums. In *Proc. WWW*, pages 447–456, 2008.
[6] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11–16):1623–1640, 1999.
[7] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks*.
[8] J. Cho and U. Schonfeld. RankMass crawler: a crawler with high personalized PageRank coverage guarantee. In *Proc. VLDB*, pages 375–386, 2007.
[9] D. Fetterly, N. Craswell, and V. Vinay. The impact of crawl policy on web search effectiveness. In *Proc. SIGIR*, pages 580–587, 2009.
[10] P. Grünwald. *A tutorial introduction to the minimum description length principle*. Advances in Minimum Description Length: Theory and Applications. MIT Press, 2005.
[11] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.
[12] T. Lei, R. Cai, J.-M. Yang, Y. Ke, X. Fan, and L. Zhang. A pattern tree-based approach to learning URL normalization rules. In *Proc. WWW*, pages 611–620, 2010.
[13] F. Menczer, G. Pant, and P. Srinivasan. Topical web crawlers: evaluating adaptive algorithms. *ACM Trans. Internet Techn.*, 4(4):378–419, 2004.
[14] C. Olston and M. Najork. Web crawling. *Foundations and Trends® in Information Retrieval*, 4(3):175–246, 2010.
[15] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: bringing order to the Web. Technical report, Stanford University, 1999.
[16] S. Pandey and C. Olston. User-centric web crawling. In *Proc. WWW*, pages 401–411, 2005.
[17] S. Pandey and C. Olston. Crawl ordering by search impact. In *Proc. WSDM*, pages 3–14, 2008.
[18] G. Pant and P. Srinivasan. Learning to crawl: comparing classification schemes. *ACM Trans. Inf. Syst.*
[19] B. Pinkerton. Finding what people want: experiences with the web crawler. In *Proc. WWW*, pages 3–14, 1994.
[20] J. Rissanen. Stochastic complexity and modeling. *The Annals of Statistics*, 14(3):1080–1100, 1986.
[21] M. L. A. Vidal, A. S. da Silva, E. S. de Moura, and J. M. B. Caval-canti. Structure-driven crawler generation by example. In *Proc. SIGIR*, pages 292–299, 2006.
[22] Y. Wang, J.-M. Yang, W. Lai, R. Cai, L. Zhang, and W.-Y. Ma. Exploring traversal strategy for efficient web forum crawling. In *Proc. SIGIR*, pages 459–466, 2008.