A Thread-wise Strategy for Incremental Crawling of Web Forums

Jiang-Ming Yang†, Rui Cai†, Chunsong Wang†, Hua Huang‡, Lei Zhang†, Wei-Ying Ma†

†Microsoft Research, Asia. {jmyang, ruicai, leizhang, wyma}@microsoft.com
‡University of Wisconsin-Madison. chunsong@cs.wisc.edu
§Beijing University of Posts and Telecommunications. huanghua@bupt.edu.cn

ABSTRACT

We study in this paper the problem of incremental crawling of web forums, which is a very fundamental yet challenging step in many web applications. Traditional approaches mainly focus on scheduling the revisiting strategy of each individual page. However, simply assigning different weights for different individual pages are usually inefficient in crawling forum sites because of different characteristics between forum sites and general websites. Instead of treating each individual page independently, we propose a thread-wise strategy by taking into account thread-level statistics, for example, the number of replies and the frequency of replies, to estimate the activity trend of each thread. To extract such statistical information, we develop a simple yet very robust approach to extracting the timestamp of each post in a discussion thread. We also employ a regression model to predict the time of the next post for each thread. Based on this model, we developed a highly efficient crawler which is 2.6 times faster than state-of-the-art methods in terms of fetching new generated content, and meanwhile can still ensure a high coverage ratio. Experimental results show encouraging performance of Coverage, Bandwidth utilization, and Age for our approach on various forums.

Categories and Subject Descriptors
H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval → C clustering, information filtering

General Terms
Algorithms, Performance, Experimentation

Keywords
Forum, Sitemap, Incremental crawling

1. INTRODUCTION

Due to the explosive growth of Web 2.0, web forum (also named bulletin or discussion board) is becoming an important data resource on the Web [1], and many research efforts and commercial systems, such as Google, Yahoo!, and Live, have begun to leverage information extracted from forum data to build various Web applications. This includes, for example, understanding user opinions from digital product reviews [1] for both product recommendation and business intelligence analysis, integrating travel notes for travel recommendations [2], extracting question-answer pairs for QnA service [9], etc.

For most web applications, the first step is to fetch data pages from various forum sites distributed on the whole Internet. However, web forum crawling is not a trivial task and cannot be easily handled by using a generic web crawler. Compared to general websites, web forums have some unique characteristics. Every day, a substantial number of discussion threads are created or frequently updated, and meanwhile many discussion threads become inactive and are no longer changed. This is quite different from generic web pages which usually have static URLs in traditional websites.

Realizing the importance of forum data and the challenges in forum crawling, Cai et al. [5, 17] studied how to reconstruct the sitemap of a target web forum, and how to choose an optimal traversal path to crawl informative pages only. However, this work only addressed the problem of fetching as much as possible valuable data, but left the problem of keeping previously downloaded pages as fresh as possible [14] untouched.

An optimal forum crawling strategy needs to balance the tradeoff between getting new discussion threads and updating existing discussion threads. The first part corresponds to how to efficiently find new URLs of new discussion threads. This is important because many new problems and events are responded quickly in forum sites. The second part is to identify updated threads and recrawl the new added content efficiently. In many cases a discussion thread starts from a question, and answers are followed in the later replies. An efficient solution to keep updating existing threads can significantly improve the quality of search engine index and the ability of a web application to respond to latest affairs timely.

Traditional methods mainly focus on scheduling the revisiting strategy of each individual page. However, simply assigning different weights for different individual pages are usually inefficient in crawling forum sites because of different characteristics between forum sites and general websites. First, long discussion threads or thread lists are usually divided into multiple pages and connected by page-flipping links. The update frequencies of these pages are relevant.

1www.dpreview.com
2www.tripadvisor.com
with each other. For example, the content of a discussion thread grows periodically, when new replies are appended. As a result, old replies will be moved to other pages belong to this thread. Once these replies are downloaded, the crawler will not need to revisit them again. Thread list pages are also updated and grown in a similar way. Ignoring relationships among forum pages, existing approaches will result in a considerable waste of network bandwidth. Second, most discussion threads are only kept active within a few days after they are first created. From our empirical study, more than 75% threads are no longer active three days after they were created. That is, the forum pages belonging to these threads will no longer be updated. This is also quite different from the longevity of web pages in general websites. Existing approaches will also waste substantial bandwidth for revisiting such static content.

In this paper, we target at developing an incremental crawling strategy as general as possible to efficiently crawl web pages from any forums sites. Instead of treating each individual page independently, as did in generic crawlers, we take into account thread-level statistics in each discussion thread, for example, the number of replies, the frequency of replies, and the time stamp of each reply. Such information is of great help for developing an efficient recrawl strategy. To have a better understanding of user behaviors in a target forum site, we first reconstruct the sitemap of this forum site by grouping forum pages according to their page layout similarities [5]. With the automatically reconstructed sitemap, we then identify page-flipping links which are purposely designed to help users to browse a long discussion thread that is divided into multiple pages. Thereafter, we can analyze each discussion thread as a whole by concatenating multiple pages belonging to this thread together. List-of-thread pages can also be processed similarly. As thread-level statistics can be utilized, it is expected that a much efficient forum crawling strategy could be developed to outperform existing approaches which usually ignore the relationships among forum pages.

To leverage such thread-level and site-level statistical information, we need to reliably extract the timestamp of each post in a discussion thread. Based on a simple observation that all the posts belonging to one thread are organized sequentially in multiple pages, we develop a simple yet very robust approach to finding the HTML pattern that timestamps are sorted in an ascending or descendant order. We will describe the timestamp detection approach in detail in the following sections. This approach enables us to estimate the activity trend of each thread. In case some threads are short in length and have limited information, we can leverage some site-level statistics, such as the average time interval between two consecutive posts, to estimate its approximate update frequency. We have analyzed such kinds of information in both threads and the whole site, and found most of them are linearly related to time interval between two consecutive posts. To take into consideration all these factors, we employ a regression model to predict the time of the next post. Based on this model, we developed a highly efficient crawler which is 2.6 times faster than state-of-the-art methods in terms of fetching the new generated content, and meanwhile can still ensure a higher coverage ratio.

We briefly highlight the novelties and contributions of our approach as follows.

- **Thread-wise Estimation.** Our method can automatically learn page relationships for a given forum site. By following the page-flipping links of each discussion thread or list-of-thread [17], we can concatenate multiply pages belonging to the same thread or list-of-thread together and treat them as a whole. In this way, we can avoid to repeatedly crawl existing posts but focus on newly updated content. The thread-level statistics are also very helpful for estimating the update frequency of each discussion thread.

- **Regression Based Prediction Model.** In forum sites, a lot of site-level information can help us to estimate the update frequency of an object (a discussion thread or a thread list) and its longevity more accurately, such as the number of users, the hotness of each discussion board, and users’ historical behaviors. Based on the statistics of each discussion thread and the site-level information, we propose a regression-based prediction model, which can effectively combine various statistics to predict the time interval between the last reply and the upcoming reply in this discussion thread.

- **Bandwidth Control.** By treating a discussion thread as an entire post list, we can easily find the latest replies in each thread. Similarly, by understanding thread lists, we can discover new discussion threads efficiently. As the two types of pages play different roles in forums, we propose an efficient bandwidth control strategy for different types of pages to balance between discovering new content (crawl) and refreshing existing content (crawl).

This paper is organized as follows. First, we briefly review some related works in Section 2 and describe the problem setting and system framework in Section 3 and Section 4. Then the object-based incremental crawling strategy for web forums is presented in detail in Section 5. Experimental evaluations are reported in Section 6. And in the last section, we draw conclusions and point out some future research directions.

2. RELATED WORKS

To the best of our knowledge, little existing work in literatures has systemically investigated the problem of forum incremental crawling. However, there are still some previous work that should be reviewed, as our approaches were motivated by them.

Some early work in [3, 4] first investigated the dynamic web and treat the information as a depreciating commodity. They first introduced the concepts of **lifetime** and **age** of a page which is important for measuring the performance of an incremental crawler. However, they treated every page equally and ignored the importance and change frequency which are also important to an incremental crawler.

Whether to minimize the **age** or to maximize the freshness leads to a variety of analytical models by investigating different features. We classify them into three categories:

1. How often the content of a page is updated by its owner. Coffman et. al. [8] analyzed the crawling problem theoretically. Cho et. al. [7] proposed several methods based on page update frequency. However, most of these methods are based on the assumption that most web pages change as a
Poisson or memoryless process. But the experimental result in [4] showed that most web pages are modified during the span of US working hours (between 8 AM and 8 PM, Eastern time, Monday to Friday). This phenomenon is even more evident in forum websites due to the content are all generated by forum users. Edwards et. al. [11] tried to use an adaptive approach to maintaining data on actual change rates for the optimization without the above assumption. However, it still treated each page independently and may waste bandwidth in crawling the same post which appears several times in other pages belonging to the same discussion thread. Furthermore, we argue that some other factors, such as the time intervals for the latest several posts and the average time interval between two consecutive posts, are more important in web forums. We will show their importance in the experiment part.

(II) The importance of each web page. Baeza-Yates et. al. [2] tried to determine the weight of each page based on some strategies similar to PageRank. Wolf et. al. [18] assigned the weight of each page based on the embarrassment metric of users’ search results. In the user-centric crawling [15], the targets are mined from user queries to guide the refreshing schedule of a generic search engine; First of all, some pages may have equal importance weight but different update frequency, thus only measuring the importance of each web page is not enough. Second, both PageRank importance and content importance are useless in web forums. Most pages in web forums are dynamic pages which are generated using some pre-defined templates. It is very hard to compute their PageRank importance since there are medial links among these pages. Furthermore, the content importance measurement is also useless. Once a post is generated, this post always exists unless the user deletes it manually. But before we get these post information, it is very hard to measure their importance or index quality through related search results. But while we have their content importance information we usually do not need to revisit it anymore. Some work named focused crawling attempts to only retrieve web pages that are relevant to some pre-defined topics [10, 6, 13] or some labeled examples [16] by assigning pages similar to the target page a higher weight. The target descriptions in focused crawling are quite different in various applications.

(III) The information longevity of each web page. Olston et. al. [14] introduced the longevity factor to determine revisit frequency of each web page. However, the information longevity in web forums are useless since once a post is generated, this post exists unless being deleted manually. This is one of the major differences between general web pages and forum web pages. Moreover, its three generative models are still based on the poisson distribution and some modified forms.

All the existing methods lack considering of the trade-off between discovering new threads and keeping existing threads fresh. For example, discussion thread pages usually contain most useful contents in web forums while list-of-thread pages contain fewer detailed contents. And a list-of-thread page usually gets lower update frequency than a post-of-thread page which is affected by every posting activity. From all the above aspects, it seems that we should assign lower weights for list-of-thread pages when scheduling the crawling queue. In contrast, since we can only get new discussion threads from list-of-thread pages, we may get a very poor performance using the above strategy. Unfortunately, none of existing methods has taken this into account. We will show its importance in the experiment part.

3. PROBLEM SETTING

To make a clear presentation and to facilitate the following discussions, we first explain some concepts used in this paper.

- **Forum Organization.** For users’ convenience, a well organized forum site consists of a tree like directory structure containing at the lowest end topics (commonly called threads) and inside them posts. The messages within these most conventional forum sites are displayed in several sub-boards or sub-topics. Fig. 1 provides an illustration of the organization from the ASP.NET forum (http://forums.asp.net). Different forum sites may have different organizations which cause different tree levels. For example the level of the tree for the ASP.NET is four.

- **List-of-thread.** The pages of the branch node in the tree are called list-of-thread pages. A long list may be divided into several pages and connected by page-flipping links in sequence. These pages usually belong to the same sub-boards or sub-topics and share similar content. While some new items are appended to this list, some old items may not be shown in the original page. These old items are not removed, but scrolled down to the following pages.

- **Discussion thread.** The pages of the leaf node in the tree are called discussion thread pages. A long discussion thread may be divided into several pages and connected by page-flipping links.
Since the list-of-thread pages and discussion thread pages contain almost all valuable information in forum sites, in this paper, we only focus on the crawling strategy for such two kinds of pages.

4. SYSTEM OVERVIEW

The flowchart of the proposed method is illustrated in Fig. 2, which mainly consists of three parts: (a) identifying discussion threads and related lists with automatically generated sitemap; (II) predicting page update frequency based on the regression model; and (III) balancing bandwidth among different kinds of pages and feed top K pages into the queue to crawl.

To identify discussion thread pages and list-of-thread pages, we first sample some pages from the target site. Empirical study showed that sampling around 2000 pages is enough for most forum sites. After that, pages with similar layout are clustered into groups (vertices) using the single linkage algorithm [5]. Since the data of discussion threads are usually stored in a backend database and rendered using some pre-defined template, discussion thread pages usually have similar layout and belong to the same group (vertex). It is the same for list-of-thread pages.

To identify discussion thread pages and list-of-thread pages, we first sample some pages from the target site. Empirical study showed that sampling around 2000 pages is enough for most forum sites. After that, pages with similar layout are clustered into groups (vertices) using the single linkage algorithm [5]. Since the data of discussion threads are usually stored in a backend database and rendered using some pre-defined template, discussion thread pages usually have similar layout and belong to the same group (vertex). It is the same for list-of-thread pages.

5. DETAILED ALGORITHMS

In this section, we will describe the three steps one by one.

5.1 Thread and List Generation

With the category information for discussion thread and list-of-thread pages, we can avoid repeatedly crawling the content within the same thread or list but scrolling down in following pages. It also helps on estimating the update frequency of such pages. Since discussion thread and list-of-thread are logical concepts, we first need to identify them from individual pages.

We first randomly sample about 2000 pages and cluster the pages based on their layout similarity into groups (vertices) using the single linkage algorithm [5]. After that, we map each individual page into one of the vertexes. We wish to concatenate all the individual pages belonging to the same object together. Obviously, the pages belonging to the same object should share the same template and belong to the same vertex. To achieve these pages together, we only need to achieved the pages connected by page-flipping links within the same vertex together. Page-flipping link is a kind of loop-back links of a vertex and can be distinguished from other loop-back links [17] as shown in Figure 3.

Valuable information in forum pages is mostly shown in some repetitive manners both for list pages and post pages, as it is essentially data records stored in a database. A repetitive region on a Web page is a block area containing multiple data records in a uniform formation [5]. Thus we will treat the repetitive region with largest area ratio as the main block area and segment pages into each record unit. These algorithms have been well studied recently; for more connected by page-flipping links, we can achieve them together by detecting the page-flipping links and treat them as an entire thread list or post list [17].

After that, some information can be extracted from the object and shared with all corresponding individual pages. As list pages and post pages have different information, we use separate models for these two kinds of objects. Combining with some global information, we can predict the update frequency for each page and give it an appropriate weight.

Finally, we can get new discussion threads by revisiting list pages and get the new posts in existing discussion threads by revisiting post pages. With the fixed bandwidth, our method will balance the requirement as we described in Section 1 by balancing the bandwidth between two kinds of objects with an optimized ratio and selecting the top K pages with the highest weights to crawl. More details about our method will be discussed in the following sections.
The time interval of next post may also become random. Thus it may also affect the update frequency of this thread. can represent its hotness which may reflect users’ interests.

The length of thread is too long. The experimental result in [4] shows that most web pages are modified during the span of US working hours (between around 8 AM and 8 PM, Eastern time, Monday to Friday). We conducted this experiment to verify this result in web forum sites. We analyzed the average number of posts in above sites according to each day of week or each hour of day. The results are shown in Figure 4(c) and Figure 4(d). It is quite similar to result in previous work. And then we describe the detailed factors we used in this paper one by one.

- Time intervals of the latest six records. The post time is recorded by a forum site which is more accurate to describe the update history of each discussion thread than crawler’s revisit history. The relation between the current post time and the time interval of next post can be verified similar to Figure 4(a). In contrast to leveraging the post time directly, we will leverage the time intervals between each two consecutive post time which represent the recent update frequency of this discussion thread. There are three steps to extract the time information. We first get the post time candidates whose content is a short one and contains digit string such as mm-dd-yyyy and dd/mm/yyyy, or some specific words such as Monday, Tuesday and Wednesday. Secondary, we align the html elements contain time information into several groups based on their DOM path since the post time in each post record should have the same DOM path in html document. Finally, since the post records are generated in sequence, the post time should satisfy a sequence order. This helps us filter noisy time content, such as users’ registration time, and get the right information.

5.2 Prediction Model

In this section, we will describe the details of our methods. Besides traditional three factors: revisit history, importance, and longevity information of each individual page, we think some other factors are more important in web forum sites. We will analyze some behaviors based on these factors in forum sites, such as the time intervals for the latest several posts, the average time interval between consecutive posts and so on. The results are shown in Figure 4.

We first look at the relation between the time interval of next post and the average time interval in thread, as shown in Figure 4(a). When the average time interval of existing posts in a given thread increases, the time interval of next post may increase accordingly. But it becomes noisy when the average time interval of a given thread is larger than 8 hours. This makes sense because the users’ behavior may be quite random since the last post is 8 hours ago. We can get similar results for the time interval of the latest several posts.

Another experimental result in Figure 4(b) shows the relation between the time interval of next post and the length of thread. The result is quite similar. The length of thread can represent its hotness which may reflect users’ interests. Thus it may also affect the update frequency of this thread. The time interval of next post may also become random when the length of thread is too long.

Figure 4: Some analysis of forum behaviors.
We can get the thread generation time in list record similarly. In our experiment, we only use the time intervals of the latest six records and represent them as:

\[ \Delta t_0, \Delta t_1, \Delta t_2, \Delta t_3, \Delta t_4 \]  

(1)

- Average time interval in thread or list. The average time interval represents the thread or list update history. Since the time intervals of the latest six records may be affected by some accidents, while the average time interval helps on smoothing the result and tolerates these accidents. The effect is shown in Figure 4(a). We can represent it as:

\[ \Delta t_{avg} \]  

(2)

- Average time interval in forum site. At the beginning of each discussion thread or thread list, few information can be used in predicting its update frequency. We can leverage the average time interval in current site to approximately estimate its possible update frequency. We can represent it as:

\[ \Delta t_{site} \]  

(3)

- Thread or list length. The length of a thread or list can represent its hotness which may lead to users’ interests. Thus it may also affect the update frequency of this thread or list. The result is shown in Figure 4(b). We can represent it as:

\[ \text{len} \]  

(4)

- Current time. There may exist more new records in the day and fewer posts in the night. The experiment result in [4] also shows the update frequency of web pages is highly dependent on the current time. We have verified this in Figure 4(c). We can represent the current time in 24 hours via the vector with 24 dimensions. For example, we can represent 3 PM by setting \( c_{t15} = 1 \) and others to zero.

\[ c_{t0}, c_{t1}, c_{t2}, \ldots, c_{t23} \]  

(5)

- Average time interval in current hour. Since the update frequency of web pages is highly dependent on current hour, we split one day into 24 hours and leverage the average post number of forum site in current time span.

\[ t_{hcurr} \]  

(6)

- Current day. There may exist more new records in working day and fewer posts in weekend. We have verified this in Figure 4(d). We can represent the current via the vector with 7 dimensions. For example, we can represent Wednesday by setting \( c_{d3} = 1 \) and others to zero.

\[ c_{d1}, c_{d2}, c_{d3}, \ldots, c_{d7} \]  

(7)

- Average time interval in current day. Since the update frequency of web pages is highly dependent on the current day, we split one week into 7 days and leverage the average post number of forum site in current time span.

\[ t_{dcurr} \]  

(8)

- The state of current thread or list. Ideally, we can estimate update frequency of current thread or list by checking similar threads. To achieve this goal, we first represent the state of current thread or list via Equation 1 and then clustering them into 15 clusters by their Euclidean distances. An example of the clustering result is shown in Fig. 5. For each new object, we assign it to one of the states with smallest Euclidean distance. We can represent the 15 states via the vector with 15 dimensions. For example, we can represent state 5 by setting \( s_5 = 1 \) and others to zero.

\[ s_0, s_1, s_2, \ldots, s_{14} \]  

(9)

- Thread Dead. We first have a look at the analysis of the average thread activity in forum sites. We process all the threads and calculate their activity time by checking the time of the first post and the last post in the thread. The result is shown in Fig. 6. The figure represents the percentage of threads with different activity time. From the figure we can see the thread activity is a typical power law curve. The activities of more than 40% threads are no longer than 24 hours and 70% threads are no longer than 3 days. This is the major reason why forum incremental crawling strategy is different with traditional incremental crawling strategies.

Once a thread is created, it may become static when there is no discussion activity for several days. We introduce this dead state indicator to avoid wasting bandwidth. Suppose there is no discussion activity.
for $\Delta t_{na}$ time from the last post. We compute the standard deviation of time interval $\Delta t_{sd}$ by $\Delta t_{sd} = \sqrt{1/(N-1) \cdot \sum_{i=1}^{N} (\Delta t_i - \Delta t_{avg})^2}$ and $N$ is the number of post records. If $\Delta t_{na} - \Delta t_{avg} > \alpha \cdot \Delta t_{sd}$, we may set $ds = 1$, otherwise, $ds = 0$. This factor is for discussion thread only.

$$ds = (10)$$

The results in Figure 4 show that most factors have linear relationship with the time interval of next post. To combine these factors smoothly and rapidly, we will leverage the linear regression model which is a lightweight combination model and efficient for online processing.

$$F(x) = w_0 + \sum_{i=1}^{N} w_i \cdot x_i \quad (11)$$

For each forum site, we can train two models for list object and post object separately. Before we begin the real crawling task, we can first fetch some sample pages. For each list record or post record from sampled pages, we can generate features $x$ based on above formulas and predict the time interval $y$ of the next record. We collect all these features and time interval pair $(x, y)$ as the training samples. By setting $x_0 = 1$, we can get the corresponding $\mathbf{W}$ by Equation 12.

$$\mathbf{W} = (\mathbf{X}^T \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^T \cdot \mathbf{Y} \quad (12)$$

In the crawling process, we will estimate the new information of each object by $\Delta I = (CT - LT)/F(x)$, where $CT$ is the current time and $LT$ is the last revisit time. We will use this weight to schedule the crawling. The two models will also be updated during the crawling process.

5.3 Bandwidth Control

As we have discussed in previous sections, we need to consider the tradeoff between discovering new pages and keeping existing pages fresh. It seems that the post-of-thread pages may get lower weight considering all existing factors. But we need to guarantee the bandwidth for the post-of-thread pages since we can only get the new discussion threads from post records. We need to guarantee the bandwidth for the list object, in this paper, the scenario we have considered is a real case. The crawler is required to crawl a target forum site from its portal page. The crawling task may last for about one year. We evaluate the performance of the crawler integrally by different metrics and in different periods. Before we describe the experiment details, we first introduce the experiment setup and the metrics.

6. EXPERIMENTS

Different from existing work which only considered revisiting existing pages, in this paper, the scenario we have considered is a real case. The crawler is required to crawl a target forum site from its portal page. The crawling task may last for about one year. We evaluate the performance of the crawler integrally by different metrics and in different periods. Before we describe the experiment details, we first introduce the experiment setup and the metrics.

<table>
<thead>
<tr>
<th>Id</th>
<th>Forum Site</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><a href="http://www.aswforum.com">www.aswforum.com</a></td>
<td>Audio and video</td>
</tr>
<tr>
<td>2</td>
<td>boards.cruisecritic.com</td>
<td>Cruise travel message</td>
</tr>
<tr>
<td>3</td>
<td><a href="http://www.cybertechhelp.com">www.cybertechhelp.com</a></td>
<td>Computer help community</td>
</tr>
<tr>
<td>4</td>
<td><a href="http://www.disboards.com">www.disboards.com</a></td>
<td>Disney trip planning</td>
</tr>
<tr>
<td>5</td>
<td>drupal.org</td>
<td>Official website of Drupal</td>
</tr>
<tr>
<td>6</td>
<td><a href="http://www.esportbike.com">www.esportbike.com</a></td>
<td>Sportbike forum</td>
</tr>
<tr>
<td>7</td>
<td>web2.flyertalk.com</td>
<td>Frequent flyer community</td>
</tr>
<tr>
<td>8</td>
<td><a href="http://www.gpsreview.net">www.gpsreview.net</a></td>
<td>GPS devices review</td>
</tr>
<tr>
<td>9</td>
<td><a href="http://www.kawiforums.com">www.kawiforums.com</a></td>
<td>Motorcycle forum</td>
</tr>
<tr>
<td>10</td>
<td><a href="http://www.pctools.com">www.pctools.com</a></td>
<td>Personal computer tools</td>
</tr>
<tr>
<td>11</td>
<td><a href="http://www.photo.net">www.photo.net</a></td>
<td>Photography community</td>
</tr>
<tr>
<td>12</td>
<td>photography-on-the.net</td>
<td>Digital photography</td>
</tr>
<tr>
<td>13</td>
<td>forums.photographyreview.com</td>
<td>Photography review</td>
</tr>
<tr>
<td>14</td>
<td><a href="http://www.phpbuilder.com">www.phpbuilder.com</a></td>
<td>PHP programming</td>
</tr>
<tr>
<td>15</td>
<td><a href="http://www.pocketgpsworld.com">www.pocketgpsworld.com</a></td>
<td>GPS help, news and review</td>
</tr>
<tr>
<td>16</td>
<td><a href="http://www.railpage.com.au">www.railpage.com.au</a></td>
<td>The Australian rail server</td>
</tr>
<tr>
<td>17</td>
<td><a href="http://www.storm2k.org">www.storm2k.org</a></td>
<td>Weather discussion forum</td>
</tr>
<tr>
<td>18</td>
<td>forum.virtualtourist.com</td>
<td>Travel and vacation advice</td>
</tr>
</tbody>
</table>

6.1 Experiment Setup

To evaluate the performance of our system on various situations, 18 different forums were selected in diverse categories (including bike, photography, travel, computer technique, and some general forums) in the experiments, as listed in Table 1.

To set up a consistent data collection for further evaluation and comparison, we first mirrored these 18 forum sites using a customized commercial search engine crawler. The crawler was adjusted to be domain-limited and depth-unlimited. For each site, the crawler started from the portal page and followed any links within that domain; and a unique URL address was followed only once. Consequently, the crawler mirrored all the pages up to June 2008 from 18 forum sites. The mirrored dataset contains 990476 pages and 5407854 individual posts, from March 1999 to June 2008. The average longevity of 18 sites is about 4.08 years.

Thereafter, we manually wrote a wrapper for each forum to dump all useful structured data from raw web pages to a database, such as title, author, post time, and detailed post content. Having these content information and the corresponding post time, we can restore the state of a forum site at any given time. This is based on a simple yet reasonable assumption that a post will not be deleted after it is generated. At any given time, we can easily figure out whether a record exists based on its post time and how records are organized based on the forum’s layout. Therefore, to accelerate the experiments, we wrote 18 page generators for the 18 forum sites to simulate responses to downloading requests. The following crawling experiments were all simulated on this data collection. Empirical studies showed that simulated experiments are the same as real experiments. Assuming a fixed bandwidth, the crawler was required to crawl the given forum site starting from its portal page and from the dummy time 2006-01-01 to 2007-01-01.

To differentiate the advantage of thread-wise strategy and the benefit of bandwidth control, we split our method into two methods: (1) thread-wise strategy (TWS); (2) thread-wise strategy + bandwidth control (TWS+BC).

Since the Curve-Fitting policy and Bound-Based policy
in [14] is the state-of-the-art approaches and are more relevant to our work, we also included them in the experiments. The original Bound-Based policy only crawl existing pages. We have tried our best to adapt the structure-driven-based approach to forum crawling by: 1) giving a new discovered URL the highest weight; and 2) relaxing the interval condition for adjusting refresh period and reference time to accommodate the high frequent update situation in forum sites.

We also introduced an oracle strategy for comparison. In the oracle strategy, every update activity of each page in the target site is supposed to be known exactly. Given the fixed bandwidth, the oracle policy can choose the pages with more new valuable information to visit. The oracle strategy is an ideal policy and the frame of reference for other methods.

### 6.2 Measurements

Following pervious work, we assume that the costs of visiting different pages are equal, and we measure the bandwidth cost as the total number of pages which are required to crawl in a given time period [14]. To evaluate the crawler integrally, we use the measurements in three aspects:

- **Bandwidth Utilization.** Suppose the bandwidth is fixed, bandwidth utilization is an important measure of crawler’s capability. This measurement is used to analyze if the crawler can make the best use of a limited bandwidth:

  \[ B = \frac{I_{\text{new}}}{I_B} \]  

  where \( I_B \) is the bandwidth cost defined as the total number of pages can be crawled in a time unit and \( I_{\text{new}} \) is the number of pages which contain new information comparing with the existing indexed repository.

- **Coverage.** When the crawler need to balance between fetching as much as possible valuable data and keeping previously-downloaded pages as fresh as possible, it is possible that it may not download all the required pages. Particularly, if a crawler wastes too much bandwidth to keep previously-downloaded pages fresh, it may not be able to crawl all new valuable data, and vice versa. To formalize the notion, we define the coverage:

  \[ \text{Cov} = \frac{I_{\text{crawl}}}{I_{\text{lat}}}, \]  

  where \( I_{\text{lat}} \) is the measurement of valuable information existing in the target site and \( I_{\text{crawl}} \) is the measurement of valuable information having been downloaded by crawler.

- **Freshness.** To capture ”how fast” we can fetch each post, we introduce the freshness. Suppose there are \( N \) elements of valuable information which we have downloaded. The freshness is defined as:

  \[ \text{Age} = \frac{1}{N} \sum_{i=1}^{N} \Delta t_i \]  

  where \( \Delta t_i \) represent the time period from its creating time to the time it is downloaded. If the element was created three day ago and we downloaded it one day ago, \( \Delta t_i \) is two day.

### 6.3 Warming Up Stage

In this paper we wish to analyze the ability of a crawler to balance between fetching new data and keeping data fresh. To be fair for all crawlers, we require all crawlers to begin with the portal page of the given site with a fixed bandwidth 3000 pages per day. The crawlers begin to crawl pages from dummy time 2007-01-01 and last about one year. To illustrate the performance changes of all crawlers in different time periods, we calculate the average performance in everyday in terms of the aforementioned three measurements and present the results in Fig. 7.

![Figure 7: One year performance of different crawlers in (a) Bandwidth Utilization, (b) Coverage, and (c) Freshness.](image)

Apparently, the performance changed observably in the
first 100 days and become stable after about 120 days. It can be explained as follows. Suppose there are $P_{old}$ posts existing before the crawler starts, $\Delta P$ new posts generated every day and the bandwidth is $B$ posts per day. At the $d^{th}$ day, there are about $P_{old} + \Delta P \cdot d$ posts existing in the target site. At the beginning, since $P_{old} \gg \Delta P \cdot d$, the crawler is required to download almost all posts belong to $P_{old}$. These are all new valuable information comparing with the indexed repository. This is why in the first 100 days the bandwidth utilization was approximate to 1, the coverage increased quickly and the freshness decreased quickly. We call this stage the warming up stage for the crawler. After about $P_{old}/(B - \Delta P)$ days, the crawler may finish downloading all old posts and begin to only focus on the posts belong to $\Delta P$ every day and the performance may become stable.

Whatever refresh strategies a crawler chooses, if it only assigns new valuable information with the highest weight, the length of the warming up stage will only depend on the number of old posts $P_{old}$, new post generation speed $\Delta P$ and the bandwidth $B$. Furthermore, if the bandwidth $B < \Delta P$, it means the bandwidth is too small to cover daily generated new posts. In this case, the crawler may not be able to mirror the old posts unless the forum site stops update.

In general, the oracle method always performs the best in all measurements and acts as an ideal method. Beside the oracle method, the TWS+BC performs significantly better than other methods. The average coverage per day for all methods will become to 100% after 100 days. But the freshness results are quite different for different methods. For the given bandwidth, the freshness of TWS+BC will decrease after 100 days. This is because TWS+BC can vacate enough buffer to catch up the update progress for the new posts every day after it finishes crawling all existing posts. The TWC can just keep the freshness stable because it may not have additional bandwidth to catch up the update progress. The bound-based and curve-fitting policies get very similar performance. The freshness of them all increases (note that the smaller the better for the freshness measure) since they cannot fetch new posts timely and downloading of these posts will always be delayed. In next section, we will evaluate them more in detail.

6.4 Comparison with other methods

A crawler for a commercial search engine is usually not allowed to restart at any time, the performance after its warming-up stage is thus more meaningful. We evaluated all methods with different bandwidths and all crawlers were required to start from the portal page of the given site. The crawlers crawled forum pages from dummy time 2006-01-01 and last about one year. We only calculate the average performance of different methods in the last 90 days from 2006-10-01 to 2006-12-31 and present the results in Fig. 8.

The curve-fitting policy and bound-based policy perform similarly on freshness and bandwidth utilization while curve-fitting policy performs slightly better on coverage. This is consistent with their original results in [14]. TWS is better than curve-fitting policy and bound-based policy on all measurements. This is because we can estimate the update frequency more accurately with thread-wise information. Furthermore, we can also avoid visiting duplicate posts in TWS and thus save considerable bandwidth.

TWS+BC is the best policy which further improves the performance apparently comparing with TWS. Although TWS can estimate the update frequency for list-of-thread pages and discussion thread pages relatively more accurately, it is still very hard to balance these two kinds of pages. In contrast, the bandwidth control policy is a good way to balance between fetching valuable data and refreshing downloaded pages. Since the number of thread pages is much larger than the number of list-of-thread pages, such a policy only slightly affects discussion thread pages but benefits the list-of-thread pages a lot. When the bandwidth is set to 3000 pages per day, the average freshness for TWS+BC is about 65 minutes while the average freshness for bound-based policy or curve-fitting policy is about 170 minutes and 165 minutes respectively. Thus TWS+BC is about 2.6 times faster than bound-based policy or curve-fitting policy and thus is capable of downloading new posts timely. At the same time, it can also achieve a high coverage ratio comparing with other methods. To get more insights on this problem, we will evaluate these two kinds of pages separately in next section.

6.5 List-of-thread pages and discussion thread pages

Given a fixed bandwidth 3000 pages per day, we evalu-
ated the performance on list-of-thread pages and discussion thread pages separately and show the results in Table 2.

From the table, we can see that TWS can improve the performance for both two kinds of pages. The freshness of both list-of-thread pages and discussion-thread pages decreases significantly in Table 2 comparing with curve-fitting policy and bound-based policy. This is because TWS leverages more information and can estimate the update frequency for both two kinds of pages more accurately.

TWS+BC further improves the performance for list-of-thread pages comparing with TWS. The freshness of list-of-thread pages decreases significantly in Table 2 while the freshness of discussion-thread pages keeps the same. It confirms our previous assumption that it is very hard to balance these two kinds of pages only by estimating their update frequencies. Bandwidth control can assign the right ratio according to the real update numbers for different kinds of pages.

7. CONCLUSION

Realizing the importance of forum data and the challenges in forum crawling, in this paper, we proposed a thread-wise strategy for incremental crawling of web forums. Instead of treating each individual page independently, as did in most existing methods, we have made two improvements. First, we analyze each discussion thread as a whole by concatenating multiple pages belong to this thread together. And then we take into account user behavior-related statistics in each discussion thread, for example, the number of replies, the frequency of replies, and the time stamp of each reply. Such information is of great help for developing an efficient recrawl strategy. Second, we balance discovering new pages and keeping existing pages fresh by introducing a bandwidth control policy for thread pages and list-of-thread pages. To evaluate the proposed crawling strategy, we conducted extensive experiments on 18 forums, compared it with several state-of-the-art methods, and evaluated it in various situations, including different stages in one year’s crawling simulation, different bandwidths, and different kinds of pages. Our method outperforms state-of-the-art methods in terms of bandwidth utilization, coverage, and freshness in all situations. Based on this method, we developed a highly efficient crawler which is 2.6 times faster than existing methods while it can also achieve a high coverage ratio.

8. REFERENCES

[1] Internet forum.


