Incorporating Site-Level Knowledge for Incremental Crawling of Web Forums: A List-wise Strategy*

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 is usually inefficient in crawling forum sites because of the different characteristics between forum sites and general websites. Instead of treating each individual page independently, we propose a list-wise strategy by taking into account the site-level knowledge. Such site-level knowledge is mined through reconstructing the linking structure, called sitemap, for a given forum site. With the sitemap, posts from the same thread but distributed on various pages can be concatenated according to their timestamps. After that, for each thread, we employ a regression model to predict the time when the next post arrives. Based on this model, we develop an efficient crawler which is 260% faster than some state-of-the-art methods in terms of fetching new generated content; and meanwhile our crawler also ensure a high coverage ratio. Experimental results show promising performance of *Coverage*, *Bandwidth utilization*, and *Timeliness* of our crawler on 18 various forums.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - clustering, information filtering

General Terms

Algorithms, Performance, Experimentation

Keywords

Web 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[2] for both product recommendation and business intelligence, integrating travel notes for travel recommendations[3], and extracting question-answer pairs for QnA service [8], etc.

Fetching forum data (mainly user created posts) from various forum sites is the fundamental step of most related web applications. To satisfy the application requirements, an ideal forum crawler should make a tradeoff between updating existing discussion threads (*i.e., completeness*) and getting new discussion threads in time (*i.e., timeliness*). *Completeness* refers to identify updated discussion threads and download the newly published content. This is important to applications like online QnA services. As pagination is widely used in web forums, the posts of a question and its best answers are most likely to be distributed in different pages. In this case, a forum crawler should fetch all the pages from a discussion thread to ensure not missing any best-answers. *Timeliness* refers to efficiently discover and download newly published discussion threads. This is also important because many web applications need fresh information to response quickly to news events and new product releases. For example, users may be interested in knowing the fair and square feedbacks of iPhone shortly after it was released; and this creates the need to rapidly gather consumer comments of iPhone from various web forums.

However, most existing crawling strategies designed for general websites may not be suitable for forum crawling, and cannot well satisfy the above requirements. This is because web forums have some unique characteristics. (I) Content information in forum sites is organized in form of *list*. Because forum content, such as a long discussion thread, is usually too huge to be displaced in one single page, most web forums adopt pagination to divide long content into multiple pages, as shown in Fig. 1. (II) In web forums, pages can be categorized into different types according to their functions. The most two common page types in web forums are *index-of-thread* and *post-of-thread*. As shown in Fig. 1(a), a list of *index-of-thread* pages (called *index list*) shows the basic information (such as title, author, and creation-time) of all the discussion threads in a discussion board or sub-boards; and as shown in Fig. 1(b), a list of *post-of-thread* pages (called *post list*) consists of the detailed content of all the posts from the corresponding thread. Unfortunately, these two characteristics are ignored by most existing crawlers. They simply

*This work was performed when the 3^{rd} and the 4^{th} authors were interns in Microsoft Research Asia.

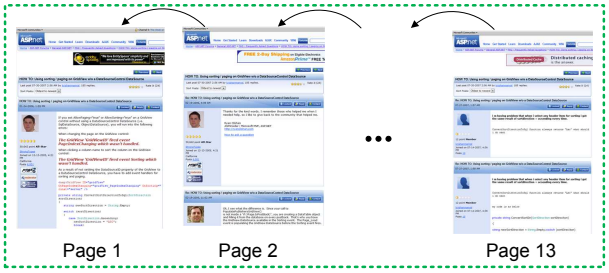
[1]http://en.wikipedia.org/wiki/Internet_forum

[2]www.dpreview.com

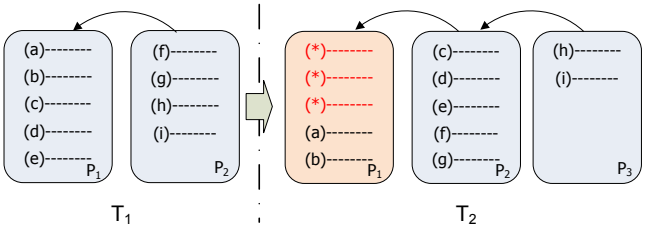
[3]www.tripadvisor.com

KDD'09, June 28–July 1, 2009, Paris, France.

(a) An exemplary *index list* and its *index-of-thread* pages



(b) An exemplary *post list* and its *post-of-thread* pages



(c) Data growth of an *index list*

Figure 1: An illustration of (a) Index-of-thread pages[3] in sub-board "Getting Started", (b) Post-of-thread pages[4] in thread "HOW TO: Using sorting / paging on GridView w/o a DataSourceControl DataSource", and (c) Data growth of an *index list*.

treat each individual page in a forum as a single object, and optimize the crawling schedule for each page independently. This leads to the following two problems in forum crawling:

- Incapable of properly predicting the re-crawling schedule. As the list-wise structure is missed, a crawler doesn't know which content is indeed new. For example, as shown in Fig. 1(c), supposing in T_1 the *index* list contains two pages while in T_2 it has three pages with some new content added at the beginning of page P_1, a general crawler thinks all the three pages are updated. In this case, it will continually re-crawl the page P_2 and P_3 although in fact there is no new information in these pages.
- Incapable of finding newly published discussion threads in time. In forums, timely updating *index lists* is very crucial to discover new discussion threads. However, a general crawler doesn't know which pages are from *index lists* but just equally treat all the pages. Therefore, it cannot optimize the bandwidth to meet the *timeliness* requirement of forum crawling.

Towards a crawler optimized for forum crawling, in this paper, a list-wise incremental crawling strategy is proposed based on the mining of the site-level knowledge from target forums. First, unlike general crawlers which optimize crawling schedule for every individual page, in our strategy we take a *list* (either *index list* or *post list*) as the basic unit for crawling schedule optimization. Second, to understand the list-structures and the types-of-list of a forum, we propose a set of offline mining algorithms which try to first discover the site-level knowledge from the target sites and then incorporate the knowledge to optimize the crawling schedules. The contributions of this paper are briefly highlighted as follows:

- Site-Level Knowledge Mining. The site-level knowledge is this paper mainly refers to the *sitemap* of a forum site [4, 15]. A sitemap is a graph-based representation of the organization structure of a web forum. It tells us how many kinds of pages are there in the target forum and how these pages are linked with each others. With the instruction of a sitemap, we concatenate pages to reconstruct *index lists* and *post lists*.
- List-Wise Schedule Optimization. Based on the reconstructed lists, we first propose an algorithm to discover the timestamps of each thread, as well as of each post in a given thread. With the timestamps, some site-level statistics are then utilized to estimate the update frequency and longevity for each list. It should be noticed that such site-level statistics are robust as they suffer little from the noises existing in some individual pages. At last, a regression model is employed to effectively combine various statistics to predict the update frequency of each list.
- Bandwidth Control of Different Types of Lists. As we have argued, timely refreshing of *index lists* is very crucial to ensure the *timeliness* of a forum crawler. However, *post lists* are dominant in most forums and *index lists* are just a small fraction. Thus, the updating of *post lists* consumes most bandwidth and the refreshing of *index lists* is prone to be delayed. As we can identify *index lists* and *post lists*, we also propose an efficient bandwidth control strategy to balance the discovering of new content and the updating of existing content.

In the experiments, the proposed solution has been evaluated on a mirrored data set containing 990,476 pages and 5,407,854 individual posts, published from March 1999 to June 2008. It showed the crawler with our list-wise strategy is 260% faster than some state-of-the-art methods in terms of fetching the new generated content, and meanwhile our crawler still ensure a higher coverage ratio.

This paper is organized as follows. Related research efforts are reviewed in Section 2. In Section 3, we first briefly describe the framework of our approach and then explain each components in detail. Experimental evaluations are reported in Section 4, and we draw some conclusions in the last section.

2. RELATED WORK

To the best of our knowledge, little existing work in literatures has systematically 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 [2,3] first investigated the dynamic web and treated 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.

Later work improved the above work and optimized the crawling process by prioritizing the pages. 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:

(I) *How often the content of a page is updated by its owner.* Coffman et. al. [7] analyzed the crawling problem theoretically. Cho et. al. [6] proposed several methods based on the 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 [3] 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 because the content are all generated by forum users. Edwards et. al. [10] 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. Furthermore, we argue that some other factors, such as time intervals of the latest 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. [1] tried to determine the weight of each page based on some strategies similar to PageRank. Wolf et. al. [16] assigned the weight of each page based on the embarrassment metric of users' search results. In the user-centric crawling strategy [13], 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 insufficient. Second, both static rank 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-like static rank 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 author deletes it manually. Before we get these post information, it is very hard to predict their importance. However, once 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 [5, 9, 11] or some labeled examples [14] 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. [12] introduced the longevity to determine revisiting frequency of each web page. However, the information longevity in forums is useless since once a post never disappears unless being deleted. 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.

Realizing the importance of forum data and the challenges in forum crawling, Cai et al. [4, 15] 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

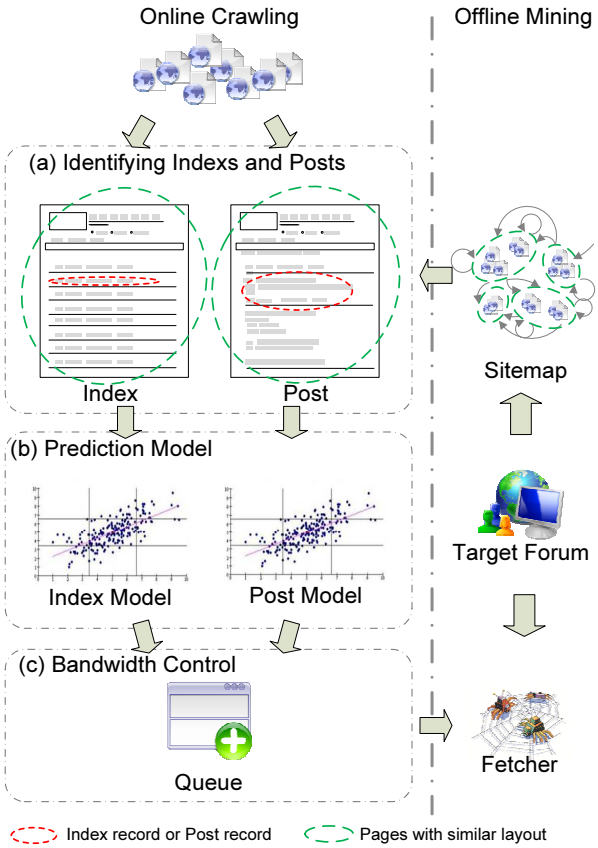


Figure 2: The flowchart of the proposed solution for incremental forum crawling.

as much as possible valuable data, but left the problem of refreshing previously downloaded pages [12] untouched.

All the existing methods ignore the tradeoff between discovering new threads and refreshing existing threads. Intuitively, *index-of-thread* pages should get higher update frequency than *post-of-thread* pages because any users' update activities will change *index-of-thread* pages. However, because the number of *post-of-thread* pages is much larger than *index-of-thread* pages, *index-of-thread* pages may be overwhelmed by a mass of *post-of-thread* pages even if only a small percentage of *post-of-thread* pages are very active. These *post-of-thread* pages will occupy most bandwidth, and as a result new discussion threads cannot be downloaded efficiently. Unfortunately, few of existing methods has taken this issue into account. We will show its importance in the experiment part.

3. OUR SOLUTION

In this section, we first describe the system overview of the proposed solution, and then introduce the details of each component.

3.1 System Overview

The proposed solution consists of two parts, *offline mining* and *online crawling*, as shown in Fig. 2.

The offline mining is to learn the sitemap of a given forum site with a few pre-sampled pages from the target site. Through analyzing these sampled pages, we can find out how many kinds of pages are there in that site, and how these pages are linked with each others [4]. This site-level knowledge is then employed in the online crawling process.

The online crawling consists of three steps: (a) identifying *index lists* and *post lists*; (b) predicting the update frequency of a list using a regression model; and (c) balancing bandwidth by adjusting the numbers of various lists in the crawling queue, as shown in the left part of Fig. 2. Given a new crawled page, we first identify if it is an *index-of-thread* page or a *post-of-thread* page according to its layout information. Since an *index list* or *post list* consists of multiple pages, to reconstruct a list, we concatenate corresponding pages following the detected pagination links [15]. Then, for each reconstructed list, we extract the timestamps of the records (either posts or thread entities) containing in that list. Several statistics are then proposed based on the timestamps to characterize the growth of a list; and a regression model is trained to predict the arrival time of the next record of that list. Finally, given a fixed bandwidth, we balance the numbers of *index*/*post* lists, to satisfy the requirements of both *completeness* and *timeliness*.

3.2 List Reconstruction

The *sitemap* knowledge is an organization graph of a give forum site, which is the fundamental component of the proposed crawling method in this paper. The details of its generation algorithm can be found in [4, 15], here we just briefly describe it as follows. To estimate the *sitemap*, we first sample a few pages from the target site. Because the sampling quality is crucial to the whole mining process, to diversity the sampled pages in terms of page layout and to retrieve pages at deep levels, we adopt a combined strategy of breadth-first and depth-first using a double-ended queue. In the implementation, we try to push as many as possible unseen URLs from each crawled page to the queue, and then randomly pop a URL from the front or the end of the queue for a new sampling. In practice, it was found that sampling a few thousands pages is sufficient to reconstruct the *sitemap* for most forum sites. After that, pages with similar layout structures are further clustered into groups using the single linkage algorithm, as marked with blue nodes in Fig. 3. In our approach, we utilize the repetitive regions to characterize the layout of each page. Repetitive regions are very popular in forum pages to present data records stored in a database. Considering that two similar pages usually have different numbers of advertisements, images, and even some complex sub-structure embedded in user posts, the repetitive region-based representation is more robust than the whole DOM tree [17, 18]. Finally, all possible links among various page groups are established, if in the source group there is a page having an out-link pointing to another page in the target group.

After that, a classifier is introduced to identify the node of *index-of-thread* pages or *post-of-thread* pages in a *sitemap*. The classifier is built by some simple yet robust features. For example, the node of *post-of-thread* pages always contains the largest number of pages in *sitemap*, and the *index-of-thread* pages should be the parents of *post-of-thread* pages, as marked with the red rectangle or green rectangle in Fig.3. The classifier is site-independent and can be used in different forum sites. We then map each individual page into one of the nodes by evaluate its layout similarity such as $sim(p_{new}, N_{index})$ and $sim(p_{new}, N_{post})$; and δ is the threshold for decision. We concatenate all the individual pages belonging to the same *index list* or *post list* together. The details of this process is described in Algorithm 1. The above process is fully automatic. With the concatenated *index lists* or *post lists*, our crawling method becomes a list-wise strategy rather than page-level strategies used by most existing general crawlers.

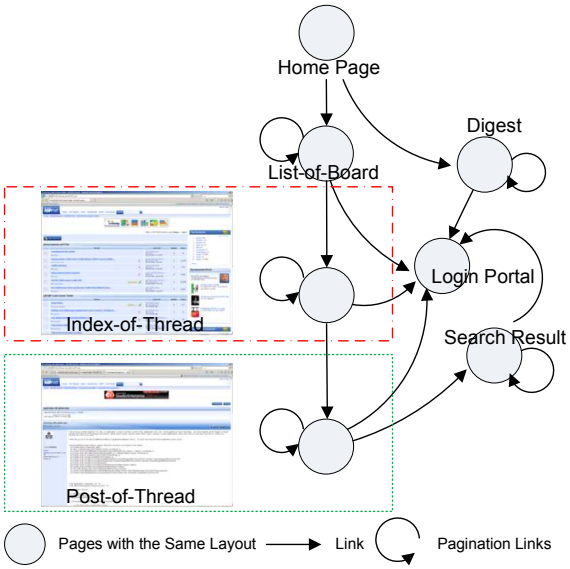


Figure 3: An illustration of a sitemap.

Algorithm 1 The algorithm for constructing *index lists* and *post lists*. Suppose $sitemap(N, E)$ is the sitemap knowledge with corresponding nodes N and edges E, N_{index} is the node of *index-of-thread* pages, N_{post} is the node of *post-of-thread* pages.

```
Suppose there is a new page p_new.
Parse the page p_new and get the pagination out-links list OutLinks_new.
if sim(p_new, N_index) < δ then
    for all page p_i in N_index, and OutLinks_i is the pagination out-links list of p_i do
        If (1) ∃link_j ∈ OutLinks_i and p_new = link_j, or (2) ∃link_k ∈ OutLinks_new and p_i = link_k.
        Concatenate p_new into the index list of p_i.
    end for
else if sim(p_new, N_post) < δ then
    for all page p_i in N_post, and OutLinks_i is the pagination out-links list of p_i do
        If (1) ∃link_j ∈ OutLinks_i and p_new = link_j, or (2) ∃link_k ∈ OutLinks_new and p_i = link_k.
        Concatenate p_new into the post list of p_i.
    end for
end if
```

3.3 Timestamp Extraction

To predict the update frequency of a list more accurately, we need to first extract the time information of each record (thread creation time in *index lists* or post time in *post lists*). It is difficult to extract the correct time information due to the existence of noisy time records in forum pages, such as users' registration time, last login time and so on. In Fig.4, the time contents in orange rectangle are all noisy content while only the purple one is the correct time information.

There are three steps to extract the time information. We first get the timestamp candidates whose content is a short one and contains digit string such as mm-dd-yyyy or dd/mm/yyyy, or some specific words such as Monday, Tuesday, January, February, etc. Second, we align the html elements containing time information into several groups based

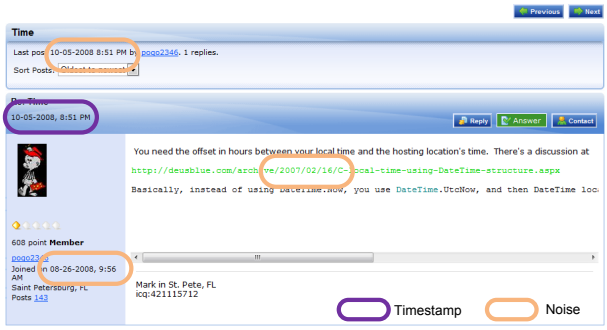


Figure 4: Example timestamps in a page.

Algorithm 2 The algorithm of extracting timestamp DOM path from list L. Suppose $tsSet$ is a set of all timestamp candidates for the pages in L and there are N unique DOM paths for these candidates in $tsSet$.

for all DOM path $path_i$, $1 \leq i \leq N$ **do**
  Set $SeqOrd = 0$ and $RevOrd = 0$.
  for all page p_j in L **do**
    Get all cadidates $tsList_j$ in p_j whose DOM path is $path_i$ and sort them in their appearance order.
    for all time candidate tc_k in $tsList_j$, $1 \leq k \leq M$ and M is the length of ts_j **do**
      if tc_k is earlier than tc_{k+1} **then**
        $SeqOrd++$
      else if tc_k is later than tc_{k+1} **then**
        $RevOrd++$
      end if
    end for
  end for
  Set $Order_i = \frac{|SeqOrd - RevOrd|}{SeqOrd + RevOrd}$
end for
Let $c = argmax_i\ Order_i$
Return $path_c$

on their DOM path since the timestamp in each record should have the same DOM path in HTML document. Finally, since the records are generated in sequence, the timestamps should satisfy a sequential order. This helps filter noisy time records and extract the correct information. The details of this process is described in Algorithm 2.

3.4 Prediction Model

The latest replies in each thread can be easily found by revisiting *post lists*. Similarly, the new discussion threads can be discovered by revisiting *index lists*. The problem of incremental crawling of web forums is converted to how to estimate the update frequency and predict when the next new record of a list (*post list* or *index list*) arrives. Based on the timestamp of each record, we propose several features to help predict the update frequency and describe them in Table.1.

Furthermore, for *index lists*, we analyze the average thread activity in forum sites. We process all the threads and calculate their active time by checking the time of the first post and the last post in each thread. The result is shown in Fig. 5. The figure represents the percentage of threads with different active time. From the figure we can see the percetage of active thread drops significantly when the active time becomes longer. More than 40% threads keep active no longer than 24 hours and 70% threads are no longer than 3 days. This is one of the major reasons why forum incremental crawling strategy is different from traditional incremental crawling strategies. Once a thread is created, it usually becomes static after a few days when there is no discussion activity. Thus we introduce a state indicator to avoid bandwidth waste. Suppose there is no discussion activity for Δt_{na} time since the last post. We compute the standard deviation of time interval Δt_{sd} by $\Delta t_{sd} = \sqrt{1/(N-1) \cdot \sum_{i=1}^{N-2} (\Delta t_i - \Delta t_{avg})^2}$, where 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 *index lists* only.

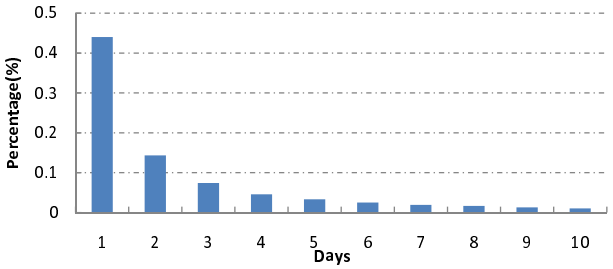


Figure 5: The thread activity analysis.

To combine these factors together, we leverage a linear regression model which is a lightweight computation model and is efficient for online processing.

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

For each forum site, we train two models $F_{list}(x)$ and $F_{post}(x)$ for *index lists* and *post lists* separately. The two models are kept updated during the crawling process for the new crawled pages. In practice, we update the two models every two months. By setting $x_0 = 1$, we can get the corresponding $\mathbb{W}$ by Equation 2.

$$\mathbb{W} = (\mathbb{X}^T \cdot \mathbb{X})^{-1} \cdot \mathbb{X}^T \cdot \mathbb{Y} \tag{2}$$

In the crawling process, we predict the new coming records of each list by $\Delta I = (CT - LT)/F(x)$, where CT is the current time and LT is the last revisit time (by crawler). We use the predicted ΔI to schedule the crawling process. Though a list may contain multiple pages, we do not need to revisit all of them. We sort the pages in each list based on the timestamps of the records in each page. For an *index list*, if there are Num_{list} records in each *index-of-thread* page, and we only need to revisit the top $\Delta I_{list}/Num_{list}$ pages, where $\Delta I_{list} = (CT - LT)/F_{list}(x)$. For a *post list*, we need to revisit the last page or the new discovered pages.

3.5 Bandwidth Control

As we have discussed in previous sections, we need to make a tradeoff between discovering new pages and refreshing existing pages. The number of *post-of-thread* pages is much larger than *index-of-thread*. Even if only a small percentage of *post-of-thread* pages are very active, *index-of-thread* pages may be overwhelmed by a mass of *post-of-thread* pages, which will occupy most bandwidth. In this case, we need to allocate a dedicated bandwidth for *post-of-thread* pages since we can only get new discussion threads from them. We introduce a hyper bandwidth control strategy to balance the bandwidth between two kinds of list. The *ratio* between *index lists* and *post lists* can be defined as:

$$ratio \propto \frac{N_{index}}{N_{Post}} \tag{3}$$

Table 1: The proposed features in prediction model.

Feature	Description
$\Delta t_1, \Delta t_2, \Delta t_3, \Delta t_4, \Delta t_5$	In contrast to leveraging the timestamp directly, we leverage the time intervals between each two consecutive timestamps which represent the recent update frequency of a list.
Δt_{avg}	The average time interval represents the list update history. Because the latest five time intervals may be affected by some accidents, the average time interval of consecutive records in a list helps on smoothing the result and tolerates these accidents.
Δt_{site}	At the beginning of each list, few information can be used in predicting its update frequency. We leverage the average time interval in the current forum site to approximate its update frequency.
len	The length of a list can represent its hotness which may attract more users. Thus it may also affect the update frequency of this list.
$ct_1, ct_2, ct_3, \ldots, ct_{24}$	The experiment result in [3] also shows the update frequency of web pages is highly dependent on the current time. We represent the current time in 24 hours by a vector with 24 dimensions. For example, we represent 3 PM by setting $ct_{15} = 1$ and others to zero.
Δt_{hour}	Since the update frequency of web pages is highly dependent on daytime/nighttime, we split one day into 24 hours and leverage the average time interval of the whole forum site in current time span.
$cd_1, cd_2, cd_3, \ldots, cd_7$	Similar to the feature of current time. We also represent the current day in a week by a vector with 7 dimensions. For example, we represent Wednesday by setting $cd_3 = 1$ and others to zero.
Δt_{day}	Since the update frequency of pages is also dependent on working days/weekend days, we split one week into 7 days and leverage the average time interval of the whole forum site in current time span.
$s_1, s_2, s_3, \ldots, s_{15}$	Ideally, we can estimate update frequency of current list by checking similar lists. To achieve this goal, we first represent the state of current list via its last five time intervals and then clustering them into 15 clusters based on their Euclidean distances. For each new list, we assign it to one of the states with the smallest Euclidean distance. We represent the 15 states by a vector with 15 dimensions. For example, we represent state 5 by setting $s_5 = 1$ and others to zero.

where given a time period Δt, N_{index} is the number of new discussion threads within time Δt and N_{Post} is the number of new posts arriving within time Δt. We use the average *ratio* in history to balance the bandwidth between two lists.

4. EXPERIMENTS

Different from previous 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 starting from its portal page. To have a thorough evaluation, a crawling task needs to last for one year or even longer so that we can measure the crawling performance using different metrics at different periods, for example, the warming up stage and the stable stage. This creates the need to build a simulation platform to compare different algorithms because the real-time crawling cannot be repeated for multiple crawling approaches. Before we describe the experimental details, we first introduce the experimental setup and the evaluation metrics.

4.1 Experimental Setup

To evaluate the performance of our system on various situations, 18 forums were selected in diverse categories (including bicycle, photography, travel, computer technique, and some general forums) in the experiments, as listed in Table 2. The average length of service of 18 sites is about 4.08 years.

As we wish to evaluate different methods in a long time period (about one year) under several different bandwidth situations while we still want the evaluation to be repeated under the same environment to fairly compare different approaches. Because it is impractical to repeat a long lasting crawling process for many times on real sites, we built a simulation platform to facilitate our evaluation. Typically, a forum hosting server organizes forum data using a backend database, and dynamically generates forum pages using page templates. To build the simulation platform for a forum, we need to first mirror the forum site by downloading all the forum pages, then parse the forum pages in a reverse engineering manner and store the parsed forum posts in a database. Thereafter, we can simulate the response to a downloading request by regenerating a forum page according to the requested URL address and the crawling time.

Table 2: Web Forums in the Experiments

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

More precisely, we first mirrored the 18 forum sites using a customized commercial search engine crawler. The crawler was configured to be domain-limited and depth-unlimited. For each site, the crawler started from its 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 990,476 pages and 5,407,854 individual posts, from March 1999 to June 2008. Using manually wrote data extraction wrappers, we parsed all the pages and stored all the data records in a database.

The basic behavior of this simulation platform is to response for a URL request associated with a timestamp. We wrote 18 page generators for 18 forum sites to simulate the responses to requests. For any requested URL, since all the corresponding records can be accessed from the database, we can generate a HTML page with the same contents, layout, and related links as the one in the real site. Here we make an assumption that a post will never be deleted after it is generated. Based on this simple yet reasonable assumption, we can easily figure out at any given time whether a record exists based on its post time and how records are organized based on the forum's layout, and therefore be able to restore a snapshot of a forum site to the given time.

This simulation platform was used in all the following crawling experiments, and provided a fair experimental environment to each crawling approach. Assuming a fixed bandwidth, each crawler was required to crawl the given forum site starting its portal page and from the dummy time period 2006-01-01 to 2007-01-01.

4.2 Methods Compared in the Experiments

To differentiate the advantage of list-wise strategy and the benefit of bandwidth control, we implemented two variants of our method: (1) list-wise strategy (LWS); (2) list-wise strategy + bandwidth control (LWS+BC).

Since the Curve-Fitting policy and Bound-Based policy in [12] are 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 an upper bound for other crawlers.

4.3 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 [12]. To evaluate the overall crawling performance for each approach, we measure from the following three aspects:

- Bandwidth Utilization. If 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:

$$B = \frac{I_{new}}{I_B} \tag{4}$$

  where I_B is the bandwidth cost defined as the total number of pages crawled in a time unit and I_{new} is the number of pages containing new information compared to the existing indexed repository.

- Coverage. A crawler needs to balance between fetching new pages and refreshing existing pages. If a crawler wastes too much bandwidth to refresh previously downloaded pages, it may not be able to crawl all new requested pages, and vice versa. To measure the issue, we define *coverage* as follows:

$$Cov = \frac{I_{crawl}}{I_{all}} \tag{5}$$

  where I_{all} is the measurement of valuable information existing in the target site and I_{crawl} is the measurement of valuable information having been downloaded by crawler.

- Timeliness. To measure if we can fetch each post "timely", we introduce *timeliness*. Suppose there are N elements of valuable information which we have downloaded. The *timeliness* is defined as:

$$T = \frac{1}{N}\sum_{i=1}^{N}\Delta t_i \tag{6}$$

  where Δt_i represents the time period from its updating time to its downloading time. If the element was updated three day ago and we downloaded it one day ago, Δt_i is two day. This value is smaller the better.

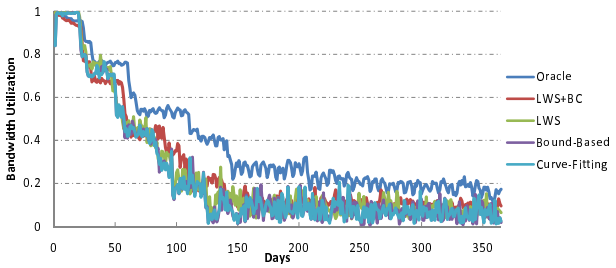
In the forum crawling task, different from general web sites, once a post is submitted, the post will always exist until it is manually deleted by the creator or administrator. In this paper, we use the number of unique posts to measure the valuable information in forum sites. In the definition of *coverage*, I_{all} should be the number of unique posts existing in the target forum site and I_{crawl} should be the number of unique posts having been downloaded. In the definition of *timeliness*, Δt_i is the time period from a post's creating time to its downloading time. If we download a post one day after the post was created, Δt_i is one day.

4.4 Warming Up Stage

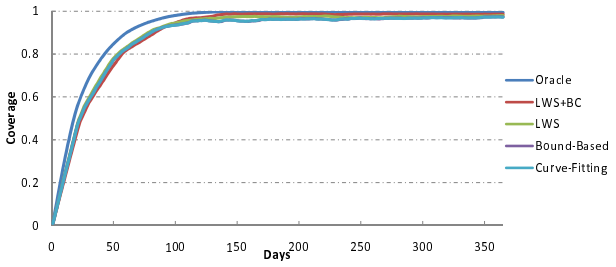
To make a fair evaluation for all crawlers, we require all the crawlers to start from the portal page of each site with a fixed bandwidth 3000 pages per day. The crawlers begin to crawl pages from the dummy time 2007-01-01 and last about one year. To illustrate the performance changes of all crawlers in different time periods, we calculate the average performance in everyday in terms of the aforementioned three measurements and present the results in Fig. 6.

From the figure we can see the performance changes apparently in the first 100 days and become stable after about 120 days. This is due to the so called warming up stage during which a crawler needs to first mirror the existing pages and after that it can download new pages. Suppose there are P_{old} posts existing before the crawler starts, ΔP new posts will be generated every day and the bandwidth allows the crawler to crawl 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 belonging to P_{old}. These are all new valuable information compared to the indexed repository. This is why in the first 100 days the bandwidth utilization was approximate to 1, the *Coverage* increases quickly and the *Bandwidth Utilization* decreases quickly. We call this stage the *warming up stage* for the crawler. After about $P_{old}/(B - \Delta P)$ days, the crawler may have finished downloading all old posts and begins to only focus on the posts belonging to ΔP every day and the performance becomes stable.

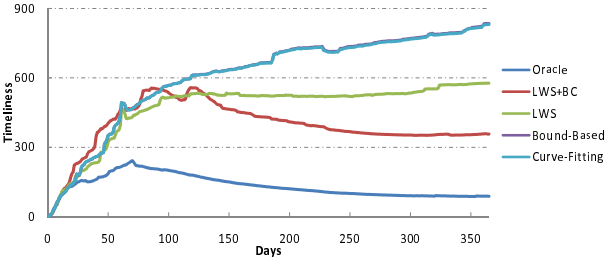
Whatever refresh strategy a crawler chooses, if it only assigns new valuable information with the highest weight,



(a) Bandwidth Utilization



(b) Coverage

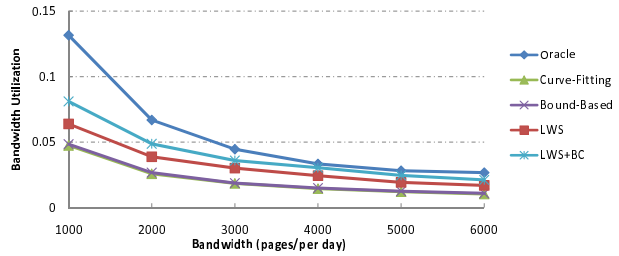


(c) Timeliness

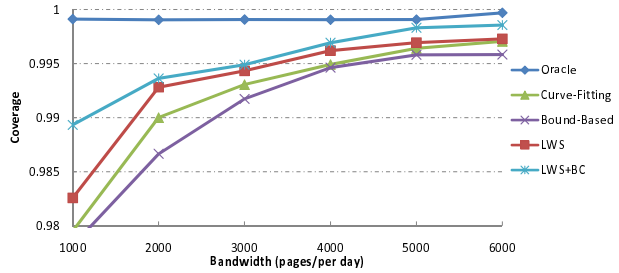
Figure 6: One year performance of different crawlers in (a) Bandwidth Utilization, (b) Coverage, and (c) Timeliness.

the length of the *warming up stage* will only depend on the number of old posts P_{old}, new post generation speed ΔP 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 generating new posts.

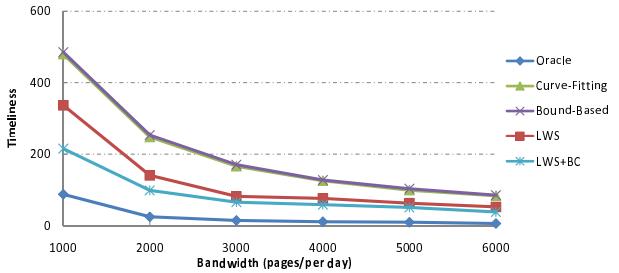
In general, the oracle method always performs the best in all measurements and acts as an ideal method. Beside the oracle method, the LWS+BC performs significantly better than other methods in terms of *timeliness*. The average *coverage* per day for all methods becomes to 100% after 100 days. This is because the bandwidth is enough and these methods can archive most historical records. But the performance of these methods on fetching daily new records are different as shown in *timeliness* results. For the given bandwidth, the *timeliness* of LWS+BC will decrease after 100 days. This is because LWS+BC can save enough buffer to catch up the update progress for new posts after it finishes crawling all existing posts. The LWS can just keep the *timeliness* stable because it does not have additional bandwidth to catch up the update progress. The bound-based and curve-fitting policies get very similar performance. The *timeliness* of them all increases (note that the smaller the better for the *timeliness* measure) since they cannot fetch new posts timely and thus delay the downloading of new posts. We will analyze them in next section.



(a) Bandwidth Utilization



(b) Coverage



(c) Timeliness

Figure 7: The performance of different crawlers for different bandwidth in (a) Bandwidth Utilization, (b) Coverage, and (c) Timeliness.

4.5 Comparison with different methods

A crawler for a commercial search engine is usually not allowed to frequently restart. The performance after its warming-up stage is thus more meaningful. We evaluated all methods under various bandwidth conditions and all crawlers were required to start from the portal page of the given site. The crawlers start from the dummy time 2006-01-01 and last about one year. We only calculate the average performance of different methods for the last 90 days from 2006-10-01 to 2006-12-31 and present the results in Fig. 7.

The curve-fitting policy and bound-based policy perform similarly on *timeliness* and bandwidth utilization while curve-fitting policy performs slightly better on coverage. This is consistent with their original results in [12].

LWS is better than curve-fitting policy and bound-based policy on all measurements, because we can estimate the update frequency more accurately with list-wise information. Furthermore, we can also avoid visiting all the pages in a list and thus save considerable bandwidth.

LWS+BC is the best policy which further improves the performance apparently compared to LWS. Though the average *coverage* of different methods seems very close (from 0.98 to 1.0), the actual gap is relatively large when it multiplies 1 million pages or 5 million posts. And the gap may become even larger when we continue to crawl more sites or the bandwidth becomes more limited, as shown in the

Table 3: The performance differences between *index lists* and *post lists*. We use "BU" as "Bandwidth Utilization" here for short.

Methods	*Index lists*			*Post lists*		
	BU	C	T	BU	C	T
BB	0.0013	0.9925	171	0.0173	0.9917	170
CF	0.0012	0.9936	167	0.0174	0.9931	166
LWS	0.0025	0.9971	**73**	0.0276	0.9943	**81**
LWS+BC	0.0030	0.9989	**28**	0.0329	0.9949	**65**
Oracle	0.0061	1	2	0.0415	0.9990	16

trends of Fig. 7(b). Although LWS can estimate the update frequency for *index lists* and *post lists* relatively more accurately, it is still very hard to balance these two kinds of pages. In contrast, the bandwidth control policy is more effective to balance between fetching valuable data and refreshing downloaded pages. Such a policy only slightly affects *post-of-thread* pages but benefits the *index-of-thread* pages a lot. When the bandwidth is set to 3000 pages per day, the average *timeliness* for LWS+BC is about 65 minutes while the average *timeliness* for bound-based policy or curve-fitting policy is about 170 minutes and 165 minutes respectively. This shows that LWS+BC is 260% faster than the bound-based policy or curve-fitting policy and thus is capable of downloading new posts timely. Meanwhile, it also achieve a high coverage ratio compared to other methods. To get more insights to this problem, we evaluate these two kinds of pages separately in the next section.

4.6 Index Lists and Post Lists

Given a fixed bandwidth of crawling 3000 pages per day, we evaluated the performance for *index lists* and *post lists* separately and showed the results in Table 3.

From the table, we can see that LWS improves the performance for both two kinds of pages. The *timeliness* of both *index-of-thread* pages and *post-of-thread* pages decreases significantly in Table 3 compared to the curve-fitting policy and bound-based policy. This is because LWS leverages more information and can estimate the update frequency for both two kinds of pages more accurately.

LWS+BC further improves the performance for *index-of-thread* pages compared to LWS. The *timeliness* of *index-of-thread* pages decreases significantly in Table 3 while the *timeliness* of *post-of-thread* pages remains the same. It confirms our previous assumption that bandwidth control can assign the right ratio according to the real update numbers for different kinds of pages.

5. CONCLUSION

Realizing the importance of forum data and the challenges in forum crawling, in this paper, we proposed a list-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 *index list* or *post list* as a whole by concatenating multiple pages belong to this list together. And then we take into account user behavior-related statistics, for example, the number of records and the timestamp of each record. Such information is of great help for developing an efficient recrawl strategy. Second, we balance discovering new pages and refreshing existing pages by introducing a bandwidth control policy for *index lists* and *post lists*. 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 under various situations, including different stages through one year's crawling simulation, different bandwidths, and different kinds of pages. Experimental results show that our method outperforms state-of-the-art methods in terms of bandwidth utilization, coverage, and timeliness in all situations. The new strategy is 260% faster than existing methods and meanwhile it also achieves a high coverage ratio.

6. REFERENCES

[1] R. Baeza-Yates, C. Castillo, M. Marin, and A. Rodriguez. Crawling a country: Better strategies than breadth-first for web page ordering. In *Proc. of WWW*, 2005.

[2] B. E. Brewington and G. Cybenko. How dynamic is the web? *Computer Networks*, 2000.

[3] B. E. Brewington and G. Cybenko. Keeping up with the changing web. *Cumputer*, 2000.

[4] R. Cai, J.-M. Yang, W. Lai, Y. Wang, and L. Zhang. iRobot: An intelligent crawler for Web forums. In *Proc. of WWW*, pages 447–456, April 2008.

[5] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific web resource discovery. *Computer Networks*, 1999.

[6] J. Cho and H. Garcia-Molina. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems*, 28(4), 2003.

[7] E. Coffman, Z. Liu, and R. R. Weber. Optimal robot scheduling of web search engines. *Journal of scheduling*, 1, 1998.

[8] G. Cong, L. Wang, C.-Y. Lin, Y.-I. Song, and Y. Sun. Finding question-answer pairs from online forums. In *Proc. of SIGIR*, pages 467–474, July 2008.

[9] M. Diligenti, F. M. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using content graphs. In *Proc. of VLDB*, 2000.

[10] J. Edwards, K. S. McCurley, and J. A. Tomlin. An adaptive model for optimizing performance of an incremental web crawler. In *Proc. of WWW*, 2001.

[11] F. Menczer, G. Pant, P. Srinivasan, and M. E. Ruiz. Evaluating topic-driven web crawlers. In *Proc. of SIGIR*, 2001.

[12] C. Olston and S. Pandey. Recrawl scheduling based on information longevity. In *Proc. of WWW*, pages 437–446, April 2008.

[13] S. Pandey and C. Olston. User-centric web crawling. In *Proc. of WWW*, 2005.

[14] M. L. A. Vidal, A. S. da Silva, E. S. de Moura, and J. M. B. Cavalcanti. Structure-driven crawler generation by example. In *Proc. of SIGIR*, 2006.

[15] Y. Wang, J.-M. Yang, W. Lai, R. Cai, L. Zhang, and W.-Y. Ma. Exploring traversal strategy for Web forum crawling. In *Proc. of SIGIR*, pages 459–466, July 2008.

[16] J. Wolf, M. Squillante, P. Yu, J. Sethuraman, and L. Ozsen. Optimal crawling strategies for web search engines. In *Proc. of WWW*, 2002.

[17] Y. Zhai and B. Liu. Structured data extraction from the web based on partial tree alignment. *IEEE Trans. Knowl. Data Eng*, 18(12), 2006.

[18] S. Zheng, R. Song, J.-R. Wen, and D. Wu. Joint optimization of wrapper generation and template detection. In *Proc. of SIGKDD*, 2007.