International Journal of Engineering Research-Online
A Peer Reviewed International Journal
Articles available online http://www.ijoer.in; editorijoer@gmail.com

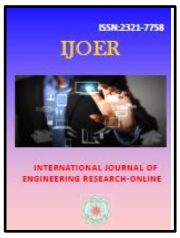
Vol.5., Issue.6, 2017
Nov-Dec

RESEARCH ARTICLE

ISSN: 2321-7758

LEARNING TO CRAWL WEB FORUMS (FOCUS)

SAGARSING G. PARDESHI[1], Prof.VIPUL D. PUNJABI[2], GANESH G. GAVIT[3]
NAYNA M. CHAUDHARI[4], DIPALI B. PAWAR[5]
[1]sagarpardeshi1694@gmail.com;[3]gavitg2@gmail.com;[4]naynachaudhari89@gmail.com;
[5]dipalipawar821@gmail.com [2]Supervisor & Assistant Professor (R.C.P.I.T) (vipulchaddha@gmail.com)



ABSTRACT

Present Forum Crawler Under Supervision (FoCUS), a supervised web-scale forum crawler. The goal of FoCUS is to crawl relevant forum content from the web with minimal overhead. Forum threads contain information content that is the target of forum crawlers. Although forums have different layouts or styles and are powered by different forum software packages, they always have similar implicit navigation paths connected by specific URL types to lead users from entry pages to thread pages. Based on this observation, we reduce the web forum crawling problem to a URL- type recognition problem. And we show how to learn accurate and effective regular expression patterns of implicit navigation paths from automatically created training sets using aggregated results from weak page type classifiers. Robust page type classifiers can be trained from as few as five annotated forums and applied to a large set of unseen forums.

1 Introduction

Internet portant services where users can request and exchange forums also called web forums are in-formation with others. For example, the Trip Advisor Travel Board is a place where people can ask and share travel tips. Due to the richness of information in forums, researchers are increasingly interested in mining knowledge from them. Zhai and Liu , Yang , and Song et al. extracted structured data from forums. Gao et al. identified question and answer pairs in forum threads. Zhang et al. proposed methods to extract and rank product features for opinion mining from forum posts. Glance et al. tried to mine business intelligence from forum data. Zhang et al. proposed algorithms to extract expertise network in forums [1].

To harvest knowledge from forums, their content must be downloaded first. However, forum crawling is not a trivial problem. Generic crawlers, which adopt a breadth first traversal strategy, are usually ineffective and inefficient for forum crawling. This is mainly due to two non-crawler friendly characteristics of forums: 1. Duplicate links and uninformative pages. 2. Page flipping links. A forum typically has many duplicate links that point to a common page but with different URLs, e.g., shortcut links pointing to the latest posts or URLs for user experience functions such as view by date or view by title. A forum also has many uninformative pages such as login control to protect user privacy or forum software specific FAQs [1][2]. Following these links, a crawler will crawl many uninformative pages. Though there are standard-based methods such as specifying the rel attribute with the no follow value (i.e., rel no follow), Robots Exclusion Standard (robots.txt), and Sitemap for forum operators to instruct web crawlers on how to crawl a site effectively, we found that over a set of nine test forums more than 47 percent of the pages crawled by a breadth first crawler following these protocols were duplicates or uninformative. This number is a little higher than the 40 percent that Cai et al. reported but both show the inefficiency of generic crawlers. More information about this testing can be found in. Besides duplicate links and uninformative pages, a long forum board or thread is usually divided into multiple pages which are linked by page flipping links. Generic crawlers process each page individually and ignore the relationships between such pages. These relationships should be pre-served while crawling to facilitate down- stream tasks such as page wrapping and content indexing. For ex-ample, multiple pages belonging to a thread should be concatenated together in order to extract all the posts in the thread as well as the reply-relationships between posts [2].

In addition to the above two challenges, there is also a problem of entry URL discovery. The entry URL of a forum points to its homepage, which is the lowest common ancestor page of all its threads. Our experiment Evaluation of Starting from Non-Entry URLs in that a crawler starting from an entry URL can achieve a much higher performance than starting from non entry URLs[2].

The major contributions of system as follows:

1. We reduce the forum crawling problem to a URL type recognition problem and implement a crawler, FoCUS, to demonstrate its applicability.
2. We show how to automatically learn regular ex-pression patterns (ITF regexes) that recognize the index URL, thread URL, and page flipping URL using the page classifiers built from as few as five annotated forums.
3. We evaluate FoCUS on a large set of 160 unseen forum packages that cover 668,683 forum sites. To the best of our knowledge, this is the largest evaluation of this type. In addition, we show that the learned patterns are effective and the resulting crawler is efficient.
4. We compare FoCUS with a baseline generic breadth first crawler, a structure- driven crawler, and a state of-the-art crawler iRobot and show that FoCUS out- performs these crawlers in terms of effectiveness and coverage[2].
5. We design an effective forum entry URL discovery method. To ensure high coverage, we show that a forum crawler should start crawling forum pages from forum entry URLs. Our evaluation shows that an entry link discovery baseline can achieve only 76 percent recall and precision; while our method can achieve over 99 percent recall and precision.
6. We show that, though the proposed approach is targeted at forum crawling, the implicit EIT like path also apply to other User Generated Content (UGC) sites, such as community QA sites and blog sites.

A recent and more comprehensive work on forum crawling is iRobot by Cai et al. iRobot aims to automatically learn a forum crawler with minimum human intervention by sampling pages, clustering them, selecting informative clusters via an in formativeness measure, and ending a traversal path by a spanning tree algorithm. However, the traversal path selection procedure requires human inspection. Follow up work by Wang et al. proposed an algorithm to address the traversal path selection problem. They introduced the concept of skeleton link and page flipping link. Skeleton links are the most important links supporting the structure of a forum site. Importance is determined by informativeness and coverage metrics. Page flipping links are deter-mined using connectivity metric. By identifying and only following skeleton links and page flipping links, they showed that iRobot can achieve effectiveness and coverage[4].

1.1 Literature Survey

Vidal et al. proposed a method for learning regular expression patterns of URLs that lead a crawler from an entry page to target pages. Target pages were found through comparing DOM trees of pages with a preselected sample target page. It is very effective but it only works for the speci c site from which the sample page is drawn. The same process has to be repeated every time for a new site. Therefore, it is not suitable for large-scale crawling. In contrast, FoCUS learns URL patterns across multiple sites and automatically ends a forums entry page given a page from the forum. Experimental results show that FoCUS is effective at large-scale forum crawling by leveraging crawling knowledge learned from a few annotated forum sites[3].

Guo et al. and Li et al. are similar to our work. However, Guo et al. did not mention how to discover and traverse URLs. However, their rules are too specific and can only be applied to specific forums powered by the particular software package in which the heuristics were conceived. Unfortunately, according to Forum Matrix, there is hundreds of different forum software packages used on the Internet. For more information about forum software packages. In addition, many forums use their own customized software [4].

2 METHODOLOGY

In this we discussed about our proposed scheme and how to implement it. We illustrate all the method in separate module with detailed description such as synopsis of anticipated scheme, ITF Regexes Learning, Online Crawling and Entry URL Discovery.

2.1 Overview Proposed Scheme

We present architectural diagram for our anticipated scheme. It consists of two major parts: the learning part and the online crawling part. The learning part first learns ITF regexes of a given forum from automatically constructed URL training examples. The online crawling part then applies learned ITF regexes to crawl all threads efficiently[6].

2.2 Page Type

We classified forum pages into page types.

Entry Page

The homepage of a forum, which contains a list of boards and is also the lowest common ancestor of all threads.

Index Page

A page of a board in a forum, which usually contains a table-like structure; each row in it contains information of a board or a thread.

Thread Page

A page of a thread in a forum that contains a list of posts with user generated content belonging to the same discussion.

Other Page

A page that is not an entry page, index page, or thread page.

2.3 URL Type

There are three types of URL.

Index URL

A URL that is on an entry page or index page and points to an index page. Its anchor text shows the title of its destination board.

Thread URL

A URL that is on an index page and points to a thread page. Its anchor text is the title of its destination thread.

Page flipping URL

A URL that leads users to another page of the same board or the same thread. Correctly dealing with page- flipping URLs enables a crawler to download all threads in a large board or all posts in a long thread.

2.4 EIT Path

An entry-index-thread path is a navigation path from an entry page through a sequence of index pages (via index URLs and index page-flipping URLs) to thread pages (via thread URLs and thread page-flipping URLs).

2.5 ITF Regex

An index-thread-page flipping regex is a regular expression that can be used to recognize index, thread, or page-flipping URLs. ITF regex is what FoCUS aims to learn and applies directly in online crawling. The learned ITF regexes are site speci c, and there are four ITF regexes in a site: one for recognizing index URLs, one for thread URLs, one for index page-flipping URLs, and one for thread page-flipping URLs[7]. Gives an example. A perfect crawler starts from a forum entry URL and only follows URLs that match ITF regexes to crawl all forum threads. The paths that it traverses are EIT paths.

Constructing URL Training Sets

The goal of URL training sets construction is to automatically construct the sets of highly precise index URL, thread URL, and page-flipping URL strings for ITF regexes learning. We use a comparable process to construct index URL and thread URL training sets since they have very comparable properties with the exception of the types of their destination pages.

Learning ITF Regexes

This sub-module, we have shown how to construct index URL, thread URL, and page-flipping URL string training set. We also elucidate how to learn ITF regexes from these training sets[7]. Vidal et al. applied URL string generalization. For example, given URLs as follows (the top four URLs are encouraging while the bottom two URLs are pessimistic).

2.6 Online Crawling

We perform online crawling using a breadth-first strategy (actually, it is easy to adopt other strategies). FoCUS first pushes the entry URL into a URL queue; next it fetches a URL from the URL queue and finally downloads its page; and then it pushes the outgoing URLs which are coordinated with any learned regex into the URL queue. FoCUS repeats this step until the URL queue is empty or other conditions are satisfied. FoCUS only needs to apply the learned ITF regexes on innovative outgoing URLs in newly downloaded pages to making the more proficient for online crawling. FoCUS does not need to group outgoing URLs, classify pages, recognize page-flipping URLs, or learn regexes again for that forum[7][8].

2.7 Online Sitemap Reconstructing and Traversal Path Selection

The goal of the online part is to mine useful knowledge based on a few sampled pages, and guide the following massive crawling. The sampling quality is the foundation of the whole mining process. To keep the sampled pages as diverse as possible, in implementation, we adopt a double-ended queue and fetch URLs randomly from the front or end. In this way, the sampling process actually adopts a combined strategy of breadth-first and depth first, and can retrieve pages at deep levels within a few steps[7]. Then, through the repetitive region-based clustering module, pages with similar layout are grouped into clusters, as illustrated with green dash ellipses, according to which kinds of repetitive patterns they hold. The by-product of this module is a list of repetitive patterns occurring in pages from the target forum. After that, according to their URL formats, pages in each layout cluster are further grouped into subsets by the URL-based sub clustering module.

Thus, each subset contains pages with both uniform page layout and URL format, marked with red ellipses, and is taken as a vertex in the sitemap graph. Arcs among various vertices are also rebuilt, where each arc is characterized by both the URL pattern and link location of the corresponding links. The third module is informativeness estimation, which is in charge of selecting valuable vertices with informative pages on the sitemap, and throwing away useless vertices with invalid or duplicate pages. The valuable vertices are labelled with shadowed red ellipses. The last module in the online part is traversal path selection, whose function is to find out an optimal traversal path to traverse the selected vertices and step aside discarded ones. The selected paths are finally shown with dark arrows, while original arcs are grey arrows [7][8].

3 SMALL CRAWLER CONFIGURATION

Small configuration with three downloaders is given, which also shows the main data flows through the system. This configuration is very similar to the one we used for our crawls, except that most of the time we used at most downloaders. A configuration as the one in would require between and workstations, and would achieve an estimated peek rate of to HTML pages per second 4.

Communication in our system is done in two ways: via sockets for small messages, and via le system (NFS) for larger data streams. The use of NFS in particular makes the design very flexible and allows us to tune system performance by redirecting and partitioning I/O between disks. While NFS has its performance limitations, we believe that the basic approach will scale well for networks of low-cost workstations and that if necessary it would be easy to switch to a more optimized le transfer mechanism. Not shown in Figure 3.3 are the interactions between crawling application and storage system, downloaders and web server, and between crawl manager and a separate web inter-face for system management[7][8]. The entire system contains about 5000 lines of C++ and Python code. Implementation was started in July 2000 as part of the first authors Senior Project. The first significant crawls were performed in January 2001, and development is still continuing. We now give some more details on each components.

3.1 Crawl Manager

The crawl manager is the central component of the system, and the first component that is started up. Afterwards, other components are started and register with the manager to offer or request services. The manager is the only component visible to the other components, with the exception of the case of a parallelized application, described further below, where the different parts of the application have to inter-act. The manager receives requests for URLs from the application, where each request has a priority level, and a pointer to a le containing several hundred or thousand URLs and located on some disk accessible via NFS. The manager will enqueue the request, and will eventually load the corresponding le in order to prepare for the download, though this is done as late as possible in order to limit the size of the internal data structures. In general, the goal of the manager is to download pages in approximately the order specified by the application, while reordering requests as needed to maintain high performance without putting too much load on any particular web server[8]. After loading the URLs of a request les, the manager queries the DNS resolvers for the IP addresses of the servers, unless a recent address is already cached. The manager then requests the le robots.txt in the web servers root directory, unless it already has a recent copy of the le. This part was initially implemented as a separate component that acted as an application issuing requests for robot les with high priority back to the manager. It is now incorporated into the man-ager process, and the robot les are written to a separate directory from the other data so they can be accessed and parsed by the manager later. Finally, after parsing the robots les and removing excluded URLs, the requested URLs are sent in batches to the down-loaders, making sure that a certain interval between requests to the same server is observed. The manager later notifies the application of the pages that have been downloaded and are available for processing[8].

3.2 Downloaders and DNS Resolvers

The downloader component, implemented in Python, fetches les from the web by opening up to 1000 connections to different servers, and polling these connections for arriving data. Data is then marshaled into les located in a directory determined by the application and accessible via NFS. Since a downloader often receives more than a hundred pages per second, a large number of pages have to be written out in one disk operation. We note that the way pages are as-signed to these data les is unrelated to the structure of the request les sent by the application to the man-ager. Thus, it is up to the application to keep track of which of its URL requests have been completed. The manager can adjust the speed of a downloader by changing the number of concurrent connections that are used[8][9].

The DNS resolver, implemented in C++, is also fairly simple. It uses the GNU and asynchronous DNS client library6 to access a DNS server usually collocated on the same machine. While DNS resolution used to be a significant bottleneck in crawler design due to the synchronous nature of many DNS interfaces, we did not observe any significant performance impacts on our system while using the above library. However, DNS lookups generate a significant number of additional frames of network traffic, which may restrict crawling speeds due to limited router capacity[9].

3.3 Crawling Application

As mentioned, the crawling application we consider in this paper is a breadth-first crawl starting out at a set of seed URLs, in our case the URLs of the main pages of several hundred US Universities, which are initially sent to the crawl manager. The application then parses each downloaded page for hyper-links, checks whether these URLs have already been encountered before, and if not, sends them to the manager in batches of a few hundred or thousand. The downloaded les are then forwarded to a storage manager for compression and storage in a repository.

The crawling application is implemented in C++ using STL and the Red-Black tree implementation in the kazlib library. (The application consists of two threads each using a Red-Black tree data structure; this required use of two different implementations since the current implementation in STL is not thread-safe).

The data structure and performance aspects of the application will be discussed. We note however the following important two points: First, since each page contains on average about hyperlinks, the set of en-countered (but not necessarily downloaded) URLs will grow very quickly beyond the size of main memory, even after eliminating duplicates. Thus, after down-loading million pages, the size of the set of encountered URLs will be well above 100 million. Second, at this point, any hyperlink parsed from a newly downloaded page and sent to the manager will only be downloaded several days or weeks later. Thus, there is no reason for the manager to immediately insert new requests into its dynamic data structures[10].

3.4 Scaling the System

One of our main objectives was to design a system whose performance can be scaled up by adding additional low-cost workstations and using them to run additional components. Starting from the configuration, we could simply add additional downloaders and resolvers to improve performance. We estimate that a single manager would be fast enough for about down-loaders, which in turn would require maybe 2 or 3 DNS resolvers. Beyond this point, we would have to create a second crawl manager (and thus essentially a second crawling system), and the application would have to split its requests among the two managers. However, the first bottleneck in the system would arise before that, in our crawl application, which is currently able to parse and process up to 400 pages per second on a typical workstation. While this number could be improved somewhat by more aggressive optimizations, eventually it becomes necessary to partition the application onto several machines[10][11].

A possible scaled up version of our system that uses two crawl managers, with 8 downloaders and 3 DNS resolvers each, and with four application components. We point out that we have never been able to test the performance of such a configuration, which would involve about 23 machines and result in download rates of probably about 1500 pages per second, way be-yond the capacity of our T3, or even a dedicated OC3 link[10][11].

4 FOCUS A SUPERVISED FORUM CRAWLER

4.1 Navigation path

Despite differences in layout and style, forums always have implicit navigation paths leading users from their entry pages to thread pages. In general crawling, Vidal et al. learned navigation patterns leading to target pages (thread pages in our case). iRobot also adopted a similar idea but applied page sampling and clustering techniques to find target pages (Cai et al.). It used informativeness and coverage metrics to find traversal paths (Wang et al.). We explicitly defined the EIT path that specifies what types of links and pages that a crawler should follow to reach thread pages[11].

5 Conclusions

Implemented FoCUS, a supervised forum crawler. Reduced the forum crawling problem to a URL type recognition problem and showed how to leverage implicit navigation paths of forums, EIT path, and de-signed methods to learn ITF regexes explicitly. Experimental results on 160 forum sites each powered by a different forum software package confirm that FoCUS can effectively learn knowledge of EIT path from as few as five annotated forums. Also showed that FoCUS can effectively apply learned forum crawling knowledge on 160 unseen forums to automatically collect index URL, thread URL, and page-flipping URL training sets and learn ITF regexes from the training sets. These learned regexes can be applied directly in online crawling. Training and testing on the basis of the forum package makes our experiments manage-able and our results applicable to many forum sites. Moreover, FoCUS can start from any page of a forum, while all previous works expected an entry URL. Our test results on nine unseen forums show that FoCUS is indeed very effective and efficient and outperforms the state of the art forum crawler, Robot.

Acknowledgements

We thank Assistant Professor of R.C.P. Institute of Technology Prof.Mr.V.D.Punjabi Sir for his continual expert guidance and encouragement throughout the project.

References

[1] R. Cai, J.-M. Yang, W. Lai, Y. Wang, and L. Zhang, iRobot: An Intelligent Crawler for Web Forums, Proc. 17th Intl Conf. World Wide Web, pp. 447-456, 2008.

[2] A. Dasgupta, R. Kumar, and A. Sasturkar, De-Duping URLs via Rewrite Rules, Proc. 14th ACM SIGKDD Intl Conf. Knowledge Discovery and Data Mining, pp. 186-194, 2008.

[3] C. Gao, L. Wang, C.-Y. Lin, and Y.-I. Song, Finding Question- Answer Pairs from Online Forums, Proc. 31st Ann. Intl ACM SIGIR

Conf. Research and Development in Information Retrieval, pp. 467-474, 2008.

[4] N. Glance, M. Hurst, K. Nigam, M. Siegler, R. Stockton, and T. Tomokiyo, Deriving Marketing Intelligence from Online Discussion, Proc. 11th ACM SIGKDD Intl Conf. Knowledge Discovery and Data Mining, pp. 419-428, 2005.

[5] Y. Guo, K. Li, K. Zhang, and G. Zhang, Board Forum Crawling: A Web Crawling Method for Web Forum, Proc. IEEE/WIC/ACM Intl Conf. Web Intelligence, pp. 475-478, 2006.

[6] M. Henzinger, Finding Near-Duplicate Web Pages: A Large- Scale Evaluation of Algorithms, Proc. 29th Ann. Intl ACM SIGIR Conf. Research and Development in Information Retrieval, pp. 284-291, 2006.

[7] H.S. Koppula, K.P. Leela, A. Agarwal, K.P. Chitrapura, S. Garg, and A. Sasturkar, Learning URL Patterns for Webpage De-Duplication, Proc. Third ACM Conf. Web Search and Data Mining, pp. 381-390, 2010.

[8] K. Li, X.Q. Cheng, Y. Guo, and K. Zhang, Crawling Dynamic Web Pages in WWW Forums, Computer Eng., vol. 33, no. 6, pp. 80-82, 2007.

[9] G.S. Manku, A. Jain, and A.D. Sarma, Detecting Near-Duplicates for Web Crawling, Proc. 16th Intl Conf. World Wide Web, pp. 141- 150, 2007.

[10] U. Schonfeld and N. Shivakumar, Sitemaps: Above and Beyond the Crawl of Duty, Proc. 18th Intl Conf. World Wide Web, pp. 991-1000, 2009.

[11] X.Y. Song, J. Liu, Y.B. Cao, and C.-Y. Lin, Automatic Extraction of Web Data Records Containing User-Generated Content, Proc. 19th Intl Conf. In-formation and Knowledge Management, pp. 39-48, 2010.

[12] V.N. Vapnik, The Nature of Statistical Learning Theory. Springer, 1995.