An Experimental evaluation of Adaptive Real Time Web Crawler

Swapnil M Phalak¹, Sandip M Walunj²

swapnilphalak@gmail.com¹, sandip.walunj@sitrc.org²

Department of Computer Engineering
Sandip Institute of Technology & Research Center, Nashik ¹, Sandip Institute of Technology & Research Center, Nashik ²

ABSTRACT
The internet is a vague collection of web pages containing vague amount of information arranged in multiple servers. The mere size of this collection is a daunting obstacle in getting necessary and relevant information. This is where search engines come into view which strives to retrieve relevant information and serve it to the user. A Web Crawler is one of the basic blocks of search engines. It is a program which browses the World Wide Web for the purpose of Web indexing and storing the data in a database for further analysis and arrangement of the data. This paper is being aimed to create an adaptive real time web crawler (ARTWC) which retrieves the web links from a dataset and then achieves fast in-site searching by extracting most relevant links with a flexible and dynamic link re-ranking scheme. Our system deduces that it is more effective than existing baseline crawlers along with an increased coverage.

General Terms
Data mining, web mining.

Keywords
Web crawler, real time crawling, in site search, deep web crawler.

1. INTRODUCTION
A Web crawler [1] is a program which scans the World Wide Web for the purpose of Web indexing and storing the data in a database for further examination and positioning of the data. The web crawling process involves collecting pages from the Web and arranging them to be picked up by the search engine. The main objective is to do so efficiently and quickly without much involvement of the remote server. A web crawler starts with a URL or a list of URLs. The crawler visits the first URL in the list. On the web page it looks for hyperlinks or anchor tags to other web pages and adds them to the existing list of URLs. The entire working of the crawler is based on the rules set for the crawler. In general crawlers sequentially crawl URLs in the list. In addition to collecting URLs, the crawler also collects data from the page. The data collected is stored and subjected to further analysis.

The adaptive web crawler [15] proposed here employs a reverse searching technique to get the pages directing to a given web link, example from a Google dataset. Further it generates site rankers for prioritizing highly relevant sites. The Site Ranker [18] is adaptively improved by an Adaptive Site Learner, which learns from the URL directing to relevant links. The crawling is focused on a topic using contents of root node of site, thus achieving more accurate results.

2. LITERATURE SURVEY
The task of web crawler [1] is not only to indexing of extracted information but it also to handle other tasks such as determining of web pages that need to be crawled, network usage and combining the extracted information together with previously downloaded data, duplication of data and integration of results.

2.1 Related Work
J. Cho and H. Garcia-Molina [2] discuss different standards and options for crawling using parallel procedure. A web crawler is either a distributed or centralized program. A web crawler is designed to avoid duplication of web pages which are downloaded while using the network efficiently. The authors describe one of the features of a web crawler that downloads the most important web pages before others. As all crawling process only focusses on local data to determine the relevancy of web pages, this process is valuable for parallel web crawler [8]. Authors also concluded that web crawlers operating in distributed mode are more favorable than web crawlers operating in multithreaded mode because of their effectiveness, throughput and scalability. Web crawler operating in parallel mode can provide high quality web pages only if network load is reduced and network distribution is completed.

Mercator [3] is a web crawler which is scalable and extensible and is currently advanced into the AltaVista search engine. [3] Authors talk about the implementation problems that arise.
while developing parallel web crawler which can also downgrade network. They talk about pros and cons of various arrangement methods and rating criteria. In short, authors agree that the operating cost of communication does not increase as more web crawlers are added, when more nodes are added then systems throughput increases and the quality of the system also, i.e. the capability to crawl and then fetch significant web pages first.

D. Fetterly [4] explains a test for measuring the web page changes rate over a large period of time. They downloaded about 151 million web pages once every week over a period of 11 weeks. The most relevant data about all crawled web pages was recorded. Single node was used for each modular job. To improve performance and avoid bottlenecks, they had to install more nodes for exact job. Hence, for adding additional crawling nodes the number of nodes escalated with a greater curvature. The communication involved in such process increases network load hugely, caused by increase in coordinating jobs among nodes with same type of jobs.

Authors [5] implemented a system for extracting web pages from web servers near to them with the use of web crawlers distributed globally. The system is susceptible to breakdowns but it has long flexibility point and the characteristics are adverse for information. When web crawler fails, the web sites which were been crawled are moved to other web crawlers for continuing the crawling process of the failed web site. Due to this result, huge duplication of data takes place. Another efficient protocol is required for deciding the next web crawler. Furthermore, a web crawler closer to various servers may be burdened while at the same time another web server at a little bit more distance may be sitting inactive.

J. Cho and Molina [6], explain the importance of URL rearrangement in the crawl frontier. If a web page currently present in the crawl frontier and connected with a number of web pages which are to be crawled, a small amount of web pages connected from it makes any sense to visit it before visiting the others. PageRank is used as a reliable metric for URL ordering and deduced three models to evaluate web crawlers. Authors conclude that PageRank is a high quality metric and web pages which have a high Page Rank are the ones with various backlinks and are needed first.

J. Cho and Molina [7], talked about building an incremental web crawler. Authors explain a periodic crawler as the one which visits the website until the collection has reached to considerable amount of web pages, and then stops. Then when it is required to refresh the data store, the web crawler creates a fresh collection using the same process explained above, and then swaps the old collection with the fresh collection. Also, an incremental web crawler [9] crawls and updates web pages after a considerable amount of web pages are crawled and saved in the data store in an incremental fashion. By these update, the crawler refreshes already existing web pages and then swaps irrelevant web pages with unusual and more relevant web pages. The authors explain Poisson process [9] which is used in web pages for checking the rate of change. A Poisson process is usually used to model an order of random events that occur autonomously with fixed rate of time. The authors describe two methods to maintain the data repository. In first method, during crawling a group of different copies of web pages is stored in the collection in which they were traced and in the second method the most recent copies of web pages are saved. For this purpose the system has to keep track of when and how frequently the web page changed. In this literature the authors deduce that an incremental web crawler can produce fresh copies of web pages more quickly and maintain the storage repository relatively fresher than a periodic crawler.

3. PROBLEM DEFINITION
To devise a real time focused web crawler to efficiently extract relevant data from web pages and serve those relevant links to the user.

4. OBJECTIVES
- To reduce the load on server system
- To improve the efficiency of search by providing relevant results.
- To provide low cost in site search solution to small web enterprises
- To obey the robot exclusion protocol

5. SYSTEM ARCHITECTURE

![Proposed System Architecture](image)

Fig 1: Proposed System Architecture

The proposed system architecture is based on MVC (Model View Architecture) design pattern and consists of three basic layers namely

- User Interface Layer
- Implementation Layer
- Data Service Layer

5.1 User Interface Layer
User Interface layer mainly consists of views for user interface of the system. The Views can be modified or updated through controllers.

5.2 Implementation Layer
Implementation layer consists of the models of the entire system. These models includes

- Web Analyzer
- Site Ranker
- Google Service

**Web Analyzer**
It continuously analyzes the results produced by the crawler to be further used of performance evaluation of the system and generating graphs.
5.3 Data Service Layer
It manages the database operations for the web controller. Data produced by Web Analyzer is stored in database.

5.4 Web Controller
It is the work force for the system as it manages most of operations within the system. It forwards the query to the Google Service handler and inputs the same data to Site ranker.

6. WORKFLOW OF THE SYSTEM

7. ALGORITHM STRATEGY
7.1 Reverse Searching
The resulting search engine results are parsed to extract links. These links are analyzed to decide whether links of sites are relevant or not using the subsequent investigative rules:
- If the page has connected links, it’s relevant.
- If the amount of seed sites or fetched web sites within the page is larger than a user defined threshold, the page has relevancy.

Algorithm –
Input: seed sites
Output: partially ranked sites

WHILE No. of candidate sites less than a threshold value:
INVOKE getWebSite (site Database, seed Sites) method
INVOKE makereverseSearch (site) method
INVOKE extractLinksfrom (result Page) method
REPEAT while any links exists:
INVOKE downloadPagefollowing (link) method
INVOKE classify Respective (page) method
IF relevant
INVOKE extractUnvisitedSite (page)
DISPLAY MostRelevantSites
END
END
REPEAT
END WHILE

7.2 Prioritizing Sites Incrementally
First, the previous data (information obtained from past crawling, like websites, links, etc.) is employed for starting website Ranker and Link Ranker. Then, unvisited sites are allotted to website Frontier and are prioritized by website Ranker, and visited websites are assigned to visited site list.

Algorithm –
Input: partially ranked sites
Output: external links

INVOKE SiteFrontier.CreateQueue (High Priority) method
INVOKE SiteFrontier.CreateQueue(Low Priority) method
WHILE site Frontier is not empty:
IF HPQueue is empty
INVOKE addAll(Queue) method on HQueue instance
INVOKE clear() method on LQueue instance
END IF
INVOKE classifySite (site) method
IF relevant
INVOKE performInSiteExploring (site) method
DISPLAY OutOfSiteLinks
INVOKE siteRanker.rank (OutOfSiteLinks) method
IF links are not empty
INVOKE add.(OutOfSiteLinks) method on HQueue instance
ELSE
8. EXPERIMENTAL RESULTS

We used Google dataset to provide seed sites for ARTWC. Each seed site was crawled and the results rearranged. The obtained results were satisfying in ARTWC with respect to relevancy, effectiveness, coverage and time of crawling.

8.1 Evaluation of Root URL discovery

For pages other than the first page, start at the top of the page, and continue in double-column format. The two columns on the last page should be as close to equal length as possible.

We compare our entry URL discovery method with a heuristic baseline. We developed a heuristic rule to find entry URL as a baseline. The heuristic baseline tries to find the following keywords ending with ‘/’ in a URL: forum, board, community, blogs, and discussions. If a keyword is found, the path from the URL host to this keyword is extracted as its entry URL; if not, the URL host is extracted as its entry URL. Our experiment shows that this naive baseline method can achieve about 90 percent recall and precision. To make ARTWC more practical and scalable, we design a simple yet effective entry URL discovery method. For each site in the test set, we randomly sampled a page and fed it to this module. Then, we manually checked if the output was indeed its entry page. In order to see whether ARTWC and the baseline were robust, we repeated this procedure 10 times with different sample pages. The results are shown in Table 1. The baseline had 80 percent precision and recall. On the different, ARTWC achieved 94 percent precision and 94 percent recall. The low standard deviation also indicates that it is not sensitive to sample pages. There are two main failure cases: 1) sites are no longer in operation and 2) JavaScript generated URLs which we do not handle currently.

We now define the metrics: effectiveness and coverage. Effectiveness measures the percentage of thread pages among all page crawled of a forum; coverage measures the percentage of crawled thread pages to all retrievable thread pages of the forum. They are defined as below, respectively

\[
\text{Effectiveness} = \frac{\# \text{ crawled pages}}{\# \text{ crawled pages} + \# \text{ other pages} \times 100}\%
\]

\[
\text{Coverage} = \frac{\# \text{ crawled pages}}{\# \text{ Total pages} \times 100}\%
\]

Ideally, we would like to have 100 percent effectiveness and 100 percent coverage when all internal links of a page are crawled. A crawler may have high effectiveness but low coverage and low effectiveness and high coverage. For example, a crawler can only crawl 10 percent of all retrievable pages, i.e., 10 percent coverage, with 100 percent effectiveness; or a crawler needs to crawl 10 times of retrievable pages, i.e., 10 percent effectiveness to reach 100 percent coverage. In order to make a fair comparison, we have mirrored few test sites by a generic crawler.

8.2 Evaluation of Online Crawling

To evaluate the real time online crawling performance of ARTWC we checked it against a few keywords and noted the evaluations, as shown in Table 2. We used Google dataset to provide seed sites for ARTWC. Each seed site was crawled and the results rearranged. The obtained results were satisfying in ARTWC with respect to relevancy, effectiveness, coverage and time of crawling. We now define the metrics: effectiveness and coverage. Effectiveness measures the percentage of thread pages among all page crawled of a forum; coverage measures the percentage of crawled thread pages to all retrievable thread pages of the forum. They are defined as below, respectively

\[
\text{Effectiveness} = \frac{\# \text{ crawled pages}}{\# \text{ crawled pages} + \# \text{ other pages} \times 100}\%
\]

\[
\text{Coverage} = \frac{\# \text{ crawled pages}}{\# \text{ Total pages} \times 100}\%
\]

Ideally, we would like to have 100 percent effectiveness and 100 percent coverage when all internal links of a page are crawled. A crawler may have high effectiveness but low coverage and low effectiveness and high coverage. For example, a crawler can only crawl 10 percent of all retrievable pages, i.e., 10 percent coverage, with 100 percent effectiveness; or a crawler needs to crawl 10 times of retrievable pages, i.e., 10 percent effectiveness to reach 100 percent coverage. In order to make a fair comparison, we have mirrored few test sites by a generic crawler.

8.3 Online Crawling Comparison

In this section, we report the results of the comparison between the structure-driven crawler and ARTWC. We let the structure-driven crawler and ARTWC crawl each seed site until no more pages could be retrieved. After that we counted how many links and other pages were crawled, respectively.

8.4 Evaluation of Manual redirects and forwards

To evaluate the amount of manual redirects or forwards our system takes to gather the relevant links we have check it against a few keywords and noted the evaluations as shown in Table 3.
Table 3. Results of Manual redirects and forwards

<table>
<thead>
<tr>
<th>Instances</th>
<th>Keywords</th>
<th>Manual redirects or forwards</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Existing System</td>
</tr>
<tr>
<td>1</td>
<td>Mowgli</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Mowgli</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Head First series</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Head First series</td>
<td>4</td>
</tr>
</tbody>
</table>

The evaluations noted in Table 3 reveal that in instance 1 and 3 our proposed system is more efficient as it took less redirects or forwards than existing system. Results of instances 2 and 4 deduce that our system supports machine learning and learns as per each user’s individual behavior and produces results with even less redirects which leads to reduced crawling.

9. CONCLUSION

As the data on the web increases, the focus is on retrieving the most relevant results while searching. With the use of adaptive crawler, private website with limited resources can provide efficient in-site search solution to its users thereby reducing their dependency on costly enterprise search solution. By focusing the crawling on a topic and ranking collected sites, our web crawler achieves more accurate results. Implementing this approach on private forums and blogs will improve their in search performance. Our approach eliminates the bias towards certain sections of a website for wider coverage of web directories. Experimental evaluations reveal that the proposed system’s effectiveness improves with repeated iterations thus overpassing that of baseline crawlers.

REFERENCES


