Focused Web Crawler with Revisit Policy

S Mali,
VJTI, Mumbai
swatimali@engg.somaiya.edu

B B Meshram
VJTI, Mumbai
bbmeshram@vjti.org.in

ABSTRACT

Focused crawlers aim to search only the subset of the web related to a specific topic, and offer a potential solution to the problem. The major problem is how to retrieve the maximal set of relevant and quality pages. In this paper, We propose an architecture that concentrates more over page selection policy and page revisit policy The three-step algorithm for page refreshment serves the purpose. The first layer contributes to decision of page relevance using two methods. The second layer checks for whether the structure of a web page has been changed or not, the text content has been altered or whether an image is changed. Also a minor variation to the method of prioritizing URLs on the basis of forward link count has been discussed to accommodate the purpose of frequency of update. And finally, the third layer helps to update the URL repository.

Categories and Subject Descriptors

H.2.8 Data Mining: crawler, page relevance, focused crawler, harvest ratio

General Terms

Algorithms, Performance, Design

1. INTRODUCTION

A crawler is an automated script, which independently browses the World Wide Web. It starts with a seed URL and then follows the links on each page in a Breadth First or a Depth First method [1].

A Web Crawler searches through all the Web Servers to find information about a particular topic. However, searching all the Web Servers and the pages, are not realistic, because of the growth of the Web and their refresh rates. To traverse the Web quickly and entirely is an expensive, unrealistic goal because of the required hardware and network resources [1, 2].

Focused Crawling is designed to traverse a subset of the Web to gather documents on a specific topic and addresses the above problem [3]. The major problem of the focused crawler is how to identify the promising links that lead to target documents, and avoid off-topic searches. To address this problem we not only use content of web page to improve page relevance but also uses link structure to improve the coverage of a specific topic. Also, it is no longer limited to simple HTML pages, but it supports a whole variety of pages used to display dynamic content and ever changing layouts.

The outline of the paper is as follows: Section 2 provides a more detailed overview of focused crawling. Section 3 describes the architecture and implementation of our approach. Comparisons with existing focused crawling algorithms on some test crawls are as shown in Section 4, and we conclude by discussing extensions and implications in Section 5.

2. LITERATURE REVIEW

A focused crawler is a program used for searching information related to some interested topics from the Internet. The main property of focused crawling is that the crawler does not need to collect all web pages, but selects and retrieves relevant pages only[1] [2] [3].

The design of a good crawler presents many challenges. Externally, the crawler must avoid overloading Web sites or network links as it goes about its business. Internally, the crawler must deal with huge volumes of data. Unless it has unlimited computing resources and unlimited time, it must carefully decide what URLs to scan and in what order. The crawler must also decide how frequently to revisit pages it has already seen, in order to keep its crawler informed of changes on the Web.

2.1 General Architecture

Roughly, a crawler starts with the URL for an initial page P0. It retrieves P0, extracts any URLs in it, and adds them to a queue of URLs to be scanned. Then the crawler gets URLs from the queue (in some order), and repeats the process. Every page that is scanned is given to a crawler that saves the pages, creates an index for the pages, or summarizes or analyzes the content of the pages[1] [3] [5].

Design of basic crawler is as shown in fig 1.

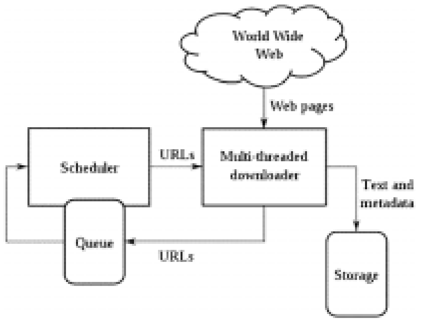


Fig 1: General architecture of web crawler [1]

2.2 Crawling Policies

The behavior of a Web crawler is the outcome of a combination of policies [1] [3]:

A selection policy : This states which pages to download As a crawler always downloads just a fraction of the Web pages, it is highly desirable that the downloaded fraction contains the most relevant pages and not just a random sample of the Web. Designing a good selection policy has an added difficulty: it must work with partial information, as the complete set of Web pages is not known during crawling.

A re-visit policy : This states when to check for changes to the pages. The Web has a very dynamic nature, and crawling a fraction of the Web can take a really long time, usually measured in weeks or months. By the time a Web crawler has finished its crawl, many events could have happened. These events can include creations, updates, and deletions. From the search engine's point of view, there is a cost associated with not detecting an event, and thus having an outdated copy of a resource.

2.2.1 A *politeness policy:* This states how to avoid overloading Web sites. Needless to say, if a single crawler were performing multiple requests per second and/or downloading large files, a server would have a hard time keeping up with requests from multiple crawlers.

2.2.2 A *parallelization policy*: This states how to coordinate distributed Web crawlers. A parallel crawler is a crawler that runs multiple processes in parallel. The goal is to maximize the download rate while minimizing the overhead from parallelization and to avoid repeated downloads of the same page. To avoid downloading the same page more than once, the crawling system requires a policy for assigning the new URLs discovered during the crawling process, as two different crawling processes can find the same URL.

2.3 Design issues of web crawler

Different types of crawlers and the different techniques used make one to consider different issues while designing and implementing them[1] [3] [5] [6].

2.3.1 Restricting followed links: A crawler may only want to seek out particular pages and avoid all other rest of the types. In order to make this happen, the designer needs to decide over restriction on the followed links. This strategy may cause numerous Web resources to be unintentionally skipped

2.3.2 Path-ascending crawling: Some crawlers intend to download as many resources as possible from a particular web site. So path-ascending crawler was introduced that would ascend to every path in each URL that it intends to crawl.

2.3.3 Focused crawling:The importance of a page for a crawler can also be expressed as a function of the similarity of a page to a given query. Web crawlers that attempt to download pages that are similar to each other are called focused crawler or topical crawlers [1] [2] [3] [7] [8].

2.3.4 Crawling the Deep Web: A vast amount of Web pages lies in the deep or invisible Web. These pages are typically only accessible by submitting queries to a database, and regular crawlers are unable to find these pages if there are no links that point to them.

2.4 Web Crawler Architecture

The basic architecture of the crawler is tailored to meet the different crawler policies. Some of them are focused crawler, intelligent crawler, deep web crawler, parallel crawler and so on.

A focused crawler is a program used for searching information related to some interested topics from the Internet[1] [2].

This identifies the most promising links that lead to target documents, and avoid off topic searches. In addition, it does not need to collect all web pages, but selects and retrieves relevant pages only. It starts with a topic vector, and for each URL , the relevance is computed for the contribution of web page in the selected domain. If it is found to be important, it gets added to the URL list else, gets discarded.

The general architecture of the focused crawler is as shown in fig 2.

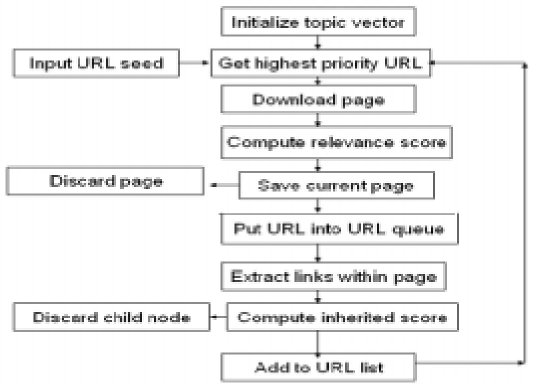


Fig 2: General Architecture of focused crawler[2]

3. PROPOSED WEB CRAWLER

The proposed architecture is divided into three layers

a. page relevance computation
b. determination of page change
c. update the URL repository

3.1 Proposed Architecture

The focused crawler that concentrates more on revisit policy starts with an URL as seed. It doesn't download the page, instead it parses to extract the URLs and words of interest into that page. Frequency of related words and number of forward and backward links to the page combinely decide the importance of the page being parsed. If the page is really important and not visited yet, it will be sent to the next layer. Otherwise, it will be checked for changes since the last visit. The changes will be considered in three categories : change in page structure, change in page contents and change in the images contained in the page. This property of the crawler helps to minimize the work of re-downloading the same pages again. In the last layer, it considers certain parameters and checks importance of the page in comparison with other pages in repository. If the level of relevance crosses predefined threshold, it will be downloaded, links will be extracted and added to the repository. The page will simply be discarded otherwise.

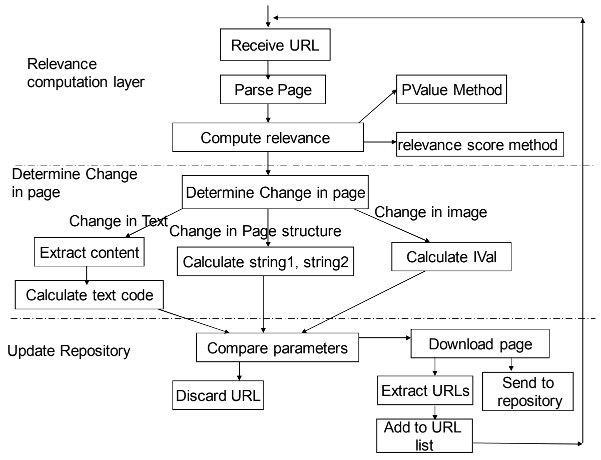


Fig 3: Proposed architecture of focused crawler with revisit policy

3.2 IMPLEMENTATION FOCUSED WEB CRAWLER WITH REVISIT POLICY

Layer 1: Page relevance computation

We propose two fold method to evaluate the importance of the relevant page with pagerank.

3.1.1 Relevance computation with topic vector

Step 1: Topic vector: T is a topic and denotes the topic vector.

T = [(k1,w1), (k2,w2), (k3,w3),…. (kl,wl),]

Where j k denotes j-th keyword or phrase of topic T. wj is the weight of the j-th keyword or phrase, denotes importance of the j-th keyword or phrase in the topic T, and $\Sigma wj = 1$, $1 \leq j \leq l$. $l = |T|$, is the amount of keyword or phrase of topic T.

Step 2: Contribution of web page as domain

The formula

$Uk = (|UKk|/|UD|)*Wk$

denotes the contribution of D for k-th keyword or phrase of topic T.

Where |UKk| is the frequency that the k-th keyword or phrase Kk of the topic T appears in the web D, |UD| is the amount of effective words in D. wk is the weight of the k-th keyword or phrase, denotes importance of the k-th keyword or phrase in the topic T, $\Sigma wj = 1$, $1\leq j \leq l$.

Step 3: Relevance-score. The relevance-score represents the relevance-score of a page. The relevance-score of the page D is defined as follows

$Sim (T,D) = \sum k=1..l\ uk$

Where $l = |T|$, it is the length of T, uk is the contribution of D for k-th keyword or phrase of topic T, and it is defined in step 2.

3.1.2 Relevance computation with link analysis Of Forward Links And Backward Links Of The Page [6]

a) The parameter used to classify URLs in the above categories is the difference of the backward link count (BkLK) and the forward link count (FwdLK) as

Pvalue = FwdLK – BkLK

b) This difference will be known as the Pvalue (priority value) of that page. A URL having the difference between FwdLK and BkLK as the highest would be given the highest Pvalue.

c) To calculate the values of FwdLK, the server would parse the page without downloading it for just to return the number of links in it.

d) To estimate the number of BkLK, the server would refer to the existing database built by crawler machines to calculate how many pages refer to this page from the point of view of current database.

The crawler will sort the list of URLs received from the repository according to descending order of Pvalue. It will then send this sorted list of URLs in order to maintain quality of crawling. This method is particularly useful as it gives weightage to current database and builds a quality database of indexed web pages even when the focus is on crawling the entire web. It works well if the database is in growing or in the maturity stage. In addition, the method would work well for broken links; as such, a URL will have a zero value for FwdLK. Even if it is referenced from pages in the database, Pvalue will always be negative resulting in low page priority [6].

Layer 2: Determine change in page (Revisit policy)[6]

The following changes are of importance when considering changes in a web page[1][2][6]

- Change in page structure.
- Change in text contents.
- Change in image (Hyperlinked or as a part of the page).

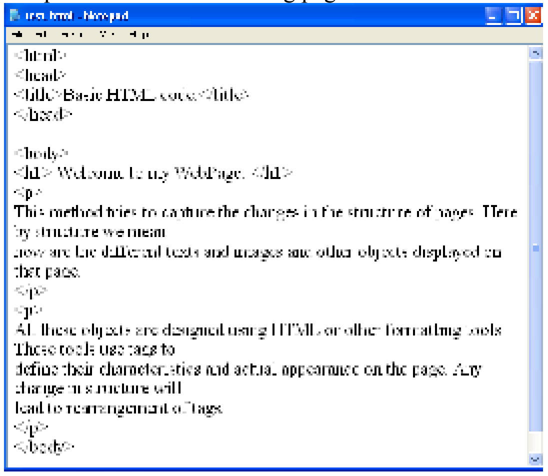
Not all the above steps are necessary to be taken care of. A parameter is compared only if the preceding parameter returns no change.

Changes in Page Structure

This method tries to capture the changes in the structure of pages. Here by structure we mean how are the different texts and images and other objects displayed on that page. All these objects are designed using HTML or other formatting tools. These tools use tags to define their characteristics and actual appearance on the page. Any change in structure will lead to rearrangement of tags. The algorithm creates two strings using the tags of that page [6].

- The first string will store the characters appearing at first position in the tag for all tags in the order they appear.
- The second string stores the character at the last position in a tag for all tags in the order they appear.
- In case the tag contains only one letter, it will be repeated as it is, in the second string.

For example consider the following page



```
<html>
<head>
<title>Basic HTML code</title>
</head>

<body>
<h1> Welcome to my WebPage. </h1>
<p>
This method tries to capture the changes in the structure of pages. Here
by structure we mean
how are the different texts and images and other objects displayed on
that page.
</p>
<p>
All these objects are designed using HTML or other formatting tools
These tools use tags to
define their characteristics and actual appearance on the page. Any
change in structure will
lead to rearrangement of tags.
</p>
</body>
```

Figure: Example Web Page

String1: [hhtbhppp]
String2: [ldey1ppp]
The proposed method offers the following advantages

- . At the time of page updating, the crawler only needs to check these two strings to determine a change.
- If the String1 itself returns a changed value, there is no need for comparison with the second string or with other change parameters for that matter.
- This method will work for different pages with different and varied formatting styles accurately capturing their structures at a given point of time.
- For most cases, a check with the first string should suffice.

Changes in Text Content

This is the next step to be carried out if the first level of determining changes does not find any changes. It may be the case that the structure of the page remains intact but there is some definite change in the actual text content of the page. This change will not be captured by the above method.

In this method, we assign a code to all text content appearing in a page. At the time of page updating, only comparison will be made to the text code of that page and if any change in that value is detected for the actual copy on the web as compared to the local copy, the page will be refreshed or re-crawled [6].

The formula for text coding = ∑ (frequency) * ASCIIsymbol / Distinct symbol count

For example, consider the following partial text content

March 26, 2007 issue - The stereotype of the "dumbjock" has never sounded right to Charles Hillman. A jock himself, he plays hockey four times a week, but when he isn't body-checking his opponents on the ice, he's giving his mind a comparable workout in his neuroscience and kinesiology lab at the University of Illinois. Nearly every semester in his classroom, he says, students on the women's cross-country team set the curve on his exams. So recently he started wondering if there was a vital and overlooked link between brawn and brains—if long hours at the gym could somehow build up not just muscles, but minds.

The ASCII sum of characters: **56652**
Total character count: **619**
Distinct character count: **43**
Therefore, the code for the text will be: **1317.488403**
Even minute changes in above text, results in significant changes in parameters as given below.

*March 26, 2007 issue - The stereotype of the "dumb jock" has never sounded right to Charles Hillman. A jock himself, **he used to play** hockey **five** times a week, but when he isn't body-checking his opponents on the ice, he's giving his mind a comparable workout in his neuroscience and kinesiology lab at the University of **Florida**. Nearly every semester in his classroom, he says, students on the women's cross-country team set the curve on his exams. So recently he started wondering if there was a vital and overlooked link between brawn and brains—if long hours at the gym could somehow build up not just muscles, but minds.*

The parameters are as follows
The ASCII sum of characters: **57113**
Total character count: **625**
Distinct character count: **43**
Therefore, the code for the text will be: **1328.209351**
The proposed method offers the following advantages,

- It gives a unique code for all text contents on a particular page.
- Even a minute change of addition or deletion of a single blank space is recorded significantly and with pin point accuracy by this code.
- The use of coding of text system eliminates the need for word-by-word parsing and comparison of each word at the time of page updating.
- Only the code of that page is compared, there is no issue of storing the whole page as an indexed structure, hence saving on large amount of storage.
- ASCII values have been used in the formula because each symbol has a distinct representation in ASCII table leading to no ambiguity.

Change in Image

While the above two methods may suffice while dealing with normal pages which uses tags for formatting their structure and text contents, they will fail for image links. In this section, we propose a method to derive a code for images to determine whether they have undergone a change or not. Ideally, a change in a link to an image hyperlink will be reflected in the label of the hyperlink for that image and the same will be depicted by the formula proposed in step 2. However, in case the text does not change but the image is replaced, it will still be left undetected[6].

Therefore, we propose the following method for image change detection

- The first step requires the image to be scaled to a standard size of n*n. Here n is of the form 2x, where value of x may vary from 4 to 6.
- Convert the image to two tone and read as an n*n array with each value being either zero or one.
- For each row of n elements, we will take 24 elements at a time and convert it to a 4 digit hexadecimal number. This will result in each row being converted to n/24

elements from n elements with each element being a 4 digit hexadecimal number.

- After doing the same operation on all the rows, we obtain an n/24* n/24 array.
- Simply, by computing the determinant of such an array, we can reduce the image to as single value (Ival).
- For each image, an Ival will be stored and at time of page updating, the crawler machines without actual download of image will calculate this Ival.

Layer 3: Repository Update Policy

If the page passes the relevancy criteria and if it has not been downloaded yet, or if the was downloaded earlier, and has also passed the change in page test, then

- Download the page
- Extract the URLs in that
- Add them to URL list

4. CONCLUSION

The architecture that has been proposed by us in this paper has the following distinct advantages:

- The centralized database of downloaded URLs reduces the dependency of the system.
- The architecture is easily scalable.
- The algorithm for checking page update parameters is designed to show even the smallest of change in a web page and leaves no ambiguity as the values stored for the parameters are distinct for small details as well. This helps the client crawler to clearly determine whether or not a page has changed and saves memory and bandwidth overheads.
- The twofold method for checking the relevancy results in download and retrieval of only most relevant pages.

Potential work related to the parallel crawler that may be added on, is the adaptation of this model for building a full fledged search engine.

5. FOCUSED CRAWLER WITH REVISIT POLICY VS OTHER ARCHITECTURES

Feature	General Focused Web Crawler	URL Based Crawler	Intelligent Crawler	focused Crawler with Revisit Policy
Relevance computation	considers only page relevance	considers URL regular expression to decide if the page is topic specific	page with least the distance of least no of clicks is given the higher priority	considers page relevance with forward and backward link analysis; can be calculated even without downloading the actual pages
Revisit policy	no specific policy is defined	no specific policy is defined	no specific policy is defined	determines the changes in image, page and text; computes the difference with original page and decides weather to visit again depending on specific parameters
Repository update	pages with better relevance factor are saved, rest are discarded	Positive pages from the given set and derivative sites are added to data store	pages with minimum distance are kept for further use, rejected otherwise	pages with highest relevance, pages with major difference in values than traversed last and the URLS extracted from those pages are added to repository
Harvest ratio	too many search insignificant results	poor harvest ratio	significant harvest ratio	significant harvest ratio
Accuracy	Poor	Moderate	accuracy improves with learning	multiple methods of calculating relevance gives more accurate results
Change detection	No	No	Slow	simple methods to detect change in page structure, and content
Broken links	doesn't recognize broken links	doesn't recognize broken links	performance improves with learning	detects broken links
Performance improvement with learning	doesn't learn	learns; but learning rate is very slow	learns and improves better	doesn't need to learn

6. REFERENCES

1. Bidoki, Yazdani et el, “FICA: A fast intelligent crawling algorithm”, Web Intelligence, IEEE/ACM/WIC International conference on Intelligent agent technology, Pages 635-641, 2007.
2. Cui Xiaoqing Yan Chun,” An evolutionary relevance calculation measure in topic crawler ” CCCM 2009, ISECS International Colloquium on Computing, Communication, Control, and Management, 267 – 270, aug 2009
3. *Junghoo Cho, Hector Garcia-Molina, Lawrence Page,* |Efficient crawling through URL ordering”, 7th International WWW Conference , April 14-18, Brisbane, 1998.
4. Mukhopadhyay et al, “A New Approach to Design Domain Specific Ontology Based Web Crawler”, ICIT 2007, 10th International Conference on Information Technology, 289 - 291, Dec. 2007
5. Peisu, Ke et el, “A Framework of deep web crawler”, 27th Chinese Proceedings of the 27th Chinese Control Conference, Pages 582-586, July 16-18, 2008.
6. www.wikipedia.org/web_crawler , accessed last May 12, 2010.
7. Yadav, Sharma et el, “Architecture for parallel crawling and algorithm for change detection in web pages”, 10th International Conference on Information Technology, Pages 258-264, ICIT 2007.
8. Yuan, Yin et el, “Improvement of pagerank for focused crawler”, 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, Pages 797-802, SNPD 2007.
9. Zheng, Chen, “HAWK: a Focused crawler with content and link analysis”, E-business engineering, 2008, ICEBE’08, IEEE international conference, pages 677-680, Oct 2008.
10. Zheng, Zhaou ET el, “URL Rule based focused crawler”, E-business engineering, ICEBE’08, IEEE international conference, Oct 2008, pages 147-154, 2008.