Intelligent Web Crawler: A Three-Stage Crawler for Effective Deep Web Mining

Jeny Thankachan¹, Mr. S. Nagaraj²
¹Department of Computer Science, Selvam College of Technology, Namakkal, Tamilnadu, India
²Assistant Professor, Department of Computer Science, Selvam College of Technology, Namakkal, Tamilnadu, India

Abstract—As deep web grows at a very fast pace, there has been increased interest in techniques that help efficiently locate deep-web interfaces. However, due to the large volume of web resources and the dynamic nature of deep web, achieving wide coverage and high efficiency is a challenging issue. We propose a three-stage framework, namely Intelligent Web Crawler, for efficient harvesting deep web interfaces. In the first stage, Intelligent Web Crawler performs site-based searching for center pages with the help of search engines, avoiding visiting a large number of pages. To achieve more accurate results for a focused crawl, Intelligent Web Crawler ranks websites to prioritize highly relevant ones for a given topic. In the second stage, Intelligent Web Crawler achieves fast in-site searching by excavating most relevant links with an adaptive link-ranking. In the third stage, the system will match the char by char user keywords with our Top-k Keywords. User will get some help to keyword typing in search panel based on Top-k Keywords. All results we compare with a threshold value (T-Value), Process those results which greater than T-value Top-k Keywords. To eliminate bias on visiting some highly relevant links in hidden web directories, we design a link tree data structure to achieve wider coverage for a website. Our experimental results on a set of representative domains show the agility and accuracy of our proposed crawler framework, which efficiently retrieves deep-web interfaces from large-scale sites and achieve higher harvest rates than other crawlers.

Keywords— web crawler, intelligent crawler, three stage crawling, site ranking, deep web

I. INTRODUCTION

This template, modified in MS Word 2007 and saved as a A web crawler is a program that goes around the internet collecting and storing data in a database for further analysis and arrangement. The process of web crawling involves gathering pages from the web and arranging them in such a way that the search engine can retrieve them efficiently. The critical objective is to do so efficiently and quickly without much interference with the functioning of the remote server. A web crawler begins with a URL or a list of URLs, called seeds. The crawler visits the URL at the top of the list. On the web page it looks for hyperlinks to other web pages, it adds them to the existing list of URLs in the list. This methodology of the crawler visiting URLs depends on the rules set for the crawler. In general crawlers incrementally crawl URLs in the list. It is challenging to locate the deep web databases because they are not registered with any search engines, are usually sparsely distributed, and keep constantly changing. To address this problem, previous work has proposed two types of crawlers, generic crawlers and focused crawlers. Generic crawlers, fetch all searchable forms and cannot focus on a specific topic. Focused crawlers such as Form-Focused Crawler and Adaptive Crawler for Hidden-web Entries can automatically search online databases on a specific topic. FFC is designed with link, page, and form classifiers for focused crawling of web forms, and is extended by ACHE with additional components for form filtering and adaptive link learner. The link classifiers in these crawlers play a pivotal role in achieving higher crawling efficiency than the best-first crawler. However, these link classifiers are used to predict the distance to the page containing searchable
forms, which is difficult to estimate, especially for the delayed benefit links. As a result, the crawler can be inefficiently led to pages without targeted forms.

Besides efficiency, quality and coverage on relevant deep web sources are also challenging. Crawler must produce a large quantity of high-quality results from the most relevant content sources. For assessing source quality, Source Rank, ranks the results from the selected sources by computing the agreement between them. When selecting a relevant subset from the available content sources, FFC and ACHE prioritize links that bring immediate return links directly point to pages containing searchable forms and delayed benefit links. But the set of retrieved forms is very heterogeneous. For example, from a set of representative domains, on average only 16% of forms retrieved by FFC are relevant. Furthermore, little work has been done on the source selection problem when more content sources are there for crawling. Thus it is crucial to develop smart crawling strategies that are able to quickly discover relevant content sources from the deep web as much as possible.

In this paper, we propose an effective deep web harvesting framework, namely Intelligent Crawler, to archive both wide coverage and high efficiency for a focused crawler. Based on the observation that deep websites usually contain a few searchable forms and most of them are within a depth of three, our crawler is divided into three stages: site locating, in-site exploring and ranking based on threshold value. The site locating stage helps achieve wide coverage of sites for a focused crawler, the in-site exploring stage can efficiently perform searches for web forms within a site and threshold value based ranking process those results which greater than T-value Top-k Keywords.

II. RELATED WORK

There are many crawlers written in every programming and scripting language to serve a variety of purposes depending on the requirement, purpose and functionality for which the crawler is built. The first ever web crawler to be built to fully function is the WebCrawler in 1994. Subsequently a lot of other better and more efficient crawlers were built over the years. The most notable of the crawlers currently in operation are as follows.

- Googlebot: The Google search uses this crawling bot. It is integrated with indexing process as parsing is done for URL extraction and also full text indexing. It has a URL server that exclusively handles URLs. It checks if the URLs have previously been crawled. It they are not crawled they are added to the queue.
- Bingbot: The Bingbot is the crawler that the Microsoft owned search engine Bing Search uses to crawl the web and collect data. It was previously known as Msnbot.
- FAST Crawl: This is the web crawler that the Norway based Fast Search and transfer uses. It focuses on data search technologies. It was first developed in 1997 and is periodically re-developed based on latest technologies.
- Web-RACE: It is a Java based crawler. It acts in part as a proxy server as it gets requests from users to download pages. When pages change they are crawled again and the subscriber is notified. The feature of this bot is it does not need a set of seeds to start crawling.
- Web Fountain: It is a distributed crawler written in C++. It has a controller and machines that repeatedly download pages. A non-linear programming method is used to solve freshness maximizing equations. We also have a lot of open source crawlers that are available online and can be used according to needs for non commercial purposes.

III. DESIGN THREE STAGE ARCHITECTURE

In the first stage, Intelligent Crawler performs site-based searching for center pages with the help of search engines, avoiding visiting a large number of pages. To achieve more accurate results for a focused crawl, Intelligent Crawler ranks websites to prioritize highly relevant ones for a given topic. In the second stage, Intelligent Crawler achieves fast in-site searching by excavating most relevant links with an adaptive link-ranking. To eliminate bias on visiting some highly relevant links in
hidden web directories, we design a link tree data structure to achieve wider coverage for a website.  
In the third stage of this paper implemented both admin and user panel. Admin will collect all 
keywords of successful search results and process the top-k results. After all results we compare with 
a threshold value (T-Value), Process those results which greater than T-value Top-k Keywords.

**Site Locating**

In order to minimize the number of visited URIs and at the same time to maximize the number of 
depth web sites we use the Reverse searching mechanism. This is being done in site collecting 
module. K mean algorithm is being used for ranking the links and the sites according to their 
relevance. After ranking Site Classifier categorizes the site as topic relevant or irrelevant for a focused crawl. If a site is classified as topic relevant, a site crawling process is launched. Otherwise, the site is ignored and a new site is picked from the frontier. A Naive Bayes classifier is used to determine whether the page is topic-relevant or not.

**In-Site Exploring**

The goals are to quickly harvest searchable forms and to cover web directories of the site as much as possible. To achieve these goals, in-site exploring adopts two crawling strategies for high efficiency and coverage. Links within a site are prioritized with Link Ranker and Form Classifier classifies searchable forms. Two crawling strategies, stop-early and balanced link prioritizing, are proposed to improve crawling efficiency and coverage. Here, in-site searching is performed in breadth-first fashion to achieve broader coverage of web directories. However, link ranking may introduce bias for highly relevant links in certain directories. Our solution is to build a link tree for a balanced link prioritizing. Link Ranker prioritizes links so that Intelligent Crawler can quickly discover searchable forms. A high relevance score is given to a link that is most similar to links that directly point to pages with searchable forms. Classifying forms aims to keep form focused crawling, which filters out non-searchable and irrelevant forms. For instance, an airfare search is often co-located with rental car and hotel reservation in travel sites.

**T-Value Ranking**

Third stage, system will take the input data from searchable forms in form database. These searchable data will match the char by char keywords with our Top-k Keywords. System will take these as keyword in search panel based on Top-k Keywords. System will evaluate the results and assign values for threshold check. All results will compare with a threshold value, Process those results which greater than T-value and will feed this set of data to adaptive link learner for ranking. System will take the input data from searchable forms in form database. These searchable data will match the char by char keywords with our Top-k Keywords. System will take these as keyword in search panel based on Top-k Keywords. System will evaluate the results and assign values for
threshold check. All results will compare with a threshold value. Process those results which greater than T-value and will feed this set of data to adaptive link learner for ranking.

**Algorithm 1: Reverse searching for more sites.**

**input:** seed sites and harvested deep websites

**output:** relevant sites

1. while # of candidate sites less than a threshold do
2. // pick a deep website
3. site = getDeepWebSite(siteDatabase, seedSites)
4. resultPage = reverseSearch(site)
5. links = extractLinks(resultPage)
6. foreach link in links do
7. page = downloadPage(link)
8. relevant = classify(page)
9. if relevant then
10. relevantSites = extractUnvisitedSite(page)
11. Output relevantSites
12. end
13. end
14. end

**Algorithm 2: Incremental Site Prioritizing.**

**input:** siteFrontier

**output:** searchable forms and out-of-site links

1. HQueue = SiteFrontier.CreateQueue(HighPriority)
2. LQueue = SiteFrontier.CreateQueue(LowPriority)
3. while siteFrontier is not empty do
4. if HQueue is empty then
5. HQueue.addAll(LQueue)
6. LQueue.clear()
7. end
8. site = HQueue.poll()
9. relevant = classifySite(site)
10. if relevant then
11. performInSiteExploring(site)
12. Output forms and OutOfSiteLinks
13. siteRanker.rank(OutOfSiteLinks)
14. if forms is not empty then
15. HQueue.add (OutOfSiteLinks)
16. end
17. else
18. LQueue.add(OutOfSiteLinks)
19. end
20. end
21. end

**Algorithm 3: T – Value Ranking**

Initialized- crawl data, k-list, t threshold, rank r.

**Input:** q query char by char,
Output: k list instant results.
1. let q != null;
2. while (q(d)) continue,
pick d1 from d,
find r from d1
if ((r(d1) ≥ t)
k=(k+d1)
continueupto k list
end if
3. end while;
4. return k;

IV. EVALUATION

Evaluation is a process of checking whether the developed system is working according to the actual requirements. Evaluation is a set of activities that can be planned in advance and conducted systematically. Evaluation is playing a vital role for the success of the system. System evaluation makes a logical assumption that if all the parts of the system are correct, the global will be successfully achieved. In adequate evaluation if not evaluation leads to errors that may not appear even many months.

Fig. 2: The numbers of relevant forms harvested by Googlebot, Smart Crawler and Intelligent 3 stage crawler

To better understand the efficiency of three crawlers, Figure 2 illustrates the number of relevant forms harvested during the crawling process. This creates two main problems. The time lag between the causes and the appearance of the problem. The effect of the system errors on the files and records within the system. A small system error can explode into a large problem. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product it is the process of exercising software with the intent of ensuring that the product is developed according to the requirement.
TABLE 1: The comparison of the number of triggered times and center pages found for reverse searching.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Times</th>
<th>Center Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Googlebot</td>
<td>Intelligent Crawler</td>
</tr>
<tr>
<td>Busfare</td>
<td>14</td>
<td>30</td>
</tr>
<tr>
<td>Automatic</td>
<td>32</td>
<td>39</td>
</tr>
<tr>
<td>Book</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Job</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Hotel</td>
<td>5</td>
<td>47</td>
</tr>
<tr>
<td>Film</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>Music</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>Rent</td>
<td>9</td>
<td>17</td>
</tr>
<tr>
<td>Product</td>
<td>35</td>
<td>22</td>
</tr>
<tr>
<td>People</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

To evaluate the performance of our crawling framework, we compare SmartCrawler to the Googlebot and smart crawler. While watching the results shown in Table 1, the efficiency of our system can be easily identified. Take reverse searching strategy in Figure 9 as a special example, Table 4 illustrates the effectiveness of reverse searching in terms of center pages collected in a crawl. Reverse searching will be triggered when the size of the Site Frontier is below the threshold, where a reverse searching thread will add sites in the center pages to the Site Frontier. Table 4 shows that reverse searching strategy is triggered many more times and gets more center pages in SmartCrawler than SCDI+reverse searching, because SmartCrawler can preferably utilize reverse searching strategy under the joint efforts of other proposed strategies.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose an effective harvesting framework for deep-web interfaces, namely Intelligent Crawler. We have shown that our approach achieves both wide coverage for deep web interfaces and maintains highly efficient crawling. Intelligent Crawler is a focused crawler consisting of three stages: efficient site locating balanced in-site exploring and T-value based ranking. Intelligent Crawler performs site-based locating by reversely searching the known deep web sites for center pages, which can effectively find many data sources for sparse domains. By ranking collected sites and by focusing the crawling on a topic, Intelligent Crawler achieves more accurate results.

REFERENCES

9. Mohammadreza Khelghati, Djoerd Hiemstra, Maurice van Keulen