ISSN:0975-9646

Smart Crawler

Mrugnayani Sharma[1], Dr. Padmapani P. Tribhuvan[2]

[1]PG Student, Department of Computer Science and Engineering
Deogiri Institute of Engineering and Management Studies,(DIEMS),Aurangabad, India
mrignayani0142@rediffmail.com

[2]Assistant Professor, Department of Computer Science and Engineering
Deogiri Institute of Engineering and Management Studies, (DIEMS),Aurangabad, India
padmapanitribhuvan@dietms.org

***Abstract*: A web crawler is a software program or a structured script that systematically, automatically browses the world-wide web. By the use of the graphical layout of the web pages, web crawlers move from web page to page. Such programs are additionally kenned as robots, spiders, and worms. In this system explained further, Data mining algorithms were used to introduce intelligence into the crawler, system architecture and performance of developed crawler is compared with HTtrack which is an opensource crawler. A statistical analysis of the performance of intelligent crawler is illustrated further. The data mining algorithm plays an important role when implementing crawler intelligence. The main goal is to create an intelligent crawler to serve the function of web indexing, which, with the aid of search engines, helps to collect relevant information from the Internet. The proposed intelligent crawler must perform crawling in minimum time with a maximum number of results.**

keywords: crawler, web indexing, statistical analysis

I. INTRODUCTION

Over the past decade, theweb has grown exponentially, resulting in the prelude of the massive amount of data in the virtual world at every instant. Consequently, the conventional crawling strategy is eventually becoming inefficient in collecting and indexing web data. Thus intelligent crawlers must be developed and used to outperform the ever increasing Internet. Of all the search engines, web crawlers are an important feature. They are the basic component of all web services, so they need high performance to be provided. The crawler is a multi-threaded bot that runs concurrently to serve the purpose of web-indexing which helps in gathering relevant information from over the Internet. This index is utilized by search engines, digital libraries, p2p communication, competitive perspicacity and many other industries. We are introducing intelligent crawler which performs crawling efficiently. Here the crawler is selective about the pages fetched and the links it will follow. This selectivity is based on the interest of the topic of the user thus at each step the crawler has to make a decision whether the next link will help to gather the content of interest. Other factors like a particular topic, the information it had already gathered also affect the efficiency and performance of the crawler [1].

While introducing perspicacity, two major approaches dominate the decisions made by the crawler. The first approach decides its crawling strategy by probing for the next best link amongst all links it can peregrinate whereas the second approach computes the benefit of peregrinating to all links and ranks them, which is utilized to decide the next link. The main objective is to develop perspicacity in crawler to accommodate the purport of web-indexing which avails in amassing pertinent information from over the Internet with the avail of search engines. The smart crawler performs crawling in minimum time with a maximum number of results. An astute web crawling strategy is to be introduced to improve the efficiency of crawling as web crawlers search the World Wide Web in a methodical, automated way. The keenly intellective crawler must perform crawling in minimum time with maximum number of URLs crawled as a result [1,2].

II. LITERATURE SURVEY

TYPES OF CRAWLERS

A. Parallel Crawlers

As the web grows in size, it becomes quite difficult or almost impossible to crawl the whole web by a single instance of a crawler. Therefore multiple processes are executed in parallel by search engines to cover the whole WWW. This type of crawler is referred to as a parallel crawler [3]. It consists of multiple crawling processes each of which performs the basic task of a single process crawler. The web pages are downloaded from the web and are stored locally. Afterwards, the URLs are extracted and their links are then followed.

B. Focused Crawlers / Topical crawlers/ Topic driven crawlers

A focused crawler [3] has three main components a classifier that takes decisions on the relevancy of a page, a distiller decides the visit priorities and a crawler which downloads WebPages and is instructed by classifier and distiller module.

C. Incremental Web Crawler

The incremental crawler [4,5] continuously crawls the web, revisiting pages periodically. During its continuous crawl, it may also purge some pages in the local collection, in order to make space for newly crawled pages. The crawler has following two goals:

- To keep the local collection fresh
- To improve quality of the local collection

D. Hidden Web Crawler

Web crawlers generally crawl the web's dense tree structure called the publicly index able Web, i.e., the set of web pages reachable purely by following hypertext links. The surface web crawlers ignore search forms and pages that require authorization or prior registration. In particular, they ignore the huge amount of high-quality content "hidden" behind the search for. The Hidden web crawler [3,4], called HiWE runs in a sequence of steps.

III **DEEP WEB CRAWLER'S FRAMEWORK** [6]

The fundamental activities of a profound web crawler are like those of other conventional crawlers [6, 7].In Figure 1 the flowchart demonstrates the normal crawler circle, comprising of URL choice, page recovery, and page preparing to extricate joins. Note that customary crawlers don't recognize pages with and without shapes. The deep web crawler's execution sequence contains additional steps for pages on which forms are detected. Specifically, deep web crawler performs the following sequence of actions for each form on a page:

Step 1 Parse and process the form to build an internal representation, based on the model outlined in Section2. (Form Analysis)

Step 2 Generate the best (untried) value assignment and submit a completed form using that assignment.(Value assignment and submission)

Step 3 Analyze the response page to check if the submission yielded valid search results or if there were no matches. This feedback could be used to tune the value assignments in step 2.(Response Analysis)

Step 4 If the response page contains hypertext links, these are followed immediately (except for links that have already been visited or added to the queue) and recursively, to some pre-specified depth. Note that we could as well have added the links in the response page to the URL queue. However, for ease of implementation, in deep web crawler, we chose tonavigate the response pages immediately and that too, only up to a depth of 1.(Response Navigation)Steps 2, 3, and 4 are executed repeatedly, using different value assignments during each iteration. The sequence of value assignments is generated using the model.

The flowchart in fig. 1 illustrates the complete architecture of the deep web crawler. It includes six basic functional modules and two internal crawler data structures. The basic crawler data structure is the URL List. It contains all the URLs that the crawler has discovered so far. When starting up the crawler, the URL List is initialized to a seed set of URLs.

IV. **FUNCTIONS OF A WEB CRAWLER**

The web searching process has two main components: offline and online [8]. The offline part is periodically performed by the search engine and it is used for building a collection of pages that will be later converted into a search index. The online part is executed each time an interrogation is performed by user. It uses the index for selecting documents that will later be sorted depending on estimation on their relevance with regard to the user's requirements. A schematic representation of this process is shown in figure 2.As web pages have different formats, the first stage for indexing web pages is represented by the extraction of a set of keywords.

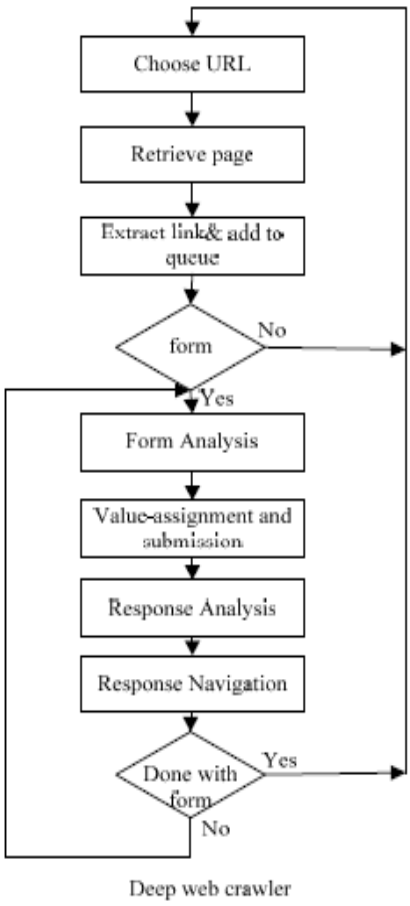

Figure 1: Deep Web Crawler Loop

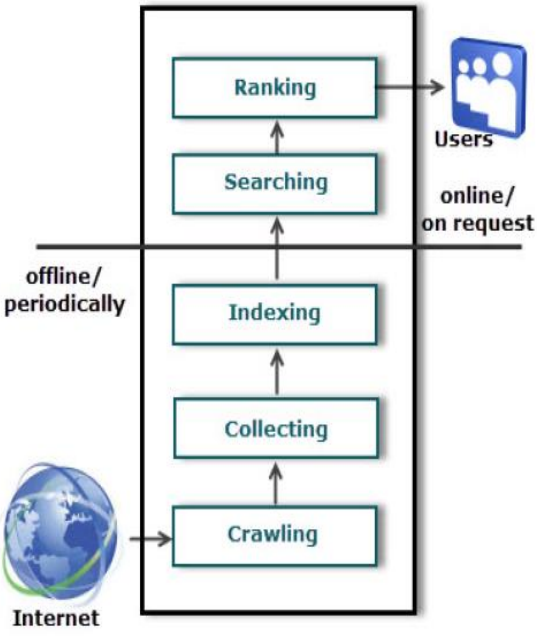

Figure 2: General structure of a web searching process

V. ARCHITECTURE OF SMART CRAWLER FOR DEEP WEB INTERFACES

Smart Crawler consists of two main stages First is Site Locating and Second is In-site exploring [10]. The figure below shows the architecture of the proposed system.

Stage 1: Site locating– In Site locating stage the smart crawler performs the operation to find out the relevant sites related to the fired query. It has a number of steps involved to give the final result of this stage.

1) Seed Sites: It is the initial stage of the architecture. Here, seed sites are the candidate sites which are given to start crawling. It begins with the following URL of the query and explores other pages and other domains.

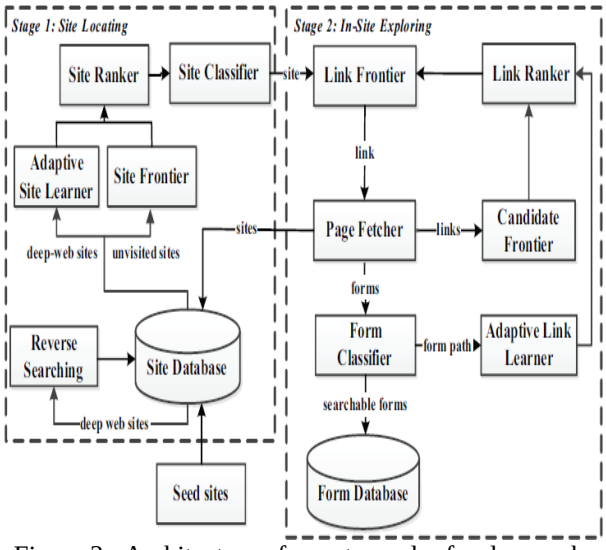

Figure 3: Architecture of smart crawler for deep web interfaces

2) Reverse searching: Pages with high rank and links to many other pages is called as a center page of the site. Some threshold is defined for seed sites, if a number of visited sited is less than the threshold then Reverse Searching is performed to know the center pages of the known deep web sites. Feed these pages back to the site database. The randomly picked site uses general search engine facility to find center pages and other relevant sites. Smart crawler first extract links on the page then download these pages and analyze these pages to decide whether the links are relevant or not. Following algorithm is used for reverse searching:

Algorithm
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 resultP age = reverseSearch(site)
5 links = extractLinks(resultP age)
6 for each 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 if
13 endfor-each
14 end while

*3) Incremental site Prioritizing:*Incremental site prioritizing is used to achieve broad coverage on websites. It records the learned pattern of deep sites and forms the path for crawling. Basic knowledge is used to initialize both rankers such as site ranked and link ranker. Unvisited sites given to site frontier later prioritize by site ranked and added to the list fetched site. Two queues are used to classify out of site links such high priority queue and low priority queue respectively. High priority queue consist of out of site links which are classifieds relevant and judge by form classifier and low priority queue consist of links that are only judged as relevant. Algorithm for Incremental site Prioritizing is given below:

Algorithm:
Input: Site Frontier.
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

3) Site Frontier: Site Frontier fetches the homepage URLsfrom the site database which is further ranked by Site Ranker to prioritize the highly relevant sites. Finding out-of-site links from visited web pages may not be enough for the Site Frontier.

4) Adaptive link learner: Site ranker and link ranker are controlled by Adaptive link learner. The feature space is decided for deep websites and links known as FSS and FSLrespectively. The Site Ranker is improved during crawling by an Adaptive Site Learner, which adaptively learns from features of deep-web sites (websites containing one or more searchable forms) found. The Link Ranker is adaptively improved by an Adaptive Link Learner, which learns from theURL path leading to relevant forms.

5) Site Ranker: Site ranker is used to rank unvisited site from the deep website. There are two parameters that are

used for ranking mechanism are Site Similarity and Site Frequency. Site Similarity depends on the topic similarity between the known deep site and new site. Site Frequency is the occurrence of the site in another website.

6) Site Classifier: The high priority queue is for out-of-site links that are classified as relevant by Site Classifier and are judged by Form Classifier to contain searchable forms. If the site is the judge as atopic relevant then site crawling process is started otherwise the new site is picked from site frontier.

B. Stage 2: In-Site Exploring –After finding most relevant sites in stage 1 stage 2 perform the in-site exploration to find searchable forms.

1) Link Frontier: Link frontier takes sites as inputs which are classified by site classifier. Link frontier mainly works for finding links withincenter pages. Criteria for stopping early are given as Crawling Strategies: Mainly two crawling strategies are present Stop early and Balance link prioritizing.

Stop Early:

SC1: when reached maximum depth.

SC2: maximum crawling pages in each depth are reached.

SC3: Predefined numbers of forms are found at each depth.

SC4: No searchable forms till threshold value.

Balance link prioritizing: Here, link tree is constructed. The rootnode is the selected site and internal leaf node is each directory present on the website.

2) Link Ranker: Link Ranker prioritizes links so that SmartCrawler 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.

3) Page Fetcher: Page Fetcher directly fetches out a center page of the website.

4) Candidate Frontier: The links in web pages are extracted into Candidate Frontier. The working of candidate frontier is similar as site frontier.

5) Form Classifier: Form classifier filters out non-searchable and irrelevant forms. The HIFI strategy is used to filter forms. HIFI consists of two classifiers, Searchable form classifier (SFC)and domain-specific form classifier(DSFC). SFC is domain independent and it filters out the non-searchable forms. It uses C4.5 algorithm for classification. DSFC is domain dependent and finds out the domain dependent form. Discuses Support vector machine.

6) Adaptive Link Learner: The Link Ranker is adaptively enhanced by an Adaptive Link Learner, which gains from the URL way prompting applicable structures.

7) Form Database: Form database contains a collection of sites; it collects all data which got input from Form Classifier.

At long last the outcome got is the most significant structures are acquired in profound web interfaces which are the coveted aftereffect of the proposed framework.

VI. SYSTEM DEVELOPMENT

By comparing DOM trees of pages with a pre-selected sample destination page, the earlier crawling system learns regular expression patterns of URLs that lead a crawler from an entry page to target pages. It is very efficient and only works for the particular site where the sample page is taken from. Every time for a new site the same process has to be repeated therefore, it is not suitable to large- scale crawling. The generic crawler started from the entry URL and a randomly selected non entry URL, respectively. It ended when no more pages could be retrieved. Nearly 100 percent effectiveness can be accomplished by a crawler pursuing only index URLs, thread URLs, and page-flipping URLs. All coverage is equal to about 100 percent when starting from the entry URL. Coverages declined dramatically when starting from a no-entry URL. For forum crawling, an entry URL is important. We suggested IWC as a smart web crawler that learns URL patterns across multiple sites and finds the entry page of a forum automatically given a page from the forum as well as URL patterns to discover new URLs instead of URL locations and there is no need to identify new crawling pages and will not be influenced by a shift in page structures. The respective results from Google, Bing and IWC showed that in Google and other web crawlers, the EIT paths and URL patterns are more stable and promising than the traversal path and URL location feature. IWC avoids duplicates URL without duplicate detection by learning patterns of index URLs, thread URLs, and page-flipping URLs and adopting a simple URL string de-duplication technique for example a string hash set.

A. Architecture of our system

Web Crawler is the essential data retrieval source that crosses the Web and downloads web documents that meet the needs of the user. The search engine and other users use the Web crawler to periodically guarantee that their database is up-to-date. The "focused crawling" technology is used when only information about a predefined subject set is needed. The Oriented Crawling technology is designed for advanced web users to concentrate on unique subjects compared to other crawling technologies and does not waste resources on irrelevant materials. An incremental crawler, on the other hand, incrementally refreshes the current set of pages by constantly visiting them; based on the estimation of how much pages shift. It also uses fresh and more significant pages to swap less important pages. This addresses the issue of the freshness of the pages. So by considering the benefits of both incremental and focused crawler we are going to introduce an intelligent crawler which can be a combination of both incremental and focused crawler.

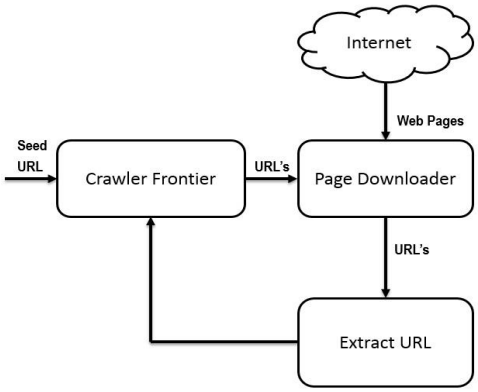

Figure 4: Architecture of our crawler.

Figure 4 shows the architecture of our smart crawler. It has two main components: a crawler frontier which stores the list of URL's to visit, Page Downloader which download pages from WWW. Here the basic processes are briefly outline.

Crawler frontier: - It contains the list of unvisited URLs. The list is set with seed URLs that a user or another program will deliver. It's simply a series of URLs. The crawler's work begins with the URL of the seed. The crawler retrieves the URL containing the list of unvisited URLs from the boundary. The page corresponding to the URL is retrieved from the Web, and the page's unvisited URLs are attached to the border. Unless the border is empty or some other condition causes it to stop, the fetching and extracting URL cycle continues. Extracting URLs based on the prioritization scheme from the boundary.

Page downloader: - The key task of the page downloader is to download from the internet the page corresponding to the URLs obtained from the border of the crawler. To do this, an HTTP client is required by the page downloader to submit the HTTP request and to read the response. To ensure that it does not take extra time to read large files or wait for a response from a slow server, the timeout duration must be set by the client. When the HTTP client is actually introduced, it is restricted to downloading only the first 10KB of a page.

The working of our web crawler is in the sequence as follows:

- Initializing the seed URL or URLs
- Adding it to the frontier
- Selecting the URL from the frontier
- Fetching the web-page corresponding to that URLs
- Parsing the retrieved page to extract the URLs
- Adding all the unvisited links to the list of URL i.e. into the frontier
- Again start with step 2 and repeat till the frontier is empty.

B. Flow Diagram of our system

The working of web crawler shows that it is recursively keep on adding newer URLs to the database repository of a search engine. This shows that the major function of a web crawler is to add new links into the queue and to choose a recent URL from it for further processing after every recursive step. It starts from seed URL i.e starting URL putting in a queue then gets first URL from the queue, visits that URL and then saves the documents from that URL after that it extracts links that are present in the URL and again adds linked URLs to the the queue. These steps are repeated till enough documents are gathered. Once all the documents are gathered it stops crawling.

Figure 5 shows the flow of stepwise procedure that carried out during crawling URL.

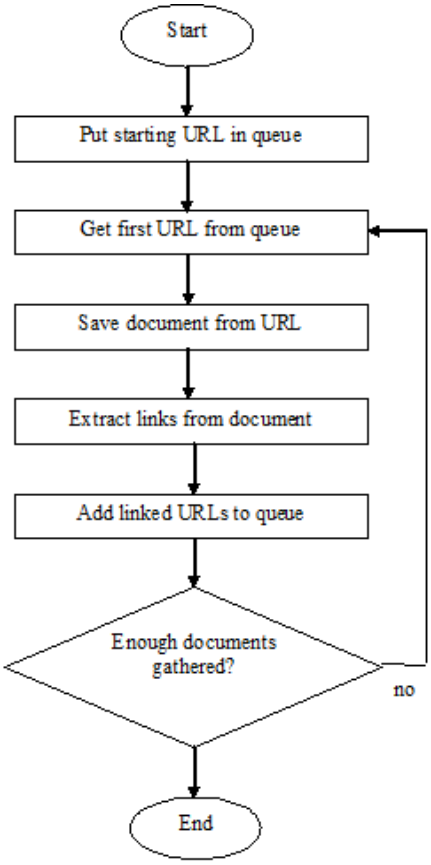

Figure 5: Functional flow diagram of proposed system.

*C. Algorithm: Adaptive A**

The main objective of the dissertation is to develop an intelligent crawler to serve the purpose of web-crawling helps in gathering relevant information from over the Internet. The intelligent crawler must perform crawling in minimum time with maximum number of results. In BFS All of the connected vertices must be stored in memory. So consumes more memory and DFS may not find optimal solution to the problem and may get trapped in searching useless path. Also, between searches by arbitrary numbers, the action costs of an arbitrary number of actions will increase. In order to concentrate its searches, adaptive A* uses informed h-values. The consumer provides the initial h-values which must be compatible with the initial costs of action. After each search, Adaptive A* updates its h-values to make them more aware and concentrate even more on their searches. So, in real-life cases, to estimate the shortest path, like-in maps, games where there can be several obstacles.

One of the best techniques used in path-finding and graph crossings is the Adaptive A* search algorithm. Adaptive A* uses A* Quest to repeatedly locate the shortest paths. To speed up the current A* search and to run faster than Repeated Forward A*, it utilizes its experience with earlier searches in the series. The role of Adaptive A* is to repeatedly find cost-minimal paths to a given set of target states in a given state space with positive action costs. The searches can vary in their starting states. Adaptive A*

Search works to concentrate its searches on educated heuristics. It changes the related value of the page with each iteration and uses it for the next traversal. The pages are modified for log(Graph Size) times, the overhead of updating is much more than the change that can be accomplished in getting more important pages after log(Graph Size) times, and then standard A* traversal is completed.

Adaptive A* Approach is as follows:

1. /*Start with given initial Seed URL as input*/
2. Adaptive_A_Star_Algo(Initial seed, Graph Size)
3. /*No. of times relevancy has to be updated to get better results*/
4. b: = log(Graph Size);
5. Repeat For (b times)
6. /*Insert Seed URLs into the Frontier*/
7. Insert_Frontier (Initial seed);
8. /*Crawling Loop*/
9. While (Frontier! = Empty)
10. /*Pick new link from the Frontier*/
11. Link: =Remove_Frontier (URL);
12. Webpage: = Fetch (Link);
13. Repeat For (each child_node of Webpage)
14. /*Calculate Relevancy Valuetill that Page*/
15. Rel_val_gn (child_node):= Rel_val(topic, node webpage);
16. /*CalculateRelevancyValuefrom that node till the Goal Page*/
17. Rel_val_hn(child_node):=Rel_val(topic,goalwebpage)-Rel_val(topic, node webpage);
18. /*Calculate Total Relevancy Value of the Path to the Goal Page*/
19. Rel_val_fn:= Rel_val_gn+Rel_val_hn;
20. /*Add new link with Maximum Relevancy Value into Frontier*/
21. Insert_Frontier (child_node_max, Rel_val_max);
22. End While Loop
23. /*After b times, A* Search more efficient on updated graphs*/
24. A_Star_Algo(seed URL, Graph (G));

At each iteration of its main loop, A* needs to determine which of its partial paths to expand into one or more longer paths. Specifically, A* selects the path that minimizes where n = last node on the path g(n) = the cost of the path from the start node to n h(n) = a heuristic that estimates the cost of the cheapest path from n to the goal.

VII. Performance Analysis

The main objective of the dissertation is to develop an intelligent crawler to serve the purpose of web-indexing which helps in gathering relevant information from over the Internet with the help of search engines. The proposed intelligent crawler must perform crawling in minimum time with maximum number of results.

A. Performance Measures

Following are some performance measures to evaluate performance of a crawler:

- Time efficiency: Crawler should be time efficient i.e. crawler should crawl the maximum URLs in minimum time span. Time efficiency can be measured as Running Time for our crawler.
- Number of URLs crawled: Number of URLs crawled should maximum in less time.
- Harvest Rate/Links tested: This rate estimates the rate of crawled pages that form relevance linking to the topic amongst all the pages that have been crawled.
- HTTrack offers functions well suited for uploading an entire website to your PC as a website crawler freeware. It has versions available for Windows, Linux, and other Unix systems, which covers most users. You can get the photos, files, and HTML code from its mirrored website and resume interrupted downloads. HTTrack works as a command-line program, or through a shell for both private (capture) and professional (on-line web mirror) use. With that said, HTTrack should be preferred and used more by people with advanced programming skills.
- So, we are going to compare our crawler's performance with the HTtrack on the basis of Running Time as a performance parameter. As shown in table we feed the same seed URLs to HTtrack and our crawler and the results are different and as we can see our crawler gives somewhat better results than HTtrack.

TABLE I
RUNNING TIME OF SMART CRAWLER AND HTTRACK

URL's	HTtrack	Our Crawler
https://docs.microsoft.com	6min:44sec	3min:08sec
https://www.britannica.com/	13min:40sec	12min:05sec
http://fabvisitingcard.in/panel/login	0min:19sec	0min:10sec
https://www.stackmint.com/index.php	0min:56sec	0min:22sec
http://www.intellspot.com/open-source-web-crawlers/	2min:16sec	1min:04sec

Figures below shows the graphs of HTtrack and our crawler comparison on the basis of Running Time for 5 different URLs. In figure 6 url: https://docs.microsoft.com is crawled and HTtrack takes 6mins:44secs for crawling the URL whereas our crawler takes 3mins:08secs. In figure 7 url:https://www.britannica.com/ is crawled and HTtrack takes 6mins:44secs for crawling the URL whereas our crawler takes 3mins:08secs.

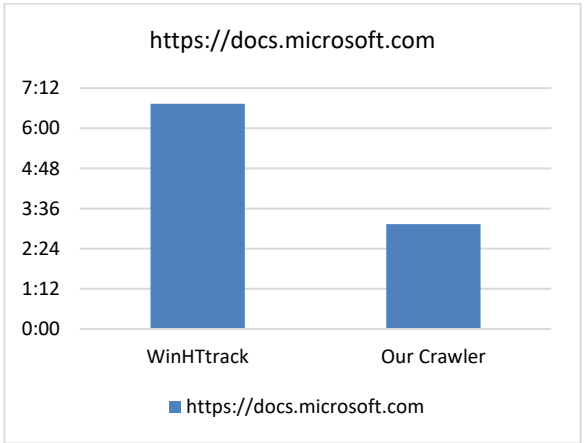

Figure 6: Running Time of HTtrack And Smart Crawler For https://docs.microsoft.com

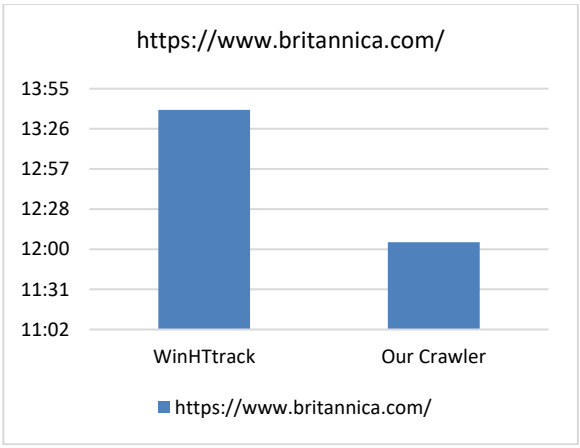

Figure 7: Running Time of HTtrack And Smart Crawler For https://www.britannica.com/

In figure 8 url: http://fabvisitingcard.in/panel/login is crawled and HTtrack takes 0mins:19secs for crawling the URL whereas our crawler takes 0mins:10secs. In figure 9 url: For https://www.stackmint.com/index.php is crawled and HTtrack takes 0mins:56secs for crawling the URL whereas our crawler takes 0mins:22secs. In figure 10 url: http://www.intellspot.com/open-source-web-crawlers/is crawled and HTtrack takes 2mins:16secs for crawling the URL whereas our crawler takes 1mins:04secs.

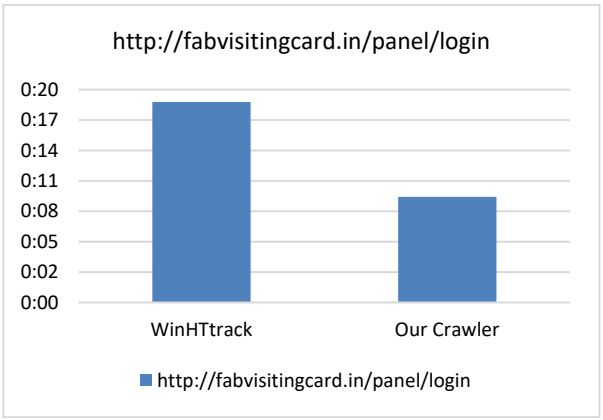

Figure 8: Running Time of HTtrack And Smart Crawler For http://fabvisitingcard.in/panel/login

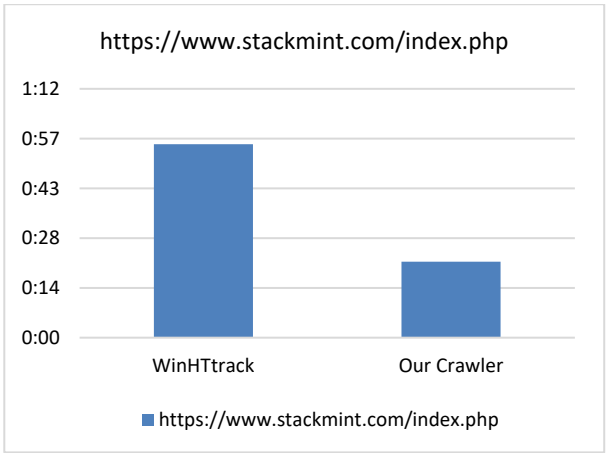

Figure 9: Running Time of HTtrack And Smart Crawler For https://www.stackmint.com/index.php

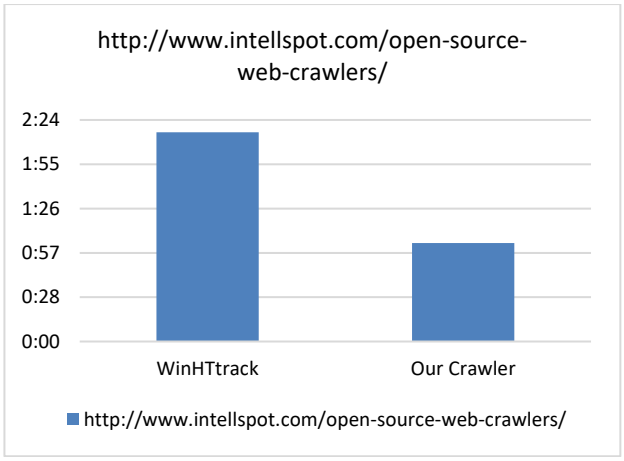

Figure 10: Running Time of HTtrack And Smart Crawler For http://www.intellspot.com/open-source-web-crawlers/

VIII. Conclusion And Future Scope

The web crawler will visit web pages on the Internet and index both new and existing web pages. The search engines use a series of crawling algorithms. The primary objective of the web crawling algorithm is to extract the URL from the crawled web pages. A successful crawling algorithm for better outcomes and high performance is implemented here. As it performs features of both, SmartCrawler is a mix of both focused crawler and incremental crawler. Our experimental findings indicate the efficacy of the smart crawler, which is higher than other crawlers in minimum running time.

In future work, the scalability of the device and the behaviour of its component can be worked on to enhance the speed and accuracy of web crawling work. While the initial findings are promising, there is still a lot of work to be done to increase the quality of crawling. For potential work, extension testing with large volumes of web pages is a big open problem.

Code optimization and URL queue optimization are also included in future work, as crawler performance is not only based on achieving the maximum number of web pages.

Acknowledgement

I offer my thanks towards my guide Dr. Ms. Padmapani P. Tribhuvan in light of his direction I have finished my work attractively

.

References

[1] https://en.wikipedia.org/wiki/Web_crawler

[2] AbhirajDarshakar, *Crawler intelligence with Machine Learning and Data Mining integration*, Pune Institute of Computer Technology, Katraj, Pune, India (ICCCA2015) ISBN:978-1-4799-8890-7/15/$31.00 ©2015 IEEE 849

[3] Shruti Sharma and Parul Gupta, *The Anatomy of Web Crawlers* ISBN:978-1-4799-8890-7/15/$31.00 ©2015 IEEE

[4] Cho, J. and Garcia-Molina, H. 2003. Estimating frequency of change.ACM Transactions on Internet Technology 3, 3 (August).

[5] Cho J and Hector Garcia-Molina, "The evolution of the Web and implications for an incremental crawler", Prc. Of VLDB Conf., 2000.

[6] Xiang Peisu, TianKe and Huang Qinzhen, *A Framework of Deep Web Crawler*.

[7] JUNGHOO C, HECTOR GM, and LAWRENCE P. Efficient crawling through URLordering. Proceedings of the Seventh

[8] MirelaPirnau, *Considerations on the functions and importance of a web crawler*, ECAI 2015 - International Conference – 7th Edition

Electronics, Computers, and Artificial Intelligence 978-1-4673-6647-/15/$31.00©2015 IEEE

[9] Keerthi S. Shetty, SwarajBhat and Sanjay Singh, *Symbolic Verification of Web Crawler Functionality and Its Properties,* 2012 International Conference OnComputerCommunication and Informatics (*ICCCI* -2012), Jan.10–12,2012, Coimbatore, INDIA

[10] Feng Zhao, Jingyu Zhou, Chang Nie, Heqing Huang, Hai Jin, *SmartCrawler: A Two-stage Crawler for Efficiently Harvesting Deep-Web Interfaces,*DOI 10.1109/TSC.2015.2414931, IEEE Transactions on Services Computing.