Abstract— Scenario in web is changing rapidly and volume of web resources is growing. Efficiency has become a challenging issue for crawling such data. The deep web content is the data that cannot be indexed by search engines as they stay behind searchable web interfaces. The proposed system aims to develop a framework for focused crawler for efficient harvesting hidden web interfaces. Initially, Crawler performs site-based searching for center pages with the assistance of web search tools to abstain from visiting more number of pages. To get more precise results for a focused crawler, proposed crawler ranks websites by giving high priority to more relevant ones for a given search. Crawler accomplishes quick in-site searching via looking for more relevant links with an adaptive link-ranking. Here we have incorporated Breath First Search (BFS) algorithm in incremental site prioritizing for broad coverage of deep web sites.

Keywords—Focused Crawler, Deep Web, BFS, Feature Selection, Ranking.

I. INTRODUCTION

Web search tools attempt to fetch data as relevant as possible. One of the parts of web search tools is the Web Crawler. A web crawler is a program that goes across the web assemble and gather data in a database for further analysis. The procedure of web crawling includes collecting pages from the web and sorting them out in a manner that the search engine can redeem them efficiently. The intention is to do it fast and efficiently without much obstruction with the operation of the remote server. A web crawler initiates with a URL also called as seed. The crawler visits the links in the list and it also looks for hyperlinks to other web pages. It then adds them to the existing list of URLs in the list. This process of crawler visiting URLs rely upon set of rules for the crawler. Generally, crawler crawls the URLs in the list incrementally. In addition to that, crawler collect data from the WebPages.

This paper is presented into following sections. Section II describes the researches done on focused crawlers previously. Section III states the aim of proposed architecture for focused crawler. Methodology, algorithms are described in section IV. Initial result obtained is given in section V.

II. LITERATURE SURVEY

The hidden web content is the information that cannot be indexed by search engines as they stay behind searchable web interfaces [2]. A Crawler encounters a variety of web pages during a crawling process. For efficient crawling and wide coverage, ranking and prioritizing links of different sites is necessary. In previous work two types of crawlers were proposed, the crawlers are generic crawlers and focused crawlers. Generic crawlers are mainly created for representing deep web and construction of directory for deep web resources. The search is not limited to a particular topic, but strives to fetch all searchable forms [3,4]. Thus, generic crawlers collects all searchable forms and does not concentrate on a specific topic. Focused Crawler is a web crawler for fetching web pages that are relevant to a particular area of interest [5]. It gathers the documents that are related to a given topic. It is called as a Topic Crawler as the result of the way it works. The focused crawler determines the relevance of the document before crawling the page. It estimates if the given page is relevant to a particular topic and how to proceeds. The fundamental point of preference of this sort of crawler is that it requires less equipment assets.

Form-Focused Crawler (FFC) [6] and Adaptive Crawler for Hidden-web Entries (ACHE) [7] automatically search online databases for a specific topic. FFC includes link, page and form classifiers for focused crawling of web forms and ACHE is an enhancement of FFC with components for form filtering and adaptive link learner. The FFC and ACHE are focused crawlers intended for searching interesting deep web interfaces. FFC Crawler performs a broad search by focusing the search on specified topic efficiently by learning to identify promising links. The crawler uses two classifiers the page and the link classifier to conduct the search. And the third classifier, the form classifier filters out useless forms. The page classifier is trained so as to classify pages according to topics specified. When the crawler fetches a page links are extracted from it. The link classifier examines links which are extracted from topic specific pages and adds them to the crawler Frontier according to their priority. The link classifier is trained to recognize links that can lead to pages which
consist of searchable form interfaces. But this approach requires considerable manual tuning, also appropriate features needs to be selected and the link classifier has to be created.

ACHE framework was proposed where crawlers could learn patterns of relevant links automatically and accordingly adapt their focus while crawling the web. This method reduced work of manual setup and tuning which was a major drawback of FFC crawlers. Adaptive learning strategy effectively manages the exploration of acquired knowledge with the exploitation of links with unknown patterns, this makes crawling process robust and corrects biases introduced in the learning process. Since this crawler learn from scratch, it is able to get harvest rates that is equal or even more than the manually configured crawlers, hence the ACHE framework reduces the effort to configure a crawler manually. ACHE improved FFC by incorporating an adaptive link learner and automatic feature selection technique. G. Alpanidis, C. Kotropoulos and I. Pitas [8] presented a latent semantic indexing classifier that incorporates link analysis along with text content in order to redeem and index the domain explicit web documents. Gunjan H. Agre and Nikita V. Mahajan [9] introduced extraction of links on the basis of search criteria or keyword. It fetches links of web pages that comprises searched keyword in their resources. It gives priority only to those pages and does not fetch web pages irrelevant to search keyword. It enhances search effectively with more accuracy, thus providing high optimality as compared to the traditional web crawler.

The main characteristic of focused crawler is to collect topic relevant pages. The previous crawling experience can help crawler to build knowledge base and learn from it, so that it can improve its performance. Niran et al [10] presented an algorithm that built knowledge bases using seed URLs, keywords and URL prediction. These knowledge bases helps crawler to learn and produce the result in more efficient way. The knowledge bases are incrementally built from the log of previous crawling. Seed URLs allow the crawler to collect as many relevant web pages as possible. Keywords support the crawler to recognize appropriate documents. URL prediction enables the crawler to predict the relevancy of the content of unvisited URLs. Qingyang Xu et al [11] introduced a new general framework for focused crawler that is based on “relational subset discovery”. To describe the relevance inbetween the pages that are not visited in the crawl frontier predicates are used, after that first order classification rules are introduced using subgroup discovery technique. Then the learned relational rules guide crawler with sufficient support and confidence.

A new approach for predicting the links that lead to relevant pages based on a Hidden Markov Model (HMM) was presented by Hongyu Liu et al [12]. The system includes three stages: user data collection, user modeling using sequential pattern learning, and focused crawling. Initially web pages are collected while browsing through web. Then these WebPages are clustered, and then hierarchical linkage pattern within pages from different clusters is then used to learn sequence of pages that lead to target pages. The Hidden Markov Model (HMM) is used for learning process. During the crawling process the priority of links are decided on basis of a learned estimate of how likely the page will lead to a target page. The performance was compared with Context-Graph crawling during experiments it was found that this approach performed better than Context-Graph crawling.

### III. Methodology

The deep web contents are the information content that cannot be indexed by search engines as they stay behind searchable web interfaces. Current deep web directories mostly have less coverage of relevant web resources which degrade their ability. A Crawler goes across a variety of web pages during a crawling process. Hence to achieve efficient crawling and wide coverage, ranking and prioritizing links of different sites is necessary. The objective of this system is to extract deep web information with wide coverage for hidden web resource and uphold efficient crawling for focused web crawler. Breath First Search (BFS) algorithm is combined with incremental site prioritizing algorithm to precisely rank the relevant links. User is then provided with the top ranked links available in searchable form.

![Flow of Proposed System](Fig.1)

The system presented in this paper is based on the framework proposed by Feng Zhao, Jingyu Zhou, Chang Nie, Heqing Huang and Hai Jin [1]. The system consist of a framework with two stages to address the problem of searching for hidden-web resources. The two stages architecture of Crawler includes, site locating and in-site exploring as shown in Figure 2. The site locating stage look for the sites relevant to the given topic. The site frequency ascertain the number of appearances of sites into another sites. The site frequency is defined in equation 1.

\[ SF(s) = \sum I_i \]  

(1)
where $I_i = 1$ if $s$ appears in a known deep web site, else $I_i = 0$. If a site is determined as topic relevant, further operation of in-site exploring is done to discover searchable forms.

An adaptive learning algorithm does online feature selection and construct link rankers automatically using those features. Each feature context can be depicted as array of terms having specific weight. The weight of term $t$ is represented by $w$ as given in equation 2.

$$W_{t,d} = 1 + \log f_{t,d}$$

where $f_{t,d}$ represent the count of appearance of term $t$ in document $d$. Adaptive learning method updates information collected successfully during crawling.

a) Site Locating - This stage finds the relevant sites for specified topic by classifying and ranking them. Site Collecting strives to maximize the number of deep websites and minimize the number of visited URLs. To maximize the number of deep websites links in the fetched pages is not enough because websites generally include less links to other sites. As a solution to this problem reverse searching and incremental two-level site prioritizing strategy are used to get more sites. Reverse searching is performed using seed site in the crawler Frontier (Site collecting). The links are extracted after parsing of the page resulted from the search engine. These pages are then downloaded and analyzed to find if the links are relevant. Then all relevant links found are intermediate output. To achieve incremental crawling and achieve broad coverage Incremental Site Prioritizing is done using High priority Queue $H_q$ and Low priority Queue $L_q$. The knowledge that is the deep websites or links gained during crawling are recorded. Then, unvisited sites are allocated to Site Frontier and are prioritized by Site Ranker and visited sites are included into the fetched site list. Once Site frontier gets enough sites, Site Ranker assigns a score to the sites depending on how relevant they are to the topic. Site Classifier function will categorize the sites according to the search query. If site is found to be relevant to the topic, crawling process is carried on. Else, that site is abstained and a new site is drawn from the frontier instead. Topic relevance of a site is based on the contents of its homepage. The homepage content of newly fetched site is harvested and parsed by removing stop words and stemming. Then a feature vector is constructed for the site the resulting vector is given as input to the Classifier to decide if the page is relevant to the topic or not.

b) In-Site Exploring - In-Site Exploring harvest the searchable forms in the ranked web sites provided by site locator. Link Ranker provide high relevance score to the links pointing to the pages with searchable form. It gives priorities to the links so that crawler can find searchable forms quickly. Form Classifier filters out the irrelevant and non searchable forms.

c) Feature selection - Feature selection discovers new search patterns. Feature space of deep web sites (FSS) and Feature space of links (FSL) are used for constructing feature space and training data. The feature space of deep web sites (FSS) is given as,

$$FSS = (U, A, T)$$

where $U$, $A$, $T$ are vectors which maps to the feature context of URL, anchor and text around URL of the deep web sites. The feature space of links of a site (FSL) is given as,

$$FSL = (P, A, T)$$

where $P$ is anchor and text around links of deep web sites and $P$ is a vector related to the path of the URL, because all the links of a particular site have same domain.

![Fig. 2. System Architecture for Deep Web Crawling](image)

**A. Algorithms**

Following are two algorithms used for deep web crawling. Here reverse searching algorithm proposed by Feng Zhao et al [1] is used as base. This algorithm [1] has been enhanced by applying Breath First Search in Incremental Site Prioritizing.

a) Reverse searching algorithm:- Reverse searching algorithm uses Seed sites and harvested deep web sites ($D_ws$) and Webpages ($W_p$) to maximize the number of deep web sites in crawling process and provide the relevant sites.

Input : Seed sites and harvested deep websites
Output: Relevant sites
1. While of candidate $W_i <$ Threshold do
2. $W_i = get(D_ws)$
3. ResultPage = ReverseSearch($W_i$)
4. $L = $ ExtractLinks(ResultPage)
5. For $\forall L$ do
   (1) $(W_p) = DownloadPage(L)$
   Relevant = Classify($W_p$)
6. If Relevant then
   RelevantSites = ExtractUnvisitedSite(Wi)
7. Provide Relevant Wi
   End

b) Incremental Site Prioritizing with BFS algorithm:- This algorithm uses learned patterns of deep websites to achieve broad coverage on websites Ws and provides the topic relevant links on the basis of crawling status Cs and page rank PR.

1. While SiteFrontier ≠ empty do
2. if Hq ≠ Empty then
3. Hq.AddAll(Lq)
4. Lq.Clear()
   end
5. (Ws) = Hq.poll()
6. Relevant = ClassifySite(Ws)
7. If Relevant then
   PerformInSiteExploring(Ws)
   Output forms and OutOfSiteLinks
   SiteRanker.rank(OutOfSiteLinks)
8. If forms ≠ Empty
   Hq.Add (OutOfSiteLinks)
   else
9. Lq.Add(OutOfSiteLinks)
10. Check(Cs)
11. Calculate(PR)
12. Sort(PR,Cs)
15. Sort(PR,Cs)
16. BFS(Links)
17. Provide the relevant Links in descending order.

Here F₁ is input function,
F₁= {Q,N,Ws}
Where
Q=Search keyword by the user
N = Number of search results required
Ws = WebSites

F₀ is output function,
F₀= {L}
Where
L= Links Retrieved

F= {Pf,PR,Rr}
Where
Pf(Ws,L) = Fetch Webpages and new Links.
BFS(L) - ReRanging of Links using BFS
PR(Ws,L) - Ranking of webpages

PR(A)=(1−∂) + ∂ (PR(T₁)/C(T₁) + ... + ∂ (PR(Tₙ)/C(Tₙ)) (3)

where,
PR(K)= PageRank of page K.
PR(T₁)= PageRank of pages Ti which link to page A.
C(T₁)= Number of outbound links on page
∂ = damping factor which can be set between 0 and 1.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

The proposed system implementation is done on a machine having 4 GB RAM , Dual core processor and 500GB Hardisk memory for storing URL data is required. Internet connection speed required is minimum 1Mbps for fast deep web crawling. For security reason it neglcts some encrypted or secured URLs from crawling process by hard coding their URL names. Minimum threshold value for number of deep websites is specified.

Using different keywords of variable length experiment was performed to explore the efficiency of proposed system. Google API is used to get the center websites. The links around those center websites are crawled using the reverse search technique and Incremental Site Prioritizing with BFS algorithm to get more links relevant to the topic. The results obtained as per experimental method explained above are given in Table 1. Number of links crawled shows the number of deep web links harvested during the crawling process. For keyword cuda 430 links were crawled and 10135 keywords were found and out of that 8 urls were displayed. Similar results were found for keywords bio-computing, android, alloy and soccer.

<table>
<thead>
<tr>
<th>Search Keyword</th>
<th>No. of URLs</th>
<th>No. of keywords</th>
<th>No. of links crawled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cuda</td>
<td>8</td>
<td>10135</td>
<td>430</td>
</tr>
<tr>
<td>Biocomputing</td>
<td>6</td>
<td>67050</td>
<td>80</td>
</tr>
<tr>
<td>Android</td>
<td>7</td>
<td>48330</td>
<td>567</td>
</tr>
<tr>
<td>soccer</td>
<td>5</td>
<td>7918</td>
<td>554</td>
</tr>
<tr>
<td>Alloy</td>
<td>8</td>
<td>18170</td>
<td>314</td>
</tr>
</tbody>
</table>

Fig. 3. System Performance

Table 1
Experimental Results
The results achieved as per experimental method explained above are given in Table 1. Number of links crawled shows the number of deep web links harvested during the crawling process.

<table>
<thead>
<tr>
<th>Keyword Length</th>
<th>Time (seconds) Existing system ((T_E))</th>
<th>Time (seconds) Proposed system ((T_P))</th>
<th>(\Delta T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5.04</td>
<td>3.36</td>
<td>33.33%</td>
</tr>
<tr>
<td>5</td>
<td>2.80</td>
<td>1.12</td>
<td>60.00%</td>
</tr>
<tr>
<td>6</td>
<td>3.09</td>
<td>1.24</td>
<td>59.87%</td>
</tr>
<tr>
<td>7</td>
<td>3.24</td>
<td>1.28</td>
<td>60.49%</td>
</tr>
<tr>
<td>8</td>
<td>3.32</td>
<td>1.32</td>
<td>60.24%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>3.49</strong></td>
<td><strong>1.66</strong></td>
<td><strong>54.78%</strong></td>
</tr>
</tbody>
</table>

The performance is compared in Table 2. In order to test the performance of the system \(\Delta T\) percentage reduction in required time to crawl deep web is calculated using equation 4.

\[
\Delta T = \frac{(T_E - T_P)}{T_E} \times 100
\]  

Where \(T_E\) is time required for crawling in the existing system \([1]\)and \(T_P\) represents time required for proposed system. The plotted bargraph Keyword Length Vs Time(seconds) for evaluated results is as shown in figure 3.

This paper experimented on enhanced algorithm Incremental Site Prioritizing with BFS achieves relevant results around 50% faster than existing system. We pursue relevant links from deep web with the use of BFS for Re-ranking the links.

### V. Conclusion

In this system an enhanced framework is proposed for deep web harvesting framework for deep-web interfaces. The technique proposed here is expected to provide efficient crawling and more coverage of deep web. Since it is a focused crawler, its searches are topic specific and it can rank the fetched sites. Incremental Site Prioritizing with BFS prioritize the relevant links according to the page ranks and displays links with higher page rank, hence it achieves accurate results. The proposed technique is expected to achieve more harvest rates than other crawlers.

### References


