*Computers, Materials & Continua*
DOI:10.32604/cmc.2021.019030
***Article***

Tech Science Press

# SmartCrawler: A Three-Stage Ranking Based Web Crawler for Harvesting Hidden Web Sources

**Sawroop Kaur[1], Aman Singh[1,*], G. Geetha[2], Mehedi Masud[3] and Mohammed A. Alzain[4]**

[1]Computer Science and Engineering, Lovely Professional University, 144411, Punjab, India
[2]CEO, Advanced Computing Research Society, Tamil Nadu, India
[3]Department of Computer Science, Taif University, Taif, 21944, Saudi Arabia
[4]Department of Information Technology, College of Computer and Information Technology, Taif University, Taif, 21944, Saudi Arabia
[*]Corresponding Author: Aman Singh. Email: amansingh.x@gmail.com
Received: 29 March 2021; Accepted: 30 April 2021

**Abstract:** Web crawlers have evolved from performing a meagre task of collecting statistics, security testing, web indexing and numerous other examples. The size and dynamism of the web are making crawling an interesting and challenging task. Researchers have tackled various issues and challenges related to web crawling. One such issue is efficiently discovering hidden web data. Web crawler's inability to work with form-based data, lack of benchmarks and standards for both performance measures and datasets for evaluation of the web crawlers make it still an immature research domain. The applications like vertical portals and data integration require hidden web crawling. Most of the existing methods are based on returning top k matches that makes exhaustive crawling difficult. The documents which are ranked high will be returned multiple times. The low ranked documents have slim chances of being retrieved. Discovering the hidden web sources and ranking them based on relevance is a core component of hidden web crawlers. The problem of ranking bias, heuristic approach and saturation of ranking algorithm led to low coverage. This research represents an enhanced ranking algorithm based on the triplet formula for prioritizing hidden websites to increase the coverage of the hidden web crawler.

**Keywords:** Hidden web; coverage; adaptive link ranking; query selection; depth crawling

## 1 Introduction

Web crawling is defined as the automatic exploration of the web. One of the trending research topics in this area is hidden web crawling. The information available on the hidden web is in the form of HTML forms. It can be accessed either by posing a query to general search engines or by submitting forms. To reach a certain high rate of coverage, using any of the methods is full of challenges. Locating the hidden web sources, selecting relevant sources and then extraction of underlying content are the three main steps in the hidden web. The scale of the hidden web is

large, so manual identification of hidden web sources is hard, it has to be automatic. In the case of dynamic web pages, data is required from the web server, but the problem is HTML does not deal with databases. So here an advanced application server is required. When a client asks for a web page from the server. It will check if it has a page. If it has to create a page, this is called a dynamic page. Now web server will interact with the application server, which will ask the page from the database. The database will process the data and it will send a page to the client. The client will never come to know how its page is processed. Pages of dynamic websites are not coded and saved separately; these are dynamically populated every time.

Web dynamism is the non-availability of static links and the web crawler's inability to automatically fill query interfaces are the main reason for the existence of hidden web [1]. A webform is composed of one or more fields and control elements [2]. The purpose of attributes of form is conveyed through labels. For example, if a book is to be searched, the attribute title of the form indicates the name of the book to be searched. The purpose of this query form would be to satisfy the queries like "find books from XYZ publication". An HTML form is recognized from $<$form$>$ and $<$\form$>$ tag. When the form is submitted with suitable values, the web browser sends an HTTP request to the server. This request includes input and their corresponding values. Two methods named 'GET' and 'POST' exist for this process. In the GET method, parameters that have to act are included as part of the URL in the request. So, URLs are unique. During the POST method, parameters are sent in the body of the HTTP request. These methods are used only when the submission of form change the state of form.

Dimensions of form identification are domain *vs.* function, form structure, and pre-*vs*. post-query approaches [3]. It has been inferred that most of the existing hidden web crawlers assume that all the corresponding documents are returned by queries. But in practice, queries return only the top $k$ matches. Due to which exhaustive data harvesting is difficult. In this way, the top k documents will be retrieved more than one time and low documents have slim chances of being retrieved. The information extraction techniques have matured but with the pervasiveness of programmable web application, a lot of research has been done in the first two steps in hidden web crawling. Query selection techniques play a vital role in the quality of crawled data. Our present work is in line with the query selection problem in text-based ranked databases. To remedy the shortcoming of ranking algorithms, the major contributions of this research are:

- An effective triplet function based ranking reward is proposed to eliminate the ranking bias towards popular websites. The algorithm incorporates out of site links, weighting function and similarity as collective value to rank a document. The drawback of ranking bias is solved by adding number of out of site links to reward functions. The approach considerably increases the coverage of the crawler.
- We proposed a function for retrieving the quality data and designed a stopping criterion that saves crawlers from falling into spider traps. These rules also mitigate the draining of the frontier for domains and status codes. Ranking, as well as learning algorithms, are adaptive and leverage the information of the previous runs.
- Since the data sources are form-based, this approach works well with both GET and POST methods. With a certain return limit, the algorithm is proved to be efficient in curating unique documents. Both the detection and submission of forms is automatic and with successive runs dependency on seed sources is reduced.

The next Section 2, throws light on existing research in hidden web crawling and ranking algorithms. In Section 3, we define our proposed approach. Section 4 shows a discussion of the

results of our experiments. Section 5 concludes our paper and also includes the future line of research.

## 2 Literature Review

The goal of the hidden web crawler is to uncover data records. The resultant pages can be indexed by search engines [3]. The attempt is to retrieve all the potential data by giving queries to a web database. Finding values for the forms is a challenging task. For few controls where the value is definite i.e., to choose from existing is relatively easy as compared to text-based queries to get resultant pages. Query generation methods in the hidden web are either based on prior knowledge i.e, to construct a knowledge base beforehand or without any knowledge [4]. A query selection technique, which generated the next query based on frequent keywords from the previous records were first presented in [5]. To locate the entry points of the hidden web, four classifiers are being used. If the page has information about a topic or not is decided by a term-based classifier. To find links that will lead to a page with a search form link page classifier is used. Search form classifier to discard non-searchable forms and domain-specific classifier to collect only relevant information from search form available on websites. The crawler is focused to find relevant forms (searchable) by using multi-level queues initialized with a seed set [6]. Search forms have different types, each type fulfilling a different purpose.

### *a. Importance of Coverage in the Hidden Web Crawler*

Authors in [7] suggested that along with scalability and freshness, coverage is another important measure for hidden web crawlers. As scalability and freshness cannot measure the effectiveness of form-based crawlers. Coverage is defined as the ratio of the total number of relevant web pages that the crawler has extracted and the total number of relevant web pages in hidden web databases. For this, a crawler is dependent on database content. Another metric called is submission efficiency is defined as the ratio of response webpages with search results to the total number of forms submitted by the crawler during one crawl activity. Suppose a hidden web crawler has crawled $N_c$ pages, and let $N_T$ denote the total number of domain-specific hidden web pages. $N_{sf}$ is the total number of searchable forms that are domain-specific. Then harvest ratio is defined as ratio if $N_{sf}$ and $N_c$. Coverage is defined as $N_{sf}$ and $N_T$. Harvest ratio measure the ratio of relevant forms crawled from per web pages while coverage is defined as the ability to crawl as many relevant pages with a single query. Whether the crawled content is relevant to the query or not is measured by precision [8]. Authors in [9] have introduced another measure called specificity for the hidden web. In another study coverage is defined as the number of web pages that can be downloaded by updating query keywords [10]. The above literature shows that different studies have defined coverage in different ways.

### *b. The Crawling Problem*

Every crawler has to set a limit for the documents to be returned. The crawlable relationship is can be represented by a document query matrix. But unlike traditional document term matric, it does not guarantee even if the document contains the query keyword. The document has to score a high rank from top k matches. Problem is to select a subset of $Q$ terms that can cover as many documents as it can. If $DH$ is a hidden web database or a data source that has $DI$ documents. $M$ is the ranking number.

In a data source, Suppose $Dm$ is placed at the rank higher than $Dn$. A database can be queried with a set of queries. There exist two types of techniques for it. (i) set covering approach, (ii) machine learning-based (iii) heuristic methods (iv) ranked crawling. Set covering method was

first implemented by [11]. The document frequencies are estimated from fully or partially downloaded pages. Fig. 1 shows the basic steps involved in hidden web crawling. Unlike traditional crawling, when forms are encountered these are analysed and filled with suitable values. Response generated after submitting the form-based data is used for further harvesting or indexing. In machine learning methods for ranking, each feature is assigned a weight, then the machine learning algorithm learns the relation between features and weights.

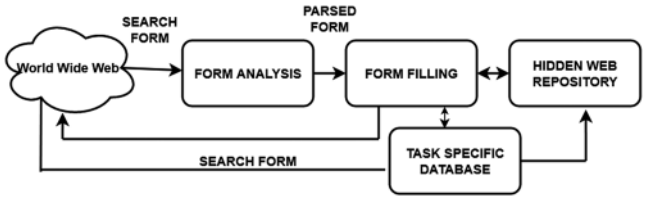


**Figure 1:** Basic steps of hidden web crawling

It has been observed that if the document has a low rank its probability of being retrieved is low. If the document has a high rank it can be retrieved multiple times. And we infer that if the document is retrieved multiple times, it is a waste of resources. So, the system needs to keep a check on the documents already retrieved. From the seven domains of consideration, the data set in total consist of 25798 documents. The document ID's were considered document rank. The query words are the most popular words with frequency ranging 1000–6000. If we set the return limit k = 100, only top 1000 documents can be reached. To address this problem, we have proposed a ranking method based on term weighting, number of out of site links and site similarity. Term weighting is estimated by the sample data. The term and documents which have frequencies less than k are selected. In performance evaluation we have tested our method over book, product, auto, flight, hotel, music and premier domains.

The critical literature review portrays the inadequacies like the form filling process execute only the current run, the outcomes of a query from the previous run are not considered. The effect of ranking the web page is not part of most of the techniques. Crawling can be made more intractable if queries are dependent on the ranking. Ordinary crawling methods face query bias problem and do not perform well on ranked data sources. Ranking bias increase with the size of the dataset. If all queries match more than N web pages, the bottom-most webpages have the least chance to be retrieved. Due to which the crawling process becomes inefficient. Therefore, in this paper, a novel technique for ranking hidden web crawling has been proposed. A hidden web database is said to be ranked if it returns k top sources. The ranking in this approach is not dependent on queries. The ranking formula is triplet factor of out of site links, term weighting and site similarity.

The following observations motivated the idea:

(1) It is desirable to have a system that can automate data retrieval from hidden web resources. It is not possible to get all the data in a single crawl, so to retrieve the data multiple runs of crawler is required. No single crawl can provide all the desired data that is resident in a database, therefore multiple crawls need to be carried out to exhaustively retrieve all the content in the web database.
(2) To address this problem of ranking, we have proposed a ranking formula that helps to retrieve the document even if it is at the bottom of the list. The key idea is to use the

term weighting, site frequency and cosine similarity for a ranking formula. Since weighting frequencies are estimated based on sample documents. unknown, we need to estimate them based on sample data.

(3) During form filling, for every domain, the crawler tries to match its attributes with the fields of the form, using cosine similarity.

(4) It is also desirable that the crawler should work on both types of submission methods. The proposed crawler has not only implemented both get and post method but also considered more types of forms of status codes.

## 3 Proposed System

Let D be the database with $d_1$, $d_2$, $d_3$ ..... $d_m$ documents, where $1 \le i \le m$. The subscript $i$ of $d_i$ is the rank of document. If ( $i < j$), this means that $d_i$ has high rank than $d_j$. Suppose for database D, queries are denoted by $q$.

$$q = \{q1,\ q2,\ q3\ \ldots\ldots\ldots\ qn\} \tag{1}$$

$(1 \le j \le n)$ is the coverage of the document in D

For the sake of simplicity, we have ignored the query terms that are not present in any of the document. If $q_j$ covers total number of documents then F($q_j$) denotes document frequency of $q_j$. Now if the return limit of document is '$L$' then in query matrix A = ($m_{ij}$) in which rows and columns represent the documents and queries respectively.

$$\text{If } m_{ij} = 1, \quad q_j \text{ covers } \mathrm{d_i}. \tag{2}$$

$$m_{ij} = \begin{cases} 1 \ \textit{if qi covers di} \\ 0 \ \textit{otherwise} \end{cases} \tag{3}$$

Goal is to select a subset of terms $q'(q' \subseteq q)$ to cover maximum '$d$' in $D$ with minimum cost. If probability of document to be matched by query is unequal, then a query is said to have query bias. Query bias and size of document are directly proportional to each other. So, query bias is larger in large documents. The ranking bias is probability that document can be returned. One of the most prominent feature of hidden web sources is that documents are ranked to return top documents for queries.

To find the weighted average the assumption are- The actual number of documents is $N$, and $q$ queries are sent to $D$. $M_d$ denotes the total number of documents that are matched. $\delta_j$ is total number of new documents retrieved by q. $U_j$ is the total number of unique documents. $D_d$ denotes the total number of duplicate documents. $L$ is the return limit. $R$ is ratio of distinct document and actual document.

$$\text{Weighted average of query} = \frac{\Sigma_{i=2}^{t} w_i M_i / d_i}{\Sigma_{i=2}^{t} w_i} \tag{4}$$

If all the document matched with query and all the documents are returned, then let $M_0$ is the model in which subscript 0 is the variation of match probability.

$$P = 1 - OR^{-2 \cdot 1} \tag{5}$$

$$M_0 = \frac{M}{P} \tag{6}$$

$$M_0 = \frac{M}{\left(1 - OR^{-2.1}\right)} \tag{7}$$

Since our method is based on machine learning the query selection function is

$$Q_s(s,q) = r(s,q) + \max_{q \epsilon Qs}[r(s_{t+1},q)] \tag{8}$$

s = state of the crawler

$Q'_s$= candidate queries

$R(s, q)$ = reward function, for simplicity lets denote $R(s, q) = €$

However only this function could not return promising queries for hidden web where the documents are ranked. And we cannot measure the quality of the data. In web crawling quality of the URL is indicated by frequency of out of site links. So, the ranking function is enhanced by performing sum of weighting term and out of site frequency. But the issue here is to find similar data. So, the third function is cosine similarity between the vectors.

$$(rj) = (1 - w).\delta j + w.ranking\ reward\ (€)/cj \tag{9}$$

$$Where\ € = wij + S + SF \tag{10}$$

Let SF be the frequency of out-links. Section 3.1 explains the details of the derived formula. If $(Sq)$ represent small queries then the $(Sq)$ is subset of $(SQ)$. Suppose $F(q)$ is the term frequency inverse document frequency. $F(q) < 'L'.Sq$ is selected randomly from $SQ$.

Fig. 2 shows the way the query could be chosen. According to our model system can choose queries with small frequencies to reach high coverage. and query words are not chosen beyond the scope of D. Trying all the frequencies is beyond the scope of this research so we have assumed the return limit of 100. Among this range only queries are selected. in case the sample size is small, then to enlarge the sample of frequencies Maximum likelihood estimator is used. For each term $f(q)$ enlargement factor $F'q$ is defined as:

$$F'q = f(q) \times \frac{|DB|}{|D|} \tag{11}$$

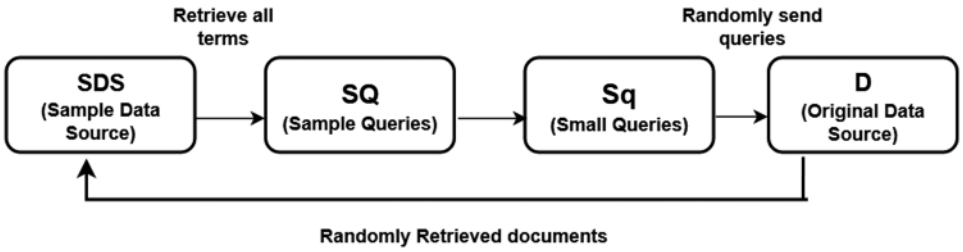


**Figure 2:** Working of query system

DB is the size of the original database, and D is the sample size. With computation of $(F'q)$ document frequencies are used to rank all the terms in D. In case of small queries or single word

query, terms are used to enhance the local terms. Function $(r_j)$ computes the complete rank, while sample size issues are handled by $(F'q)$.

In the model discussed above, we have randomly selected the documents directly from the seed set. Hence, we assumed that all the matched documents are returned. Now in case of documents with unequal probability, ranked data will retrieve only top (L/M) documents. In practice it would not be easy to select the queries with document frequencies fixed to any number. Suppose that t′ number of queries are sent to the data source, each match with mi number of documents, where $1 \leq i \leq t$. We define the overflow rate as:

$$O\mathcal{F} = \sum_{t=1}^{t} \frac{n_i}{LT} \tag{12}$$

$$N_h = \frac{OF \times N}{\left(1 - OR^{-1.1}\right)} \tag{13}$$

IN actual the data source size estimation, documents will have varying probabilities. Let's denote this case by $M_{h.}$ the relation between probability and estimation function is obtained as follows:

$$P = 1 - \alpha OR\beta \tag{14}$$

$$\ln(1 - P) = \ln\alpha + \beta \times \ln OR \tag{15}$$

Every crawler has a list of queues called frontier or crawl frontier. It has all the unvisited links. These links are fetched by requesting the HTTP server. It takes one seed URL to start the crawling. The page is processed and parsed to extract its content. All the links available are checked for the forms. Links are collected and the frontier is rearranged. This module extracts the links or the hyperlinks. The two heuristics are followed by this module. First for every extracted link rejection criterion is followed [12]. After that stopping criteria is followed as follows:

(1) The crawler stops crawling if a depth of three is reached. It is proved in [13] that most of the hidden web pages are found till depth 3. We have decided that at each depth maximum number of pages to be crawl is 100.
(2) At any depth maximum number of forms to be found is 100 or less than a hundred. If the crawler is at depth 1, it has crawled 50 pages, but no searchable form is found, it will directly move to the next depth. And the same rule is followed at depth. Suppose if at depth 2, 50 pages are crawled and no searchable form is found. The crawler will fetch a new link from the URL. This dramatically decreases the number of web pages for the crawler to crawl. But these links are relevant for a focused crawl.

The crawler according to its design keeps on crawling the webpages. But exhaustive crawling is not beneficial. So, it is required to limit the crawl. The stopping criteria discussed above serve this purpose. The most generic steps involved in designing a hidden web crawler that employs the ranking of URLs is defined in Algorithm 1 as follows:

**Algorithm 1:** Hidden web algorithm

Step 1 Crawler get a URL from the frontier and request the web server to fetch a page.
Step 2 The second step has two parts:
a. After the webpage is fetched, it is parsed and analysed, links available on it are extracted.
b. The page is checked according to rejection rules.
Step 3 Filtered links are sent to crawl frontier. Then rank of the URL is computed and the frontier list is re-ordered.
Step 4 Now the forms are analyzed for submission method, whether it post or get. With suitable values, forms are sent to the server. The server replies to the crawler about each entry to that form. Crawler sends the filled form by placing the received query words to the HTTP server.
Step 5 Fetched pages are ranked and URLs are maintained in the database.
Step 6 Links received on web pages are further sent to the crawling list.
Step 7 Step 1-6 are repeated.

Most of the times the ranking formula is based on ranking top k terms. This leads to ranking bias. To remedy this shortcoming, the traditional ranking algorithm is enhanced with triplet factor. The following section describes an efficient ranking algorithm. The proposed algorithm is within the context of text-based ranked hidden web sources to maximize the number of hidden web website but minimize the number of visits to a particular source repeatedly.

***a. Enhanced Ranking Algorithm***

The aim of ranking in hidden web crawling is to extract top $n$ documents for the queries. Ranking helps the crawler to prioritize the potential hidden websites. Three function formula is designed for ranking hidden web sites. Initially, the crawler is checked on pre-designed queries. The ranking formula is adopted from [14]. But our reward function is based on several out-links and site similarity and term weighting. The formula is explained in Eqs. (9) and (10).

$w$ is the weight of balancing $€and\, c_{j.}$ as defined in Eq. (4). $\delta_j$ is the number of new documents. The computation of $r_j$ shows the similarity of $€$ and returned documents. If the value of €is closer to 0, it means that returned value is more similar to the already seen document. $c_j$ *is a* function of network communication and bandwidth consumption. The value of the ranking reward will be closer to 0 if the new URL is similar to the already discovered URL. Similarity ($S$) is computed between the already discovered URL and the newly discovered URL. The similarity is required in the ranking section. The similarity is computed as.

$$S = sim(U, Unew) + sim(A, Anew) + sim(T, Tnew) \tag{16}$$

After pre-processing is performed, the crawler has a list of keywords. The similarity is computed using cosine similarity. The exact match found means value of cosine similarity is 1, zero otherwise. This system has to generate a repository of words for form generation. Our previous work does not include the return limit for the crawler. In addition to the work is setting the crawling return limit k = 100. Algorithm 1 explains the general steps in hidden web crawling, while steps of enhanced ranking algorithm is explained in Algorithm 2.

**Algorithm 2:** Proposed Enhanced Ranking Algorithm

Step 1 Extract the new coming URL for U, A and T.
Step 2 Identify the domain of the web page under consideration.
Step 3 Order the site frontier according to the similarity.
Step 4 Similarity is computed as the cosine similarity of vectors.
Step 5 Calculate the out of site links for the encountered URL.
Step 6 Calculate the term frequency-inverse document frequency.
Step 7 Calculate the ranking using the formula $(rj) = (1 - w).\delta j + w.(€)/cj$. where $c_j$ is the factor of the network.
Step 8 Repeat steps 1–8 for 'n' number of web pages.

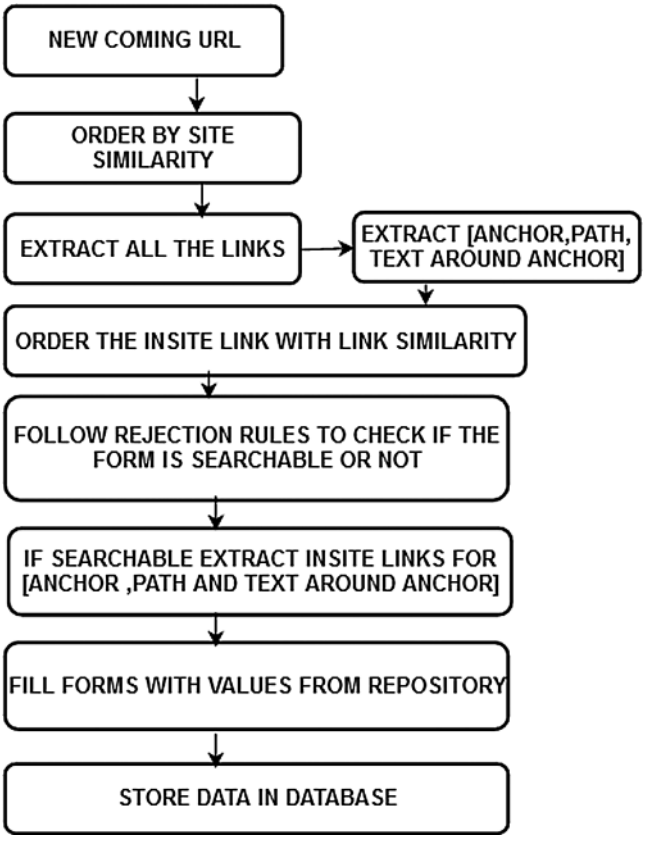


**Figure 3:** Steps involved in the learning process

In hidden web crawling, ranking and learning algorithm depends on each other. The crawler has to learn the path of the searchable forms present on the URL ranked by a crawler. The learning algorithm employs topic similarity, rejection rules and relevance judgement of seed URL. Firstly, the topic similarity will indicate how similar the web pages are comparing to an interesting topic. For example, if the crawler has visited a pre-defined number of forms i.e., 50 new hidden website or 100 new searchable forms. Each time a run of crawler is completed, the feature space of hidden web site and feature space of links are updated. New patterns are reflected due to

periodic updating. Feature space is split into a feature space of a hidden website and feature space of link to allow effective learning of features. We have used a similarity score to calculate how relevant a newly discovered webpage is to the previously discovered web pages. Fig. 3 shows the basic steps of learning in a proposed approach.

There exist numerous ways to compute the topic similarity. But we have focused on the vector space approach to represent the webpage as a vector. The content of the web page is represented by the direction of the vector. The two pages are said to be relevant if their vector points in the same direction i.e mathematically angle between them and cosine similarity is zero and one respectively. In the second phase, the crawler collects better pages depending on previous experiences. In our approach, the crawler starts directly from the home page, and then move to further links. The feature space also includes a path of the URL to help crawler the path of searchable forms.

**Algorithm 3:** Snhanced algorithm for learning new patterns

Step 1 Parse the new URL for U, A, T.
Step 2 Perform ordering of webpages according to similarity.
Step 3 Extract the in-site links for U, A and P.
Step 4 Check if the in-site link is searchable or not.
Step 5 If found searchable learn its path.
Step 6 If the number of forms found on a URL is 100 i.e., learning threshold is reached, move to the next depth.

When the crawler has retrieved searchable forms, the next step is to fill the forms with suitable values. Before that, the system is required to associate the form fields with the domain attributes.

***b. Associating Form Fields and Domain Attributes***

If a form is denoted by F, suppose it is found on an HTML page denoted by W of domain 'd'. Our goal is to find, if F allows queries related to 'd', to be executed over it. For this, the first step is to find text associated with the form field. The system will get these values from the feature vector. For the sake of simplicity, we have considered only URL, anchor and text around an anchor. The second step is to find a relationship between fields and forms w.r.t the attributes of form. This is performed by computing the similarity between the form field text and texts of attributes in the domain. To find the similarity between the text we have applied a cosine similarity measure. Now the crawler has to detect fields of forms. These fields should correspond to the target domain's attributes. The form fields are of bounded and unbounded type. Bounded fields offer a finite list of query values. For example, the select option, radio button and checkbox. While the unbounded fields have unlimited query values. For example, a text box. Our approach is limited to bounded controls only. After the forms are filled with possible values. it could result in 200 (web page correctly submitted), 400 (Bad request response status code), 401 (Unauthorized), 403 (Forbidden client error status response code), 404 (Page not found), 405 (Method Not Allowed response status code), 413 (Payload too large), 414 (URI Too Long response status code), 500 (Internal Server Error), 503 (Service Unavailable), 524 (A time out occurred). The approach is designed to submit forms using both the GET and POST method.

## 4 Result Discussion

The crawler is implemented in python and is evaluated over 11 types of response. In our implementation, site classification is based on a Naïve Bayes classifier trained from samples of the DMOZ directory. As explained earlier different authors have defined coverage in different ways. In our approach, coverage is the number of correctly submitted forms with status code 200. We have excluded those pages which do not generate response. Like if the status code is 400, this means form is submitted with correct values, but response is not generated due to unauthorized status code.

In Tab. 1, the number of forms submitted per domain are shown. The book domain has the highest coverage as compared to others.

**Table 1:** Shows the number of forms retrieved per domain

| Domain | Number ofURLs | 200 | 400 | 401 | 403 | 404 | 405 | 413 | 414 | 500 | 503 | 524 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Book** | 13936 | 9969 | 55 | 71 | 24 | 463 | 2285 | 914 | 25 | 50 | 0 | 9 |
| **Product** | 886 | 496 | 0 | 0 | 240 | 115 | 0 | 0 | 0 | 0 | 35 | 0 |
| **Auto** | 4034 | 29 | 9 | 0 | 0 | 21 | 0 | 263 | 3 | 2 | 3707 | 0 |
| **Flight** | 6016 | 3075 | 254 | 0 | 0 | 338 | 0 | 0 | 57 | 12 | 2280 | 0 |
| **Hotel** | 911 | 398 | 0 | 0 | 17 | 452 | 0 | 0 | 15 | 2 | 27 | 0 |
| **Music** | 84 | 35 | 10 | 0 | 0 | 0 | 0 | 6 | 2 | 0 | 31 | 0 |
| **Premier** | 2 | 2 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Tabs. 2 and 3 show the number forms submitted using GET method and POST method out of the total number of forms per domains.

**Table 2:** Shows the number of forms submitted using the GET method

| Domain | Number ofURLs | 200 | 400 | 401 | 403 | 404 | 405 | 413 | 414 | 500 | 503 | 524 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Book** | 9741 | 3078 | 0 | 0 | 0 | 13 | 2285 | 893 | 0 | 0 | 3471 | 1 |
| **Product** | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Auto** | 29 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Flight** | 723 | 183 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 540 | 0 |
| **Hotel** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Music** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Premier** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

If the form is submitted with status code 200, it means that the form has been submitted with correct values. Sometimes the form is submitted but the response is not generated due to some reasons like internal server error, service unavailable etc. For coverage, we have considered only those pages for which response is generated back.

**Table 3:** Shows the number of forms submitted using the POST method

| Domain | Number ofURLs | 200 | 400 | 401 | 403 | 404 | 405 | 413 | 414 | 500 | 503 | 524 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Book** | 7677 | 6891 | 3 | 24 | 24 | 450 | 0 | 21 | 25 | 2 | 236 | 1 |
| **Product** | 633 | 494 | 0 | 0 | 24 | 115 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Auto** | 99 | 29 | 9 | 0 | 0 | 21 | 0 | 0 | 3 | 2 | 35 | 0 |
| **Flight** | 4422 | 2892 | 254 | 0 | 0 | 404 | 0 | 26 | 57 | 12 | 540 | 0 |
| **Hotel** | 959 | 398 | 0 | 0 | 17 | 452 | 0 | 0 | 15 | 50 | 27 | 0 |
| **Music** | 84 | 35 | 10 | 0 | 0 | 0 | 0 | 6 | 2 | 0 | 31 | 0 |
| **Premier** | 12 | 2 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figs. 4 and 5 shows the number of forms correctly submitted and the comparison of GET and POST method respectively. Our approach has worked better with POST methods. Which indicate the efficiency of the ranking algorithm and form submission method as explained in [12]. Tab. 4 shows the comparison of GET and POST method w.r.t to documents per domain and new documents.

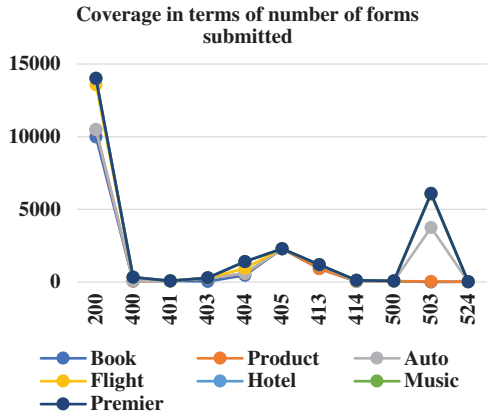


**Figure 4:** Coverage of crawler in terms forms submitted

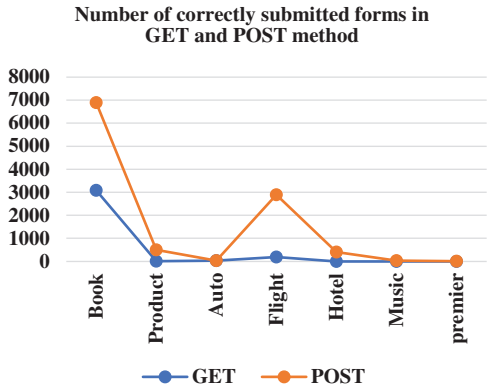


**Figure 5:** Comparison of coverage for GET and POST methods

**Table 4:** Comparison of GET and POST method w.r.t number of documents per domain *vs.* new document captured

| DOMAIN | GET | | | | POST | | | |
|---|---|---|---|---|---|---|---|---|
| | **L** | **Q** | **N** | **UJ** | **L** | **Q** | **N** | **UJ** |
| **BOOK** | 100 | 134 | 13936 | 9661 | 100 | 134 | 13936 | 9677 |
| | 200 | 146 | 29200 | 8791 | 200 | 146 | 29200 | 9781 |
| | 300 | 133 | 53200 | 8902 | 300 | 133 | 53200 | 9992 |
| **PRODUCT** | 100 | 91 | 9100 | 6271 | 100 | 91 | 9100 | 9281 |
| | 200 | 97 | 1900 | 8791 | 200 | 97 | 1900 | 8801 |
| | 300 | 85 | 3400 | 8902 | 300 | 85 | 3400 | 8002 |
| **AUTO** | 100 | 107 | 9100 | 5000 | 100 | 107 | 9100 | 5003 |
| | 200 | 90 | 1456 | 4568 | 200 | 90 | 1456 | 4788 |
| | 300 | 86 | 450 | 678 | 300 | 86 | 450 | 708 |
| **FLIGHT** | 100 | 128 | 11200 | 456 | 100 | 128 | 11200 | 458 |
| | 200 | 129 | 789 | 567 | 200 | 129 | 789 | 867 |
| | 300 | 105 | 2344 | 567 | 300 | 105 | 2344 | 500 |
| **HOTEL** | 100 | 54 | 6767 | 4567 | 100 | 54 | 6767 | 5038 |
| | 200 | 40 | 567 | 20 | 200 | 40 | 567 | 120 |
| | 300 | 51 | 450 | 34 | 300 | 51 | 450 | 71 |
| **MUSIC** | 100 | 39 | 56 | 35 | 100 | 39 | 56 | 30 |
| | 200 | 56 | 45 | 36 | 200 | 56 | 45 | 39 |
| | 300 | 61 | 32 | 4 | 300 | 61 | 32 | 0 |

In Tab. 4, keeping the number of queries same, the methods of submission are compared. Efficiency is compared with respect to unique documents retrieved. From the total number of document new unique documents are calculated. In it, Q (number of queries), N (Number of documents), $U_j$ number of new documents retrieved. Since the method has not performed well in premier domain, so we have skipped its comparison in terms of number of documents. At present we have experimented with only three value of L, i.e., 100, 200 and 300. Another inference from the above table shows that our system has worked well with return limit 100. After return 100, the system retrieved lesser number of unique values.

Our method is static limit based ranking method. If we choose many high frequencies, coverage rate is decreased. This led to skipping some high-ranking documents, this is the reason our system has not worked well with premier. But in future with use of multiple query words, this problem could be overcome. Figs. 6 and 7 shows the comparison of submission methods in terms of new document captured.

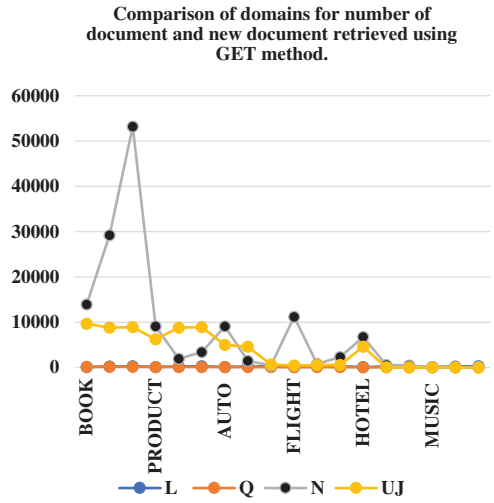


**Figure 6:** Comparison of domains for number of document and new document captured using GET method

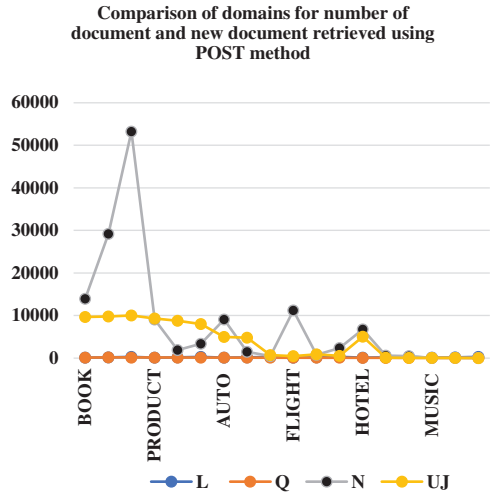


**Figure 7:** Comparison of domains for number of document and new document captured using POST method

## 5 Conclusion

We have proposed an enhanced ranking algorithm for collecting hidden websites based on priority. This paper has tackled problem when the document is missed if it has low rank. This algorithm is a triplet formula to calculate the rank of the website. By including site frequency, the documents which have low rank earlier, can have high rank. By ranking the website, the crawler minimises the number of visits and maximize the number of websites with embedded forms. The convergence of our algorithm is fast as we have also implemented stopping rules and rejection rules. In searching for new rules to improve the efficiency, we have imposed the limit on number of documents to be returned. This has also served as the drawback of the system as the crawler

should not pose any limit on number of documents. One another limitation of the system is in premier domain. The number of forms submitted is very low. For this reason, the domain could not be included in the new document retrieved factor. In future we will improve this area. We have also discussed the stopping criteria. The stopping rules save the crawler from the exhaustive crawling traps. This not only save the memory and time but also help retrieving more unique documents. On the same line we have implemented concept of crawling up to depth of three and after that new URL is picked up from the frontier. The efficiency of the crawler is shown by correctly submitted web pages. The inclusion of more domains and status code remains as future work. We are also going to combine the distance rank algorithm which we believe will yield better results. In future we will also work on unbounded forms.

**Acknowledgement:** We are very thankful for the support from Taif University Researchers Supporting Project (TURSP-2020/98).

**Funding Statement:** Taif University Researchers Supporting Project number (TURSP-2020/98), Taif University, Taif, Saudi Arabia.

**Conflicts of Interest:** The authors declare that they have no conflicts of interest to report regarding the present study.

## References

[1] M. Manvi, A. Dixit and K. K. Bhatia, "Design of an ontology based adaptive crawler for hidden web," in *Proc. CSNT*, Gwalior, India, pp. 659–663, 2013.

[2] M. C. Moraes, C. A. Heuser, V. P. Moreira and D. Barbosa, "Prequery discovery of domain-specific query forms: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 8, pp. 1830–1848, 2013.

[3] J. Madhavan, L. Afanasiev, L. Antova and A. Halevy, "Harnessing the deep web: Present and future," *Systems Research*, vol. 2, no. 2, pp. 50–54, 2009.

[4] Y. He, D. Xin, V. Ganti, S. Rajaraman and N. Shah, "Crawling deep web entity pages," in *Proc. of the Sixth ACM Int. Conf. on Web Search and Data Mining (WSDM '13)*, Association for Computing Machinery, New York, NY, USA, pp. 355–364, 2013. https://doi.org/10.1145/2433396.2433442.

[5] L. Barbosa and J. Freire, "Searching for hidden-web databases," in *Proc. WebDB*, Baltimore, Maryland, vol. 5, pp. 1–6, 2005.

[6] Y. Li, Y. Wang and J. Du, "E-FFC: An enhanced form-focused crawler for domain-specific deep web databases," *Journal of Intelligent Information Systems*, vol. 40, no. 1, pp. 159–184, 2013.

[7] S. Raghavan and H. Garcia-molina, "Crawling the hidden web," in *Proc. 27th VLDB Conf.*, Roma, Italy, pp. 1–10, 2001.

[8] L. Barbosa and J. Freire, "An adaptive crawler for locating hiddenwebentry points," in *Proc. WWW '07*, Banff, Alberta, Canada, pp. 441, 2007.

[9] P. G. Ipeirotis, L. Gravano and M. Sahami, "Probe, count, and classify," *ACM SIGMOD Record*, vol. 30, no. 2, pp. 67–78, 2001.

[10] H. Liang, W. Zuo, F. Ren and C. Sun, "Accessing deep web using automatic query translation technique," in *Proc. FSKD*, Jinan, China, vol. 2, pp. 267–271, 2008.

[11] L. Barbosa and J. Freire, "Siphoning hidden-web data through keyword-based interfaces," *Journal of Information and Data Management*, vol. 1, no. 1, pp. 133–144, 2010.

[12] S. Kaur and G. Geetha, "SSIMHAR-Smart distributed web crawler for the hidden web using SIM + hash and redis server," *IEEE Access*, vol. 8, pp. 117582–117592, 2020.

[13] K. C. C. Chang, B. He, C. Li, M. Patel and Z. Zhang, "Structured databases on the web: Observations and implications," *SIGMOD Record*, vol. 33, no. 3, pp. 61–70, 2004.

[14] A. Ntoulas, P. Zerfos and J. Cho, "Downloading textual hidden web content through keyword queries," in *Proc. JDCL '05*, Denver, CO, USA, pp. 100–109, 2005.