LIST OF PUBLICATIONS


Hybrid Focused Crawling Based Upon VSM Similarity, WordNet Semantics and Hub Score Learning

Mukesh Kumar and Renu Vig
Panjab University

Abstract

New Websites, together with new Web pages, are mushrooming in every corner of the world and gigabytes of information is being uploaded, deleted or modified every unit of time. None of the existing search engines is able to cover the complete Web as a whole for indexing due to the ever increasing size and hence is not able to provide complete and latest information all the times. Users still have to sequentially browse the search results to get the desired information. Also sometimes the search results are biased by willing full access of an unrelated page more times than a related page for some query. Focused crawler provides the solution for growing size of the Web by browsing the portion of the Web that is related to the specific domain. It covers the maximum Web space looking for the contents related to the domain and provides the more recent and exact information. In this paper we present a focused crawler architecture based upon WordNet semantics, Vector Space Model (VSM) and hub score learning. Crawling results for breadth first crawler, VSM based best first crawler, Naive Bayes breadth first crawler, Naive Bayes best first crawler, and crawler based upon WordNet semantics, Vector Space Model (VSM) and hub score learning, are shown. The results show that the proposed crawler outperforms the others in terms of the precision and also outperform all but Naive Bayes breadth first crawler, which produces the worst precision among all the competitors, in terms of average time taken for collecting 1000 domain related pages.

Keywords: Information retrieval, World Wide Web (WWW), data mining, search engines.

1. Introduction and Related Work

The exponential growth of the World Wide Web enforced the universal search engines to address the scalability limitations with huge amounts of hardware and network resources, and also by distributing the crawling process across users, queries, or even client computers. It introduces difficulties to discover topic relevant information that can be used in specialized portals and on-line search. Finding the right information becomes an increasingly difficult task which often leads to undesired results. This motivated the development of some document discovery mechanisms. To tackle this issue the
focused Web crawlers are emerging. A crawler is a program used by a search engine that retrieves Web pages by surfing the Internet from one link to another. Focused crawlers [16] dynamically browse the Web by choosing the most promising links in order to try to maximize the relevancy of the retrieved pages and thus resulting in huge savings in network and computation resources by ignoring non-relevant portions of the Web. Elyasir et al. [7] compared a general crawler against the focused crawler by mentioning resource requirements and use of both the approaches.

Fish search given by Bra and Post [3] was the initial milestone for the Web crawlers. In fish search, every link is a fish whose life depends upon relevancy of visited pages and server speed. Relevance is represented by a binary value. The fish ends her life after tracing a fixed amount of irrelevant Web portion. The path followed by the fish while tracing the relevant portion was shown as the results. The major disadvantage of this approach was that all the relevant pages were represented using a single binary digit and hence limiting the crawler to tilt its way towards more relevant pages than the less relevant pages. Focused crawler was first introduced by Menczer [13] and Chakrabarti et al. [4]. Chakrabarti et al. [4] trained a Naive Bayes classifier on a hierarchical taxonomy and used this classifier to compute the relevance scores of the fetched pages to the topics selected by the user, and then assigned this score to the unvisited URLs extracted from this page as their scores in the crawler’s queue. The crawler was developed based on the concept that relevant pages are likely to introduce other relevant pages.

Menczer et al. [12] proposed a Naive best-first crawler that measures the cosine similarity of a crawled page to a topic and used it to decide whether the hyperlinks contained within the page can act as the future crawl links or not. The use of machine learning algorithms in best-first crawler to estimate the similarity of a crawled page to the topic was proposed by Pant and Srinivasan [15]. They used different classifiers to guide the crawl process and studied the results based upon training time, precision and recall values.

Context graph was introduced by Diligenti et al. [6]. They proposed a focused crawler that models the links and contents of documents that are closely linked to target pages to improve the efficiency with which content related to a desired category could be found. The major limitation of the approach is the requirement of reverse links to exist at a known search engine for a reasonable fraction of the seed set documents.

Arbitrary predicates based crawler was given by Aggarwal et al. [1]. They proposed the intelligent crawling which learns characteristics of the linkage structure of the World Wide Web while performing the crawling. The intelligent crawler uses the in-linking Web page content, candidate URL structure, or other behaviours of the inlinking Web pages or siblings in order to estimate the probability that a candidate is useful for a given crawl.

Cho et al. [5] proposed a method for selecting the next page to crawl based upon page’s similarity to the query, back link count, forward link count, PageRank [14] and location metric. Stanford Web pages were taken for studying the crawl results. They showed that the pages which are having some query terms as the part of their URL, and the pages which are having some terms from the query as their anchor text can act as
the promising links for the future crawl. Horng et al. [8] proposed a fuzzy information retrieval system based upon term reweighing.

Chakrabarti et al. [4] proposed topic distillation into the crawling process which implements a modified version of Kleinberg’s hyperlink-induced topic search (HITS) algorithm [10] to identify good hub pages. These hubs were then revisited and the priorities of the unvisited URLs cited by them were raised.

Rungsawang and Angkawattanawit [18] proposed a focused crawler that makes use of the previous crawl results to learn the more topic specific keywords, and to better predict the next URL for crawling. Taylan et al. [20] proposed the use of Naive Bayes classifier to classify the unseen links. After classification a URL score was associated with each link based upon which the future crawl links were selected.

Bergmark et al. [2] explored the use of focused crawling in generation of digital libraries, and also discussed the problem of reaching to the relevant pages through irrelevant pages. Wang et al. [21] proposed a frontier prioritizing algorithm, OTIE (On-line Topical Importance Estimation). It combines link- and content-based analysis to find the priority of an unseen URL in the frontier. They compared harvest rate and target recall of the proposed algorithm with other crawling algorithms: breadth-first, link-context-prediction, on-line page importance computation.

Mangaravite et al. [11] presented a link context based focused crawling technique. The work shows that the link context lonely can also serve good for quality Web page retrieval from the Web. Safran et al. [19] presented a learning-based focused crawling approach. They used four relevance attributes to predict the relevance of unvisited URLs namely: URL words, its anchor text, the parent pages, and the surrounding text. Naïve Bayesian classification model was used as domain relevancy decider.

Most of the approaches in the literature makes use of keyword search, or some other very complex machine learning approaches to judge whether a page is a candidate for future crawl or not. There is a great need for improvement in this judgement criterion by using some semantic measures along with others to effectively guide the crawler. The use of WordNet database as word’s semantics extractor is not yet explored by any focused crawler. Also the use of current crawling phase results for generating seed pages for the next crawling phase based upon hub score and semantic score is not yet explored.

2. Our Work

Throughout our work we designed and developed five variants of focused crawlers including the proposed one. The design issues common for each crawler and for individual are discussed below:

Seed Pages: Focused crawler needs to be activated from the pages which are highly related to the crawling domain. The set of pages are termed as seed pages. The process of generation of seeds is to be automated because the manual collection of seeds may result in the biased crawling direction due to the individual’s own interest and likes.

Document Fetching: The links contained within the fetched page are to be extracted and inserted into the crawler queue based upon their similarity.
Pre-Processing: Downloaded pages are to be preprocessed before making their use to guide the crawler. This pre processing involves stop words removal and stemming of each word in the document to its root level, and also to bring the whole page in the form of a vector to be used in similarity calculations for various crawlers.

2.1. Hybrid Crawler.

The proposed hybrid crawler makes use of three criterions for guiding the future crawl and accordingly to reorder the Crawler Queue to retrieve the best possible results. The three criterions used are:

a. Content similarity and link similarity using VSM model.
b. Semantic similarity using WordNet semantics.
c. Next phase seed selection based upon Hub scores.

VSM assumes that \( t \) distinct terms are there in the document. Each of these \( t \) terms will form a vector space of \(|t|\) dimensions. Each term of this vector will be given a weight \( w_{ij} \). Now the document as well as the query is represented in the form of a \( t \)-dimensional vector

\[
d_j = (w_{ij}, w_{2j}, \ldots, w_{tj}).
\]

In this way collection of \( n \) documents can be represented as a matrix of the form

\[
\begin{bmatrix}
w_{11} & w_{21} & \cdots & w_{t1} \\
w_{12} & w_{22} & \cdots & w_{t2} \\
\vdots & \vdots & \ddots & \vdots \\
w_{1n} & w_{2n} & \cdots & w_{tn}
\end{bmatrix}.
\]

An entry in the matrix corresponds to the weight of the term in the document, zero means the term has no significance in the document. This weight may be anything that contains the degree of relevance to the documents collection. Many variants are proposed for this weight calculation, \( tf - idf \) weights are used most efficiently for calculation of this term weight.

\( tf - Idf \) Weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The term frequency \( tf_{t,d} \) of term \( t \) in document \( d \) is defined as the number of times that \( t \) occurs in \( d \), \( df_t \) is the document frequency of \( t \), and means the number of documents that contain \( t \). The \( df_t \) is an inverse measure of the informativeness of \( t \) also \( df_t \leq N \) where \( N \) is the total number of documents in the Relevant Page Set. Then the \( idf \) (inverse document frequency) of \( t \) is given by

\[
idf_t = \log(N/df_t).
\]

The \( Tf - Idf \) weight of a term \( t \) in the document \( d \), \( W_{t,d} \) is the product of its \( tf \) weight and its \( idf \) weight and will be given by

\[
W_{t,d} = \log(1 + tf_{t,d}) \times \log(N/df_t).
\]
WordNet [8, 17] provides an online lexical database for English, which is freely and publicly available for download. Its structure makes it a useful tool for computational linguistics and natural language processing. Words that are found in close proximity to one another in the network are semantically disambiguated. Hybrid crawler makes use of WordNet to find the words which are semantically similar to the words present in the VSM Weight Vector, which in turn came from the documents highly related to the crawler domain (after removing stop words). Now the VSM Weight Vector contains the old term and weights as well as the words semantically similar to the existing terms with score equal to that of the original term, this new vector is named as Semantic VSM Weight Vector (SVWV). It will be used by the Hybrid crawler to deduce the semantic similarity, content similarity, and link similarity of the candidate page for possible inclusion into the future crawl page category.

The first phase of our Hybrid crawler functions as according to the following algorithm

**Initiate Hybrid Crawler**

*Input:* Open Directory Project (ODP) database, domain topic for focused crawler, WordNet.

*Output:* The set of pages, R, who’s Semantic VSM Weight Vector (SVWV) score is amongst the highest.

1. Initialize the seed pages from the Open Directory Project (ODP).
2. Remove stop words and apply stemmer to each individual document in the seeds.
3. Generate $tf-idf$ weight score matrix for the seed pages.
4. Generate a single vector comprising term and the sum of $tf-idf$ score of the term from all the documents vectors; name this vector as VSM Weight Vector.
5. For each term present in the VSM Weight Vector
   a. Find the words which are semantically similar to the term by using WordNet semantic relations, and assign them the score as that of the original term from the VSM Weight Vector and insert $<term, weight>$ into the Semantic VSM Weight Vector (SVWV).
6. For each seed page
   a. Calculate similarity score of the page’s contents from the Semantic VSM Weight Vector (SVWV).
   b. Insert $<page, similarity score>$ in the Crawler Queue, which is a priority queue maintained upon the score numeric field.
7. While Crawler Queue, which is a priority queue, is not Empty
   a. Pop topmost URL from the Crawler Queue, and fetch it from the internet.
   b. Calculate its content’s similarity score from the Semantic VSM Weight Vector (SVWV), and insert the page into the results database, R.
c. For each link present in the page calculate score as
\[
\text{score} = \frac{\text{parent.content.similarity}}{7} + \text{link.anchor.similarity.}
\]
d. Insert the links into the Crawler Queue as according to their score's location.

After execution of the Initiate Hybrid Crawler, we have the set of pages R. Now we have to find the best hubs out of these pages, but before going to that we should know what a hub is? Hub URL is the one which is pointing to many other URLs and authority URL is the one which is pointed to by many URLs. Best hub is the one which is pointing to many relevant pages and the best authority is the one which is pointed to by many relevant pages. To calculate a hub score and an authority score, we use HITS family’s algorithm [10]. Hubs and authorities exhibit mutual reinforcement relationship. The algorithm for finding hub score and authority score is given below for the set S.

**Hub Authority Calculation Algorithm**

**Input:** Set of pages, S  
**Output:** Hub and Authority score for each page in S

1. For all pages p in S
   a. \(HUB_p = AUTHORITY_p = 1\);
2. For all pages p in S
   a. \(HUB_p = \sum_{Q \in S \exists \text{Link}(P \rightarrow Q)} \text{AUTHORITY}_Q\)
   b. \(AUTHORITY_p = \sum_{Q \in S \exists \text{Link}(Q \rightarrow P)} \text{HUB}_Q\)
3. Repeat Step 2 Until HUB and AUTHORITY vectors do not change.
4. Normalize the HUB and AUTHORITY vectors.

Hub Authority Calculation algorithm takes set, S, as input and calculates hub and authority score for each page present, as accordingly to their mutually reinforcing relationship. Initially hub and authority score for each page in S is initialized to 1. The hub score of a page, P, is calculated to be the sum of authority score of all the pages, Q, in S for which there is a link from P to Q and authority score for the page is determined in the same way.

**Hybrid Crawling Algorithm**

**Input:** Set of page, R, generated from the previous crawling attempt.  
**Output:** Set of pages, T, related to the domain in some greater degree than from the previous crawl.

1. Generate normalized HUB and AUTHORITY vectors for all the pages in R using Hub Authority Calculation Algorithm.
2. For each page p in R

3. Insert $M$ number of pages $p$ from $R$, which are having highest $RANK_p$ score, in the Crawler Queue, which is a priority queue maintained upon the score numeric field.

4. While Crawler Queue is not Empty
   a. Pop topmost URL from the Crawler Queue, and fetch it from the Web.
   b. Calculate its content’s similarity score from the Semantic VSM Weight Vector (SVWV), and insert the page into the results database, $T$.
   c. For each link present in the page calculate score as
      \[
      \text{score} = \frac{\text{parent.content.similarity}}{7} + \text{link.anchor.similarity}
      \]
   d. Insert the link into the Crawler Queue as according to its score’s location.

While predicting the similarity of a link with in a page the similarity is taken in account because of the fact that average number of out links from a page is 7 as suggested by Kumar et al. [9], and hence the total similarity of the parent is to be divided into seven equal parts. Depending upon the number of crawls to be made, the set, $T$, will act as the set, $R$, for the next crawling attempt and the process moves on. At the end of each crawling attempt the set of pages which are going to act as seeds are being learned based upon the hub scores, and the results shows that this learning contribute towards better precision rate of the crawler.

2.2. Breadth first search based focused crawler

Breadth first crawler starts from the seed pages (pages highly related to the crawler domain) and download them in first come first serve basis. The URLs contained within the document are extracted and inserted at the end of the crawler queue. The algorithm for the crawler is given below:

**Breadth First Search Crawler**

*Input: Domain topic and Seed pages from Open Directory Project (ODP) database.*

*Output: The set of pages retrieved by the crawler*

1. Initialize the seed pages.
2. Put all the seeds into the crawler queue.
3. While Crawler Queue is not Empty
   a. Fetch the first URL from the Crawler Queue; download the corresponding page from the internet. Calculate page’s similarity to the domain and put into the Results database. Extract all the links contained within the page and insert them at the end of the Crawler Queue.

The Breadth First Search Crawler works on the principle of BFS, which forbid the crawler to go for the next node until all the nodes at a specific level are not completely visited.
2.3 Vector Space Model (VSM) based best first crawler

Vector space model based best first crawler works on the VSM based similarity measure to guide the crawler.

**VSM Best First Search Crawler**

*Input: Domain topic and Open Directory Project (ODP) database.*

*Output: The set of pages retrieved by the crawler based upon VSM score*

1. Initialize the seed pages from the Open Directory Project (ODP).
2. Remove stop words and apply stemmer to each individual document in the seeds.
3. Generate $tf-idf$ weight score matrix for the seed pages.
4. Generate a single vector comprising term and the sum of $tf-idf$ score of the term from all the documents vectors; name this vector as VSM Weight Vector.
5. Insert all the seed URLs into the crawler queue as according to their content’s similarity total score from the VSM weight vector.
6. While Crawler Queue, which is a priority queue, is not Empty
   a. Pop topmost URL from the Crawler Queue, and fetch it from the internet.
   b. Calculate its content’s similarity score from the VSM weight vector, and insert the page into the results database.
   c. For each link present in the page calculate score as
      \[
      \text{score} = \frac{\text{parent.content.similarity}}{7} + \text{link.anchor.similarity}.
      \]
   d. Insert the links into the Crawler Queue as according to its score's location.

2.4 Naive Bayes classification based focused crawlers

Naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature, given the class variable. A mango can be called as ripe if it is pulpy and its colour is yellow. Even if these features depend on each other or upon the existence of the other features, a Naive Bayes classifier considers all of these properties to independently contribute to the probability that this mango is ripe. Naive Bayes classifiers can be trained in a supervised learning setting. The Naive Bayes is trained as according to the following algorithm in our implementation

**Naive Bayes Training Algorithm**

*Input: Collection $D$ having $D_R$, set of domain related pages, and $D_NR$, set of pages not related to the domain.*

*Output: Probability of each term's belonging to the related or non-related class.*

1. Let $D$ denotes the collection of domain related and non domain related documents.
2. $D_R$ and $D_{NR}$ are collection of documents which are related to the domain and which are not related to the domain respectively, and both of these are subsets of $D$, and let $T_R$ and $T_{NR}$ be the concatenation of documents in $D_R$ and $D_{NR}$ respectively.

3. Let $V$ denotes the vocabulary of all words in $D$.

4. Calculate probabilities

$$P(C_i) = \frac{|D_i|}{|D|} \text{ for } i = R \text{ and } NR \text{ (Related and Non Related).}$$

5. Set $N_R = |T_R|$ and $N_{NR} = |T_{NR}|$.

6. For each word $W_j \in V$
   a. Set $n_{ij}$ to the number of occurrences of $W_j$ in $T_i$ for $i = R$ and $NR$.
   b. Calculate $P(W_j|C_i) = \frac{n_{ij}+1}{n_i+|V|}$ for $i = R$ and $NR$.

**Naive Bayes Classifier Algorithm**

*Input: Text document $X$*

*Output: Class of the document (whether related or not related to the domain)*

1. Let $n$ be the number of word occurrences in $X$.

2. Calculate class

$$\text{Class}_i = P(C_i) \times \prod_{j=1}^{n} P(a_j|C_i) \text{ for } i = R \text{ and } NR, \text{ where } a_j \text{ is the word occurring at } j^{th} \text{ position in } X.$$  

3. If $\text{Class}_R > \text{Class}_{NR}$ then return Related else return Not Related.

Based upon the Naive Bayes classifier, two crawlers one based upon the breadth first search and other based upon the best first technique were developed, the crawler’s algorithms for both are given below:

**2.4.1. Naive Bayes Breadth First Crawler Algorithm**

*Input: Domain topic and Open Directory Project (ODP) database.*

*Output: The set of pages retrieved by the crawler*

1. Initialize the seed pages from the Open Directory Project (ODP).

2. Train Naive Bayes as according to the Naive Bayes Training Algorithm by considering all the seed pages as in the set of Relevant documents and other documents from ODP categories as Non Relevant Documents.

3. Insert all the seed URLs into the Crawler Queue.

4. While Crawler Queue not Empty
   a. Pick the URL from the crawler Crawler Queue.
   b. Fetch the URL from internet.
   c. Deduce the class of the page by using Naive Bayes Classifier Algorithm.
d. If (page belongs to the Relevant class)
   i. Extract the links contained with in document and insert them at the end of
      the Crawler Queue.
   ii. Insert the page itself into the result database.

2.4.2. Naive Bayes Best First Crawler Algorithm

*Input: Domain topic and Open Directory Project (ODP) database.*

*Output: The set of pages retrieved by the crawler*

1. Initialize the seed pages from the Open Directory Project (ODP).
2. Remove stop words and apply stemmer to each individual document in the seeds.
3. Generate \( tf-idf \) weight score matrix for the seed pages.
4. Generate a single vector comprising term and the sum of \( tf-idf \) score of the term
   from all the documents vectors; name this vector as VSM Weight Vector.
5. Train Naive Bayes as according to the Naive Bayes Training Algorithm by considering
   all the seed pages as in the set of Relevant documents and other documents from
   ODP categories as Non Relevant Documents.
6. Insert all the seed URLs into the Crawler Queue as according to their content’s total
   score from the VSM weight vector.
7. While Crawler Queue, which is a priority queue, is not Empty
   a. Pop topmost URL from the Crawler Queue, and fetch it from the internet.
   b. Deduce the class of the page by using Naive Bayes Classifier Algorithm.
   c. If (page belongs to the Relevant class)
      i. Insert the page itself into the result database with its VSM weight similarity
         score.
      ii. For each link present in the page calculate score as
         \[
         \text{score} = \frac{\text{parent.content.similarity}}{7} + \text{link.anchor.similarity}.
         \]
      iii. Insert the links into the Crawler Queue as according to their score’s location.

3. Experimental Setup and Results

All the above mentioned work is implemented in Java using a Windows7 setup on
an Intel(R) Core(TM) i5 Processor 3.20 GHz and 4.00 GB of RAM, and using MySQL
as the backend database. Crawler performance is generally measured in terms of harvest
rate or precision value which can be seen as

\[
\text{Harvest Rate OR Precision} = \frac{\text{Total Number of related pages retrieved}}{\text{Total number of pages retrieved}}. \tag{2.5}
\]
HYBRID FOCUSED CRAWLING

The pages which are having a VSM Weight Vector or Semantic VSM Weight Vector (SVVWV) score greater than a predefined threshold value are considered as relevant and others as non relevant to the crawler domain.

An automated process for gathering seed pages is used. Open Directory Project (ODP) provides the categorical collection of URLs that are manually edited and not biased by any commercial user. From here we can find individual categories link. One can find maximum number of URLs related to the concerned topic by just browsing the http://www.dmoz.org from root to the topic of interest. For analysis and other purposes http://www.dmoz.org also provides dump for the whole database, which can be used as according to the needs of the individual. We downloaded this dump and extracted the contents to the MySQL using PERL (Practical Extraction and Report Language) scripts. For our experiment we retrieved the URLs belonging to the categories which are pointing to “computer” at the last level. A total of 563 such URLs were found through a module written in Java. These 563 URLs acted as the initial seed pages for all the crawlers. The harvest ratio graph for different similarity values is shown in Figure 1. In this figure HYBRID1, HYBRID2 and HYBRID3 are the names of the consecutive runs of the proposed hybrid crawler.

Our next experimental domain is “agriculture”. We retrieved the URLs belonging to the categories which are pointing to ”agriculture” at the fifth level. A total of 323 such URLs were found from the ODP. These URLs acted as the initial seeds for all the crawlers. The precision, which is the percentage of the total related pages retrieved from the total pages retrieved, is calculated by considering the pages having their similarity score greater than a predefined score. The precision graphs for different thresholds are shown in Figure 2 to Figure 4.
Figure 1 shows that the consecutive runs of the Hybrid crawler outperform all the present crawlers for the "computer" domain. Also, harvest shows an upward trend with increase in number of consecutive crawls of the proposed Hybrid crawler.

Figure 2 to Figure 4 shows that the precision rate for the "agriculture" domain. Hybrid crawlers performed better than for all the similarity measures in this case also, and outperformed all other crawlers.

In all the crawlers the page is to be fetched from the Web and is to be pre-processed. The proposed approach consumes a constant amount of time for using WordNet and to decide upon the number of pre-runs for deciding HYBRIDn th crawler. But all that time consumed could be ignored while finding growth rate of the crawler's performance. However, the time comparison demands some perfect system conditions like network...
traffic, internet speed, server response time etc. We compared the different crawlers against the time taken to collect 1000 domain related pages. The time is measured in milliseconds, and a graph is plotted as shown in Fig.: 5.

![Precision graph](image)

Figure 4: Precision graph for similarity value 30 or more for “agriculture” domain.

The average time taken by various crawlers for collecting 1000 domain related pages is shown in Fig.: 6.

![Time performance graph](image)

Figure 5: Time performance graph, execution time (milliseconds) on vertical axis, range of pages collected by various crawlers on Horizontal axis.

By Figure 5 and Figure 6, we deduce that the proposed crawler performs the second best in measuring the average time taken for collecting 1000 domain related pages. It
outperformed all but Naive Bayes breadth first crawlers. But when we go to the precision comparison, the Naive Bayes Breadth First crawler performs the worst. This fact justifies the applicability of the proposed crawler.

4. Conclusion

We developed five different approaches for focused crawling through Web including the proposed one. All crawlers were activated for collecting pages related to “computer” and “agriculture” domain. Precision rate was calculated for each of the crawling approach by collecting 5000 pages for each approach. Graphs were drawn for the precision rate, and time taken for collecting 1000 pages. The results show that the proposed hybrid crawlers outperformed the conventional breadth first crawler, VSM best first crawler, Naive Bayes Breadth first crawler and Naive Bayes best first crawler in terms of precision and also outperformed all but Naive Bayes Breadth first crawler, which is having the worst precision performance amongst all.

References


Department of Computer Science and Engineering, University Institute of Engineering and Technology, Panjab University, Chandigarh, India.
E-mail: mukesh_rai@yahoo.com
Major area(s): Web mining, algorithms and soft computing.

Department of Electronics and Communication Engineering, University Institute of Engineering and Technology, Panjab University, Chandigarh, India.
E-mail: remuvig@hotmail.com
Major area(s): fuzzy logic, data mining and image processing.

(Received September 2012; accepted July 2013)