2013 5th International Conference on Computational Intelligence and Communication Networks

An Improved Indexing Mechanism to Index Web Documents

Pooja Mudgil
Department of CSE, Banasthali Univ.
Rajasthan, India
engineer.pooja90@gmail.com

A. K. Sharma
Department of CSE, YMCAU
Faridabad, Haryana, India
ashokkale2@rediffmail.com

Pooja Gupta
Department of CSE, GGSIPU, MAIT
Rohini, Delhi, India
poojaguptamait@gmail.com

***Abstract*— Owing to the dynamic nature of the web, it is difficult for search engine to find the relevant documents to serve a user query. For this purpose, search engine maintains the index of downloaded documents stored in the local repository. Whenever a query comes search engine searches the index in order to find the relevant matched results to be presented to the user. The quality of the matched result depends on the information stored in the index. The more efficient is the structure of index, more efficient the performance of search engine. Generally, inverted index are based solely on the frequency of keywords present in number of documents. In order to improve the efficiency of the search engine, an improved indexing mechanism to index the web documents is being proposed that keeps the context related information integrated with the frequency of the keyword. The structure is implemented using *Trie*. The implementation results on various documents show that proposed index efficiently stores the documents and search is fast.**

Keywords- WWW, World Wide Web, Search Engine, URL, Indexing, Context, Web Documents

I. INTRODUCTION

WWW is a huge repository of hyperlinked web documents accessible through the internet. Search engines are the tools for extracting and exploring specific information on the web. With the rapid growth of technology, internet has become the prime source for information research and also one of the most important media of information exchange. Search engines include various components for simplifying the task of searching the enormous WWW. Otherwise, it is very complex to search such a huge collection of web pages and get the result in few seconds.

A basic search engine consists of three parts: crawler, indexer and query processor. The Crawler also called the spider traverses the web collecting information and stores them into a huge repository after being compressed. It follows hyperlinks across the web collecting information from HTML web pages. Indexing module prepares the index of the local database. Every Web document has an associated ID number called document identifier, which is assigned whenever a new URL is parsed out of a web page. The indexer takes the web pages collected by the crawlers and stores them into a highly efficient index. Query module handles the user queries by searching the index. The core of search engine is an index module which enhances the speed of searching the required information with help of an efficient indexing technique.

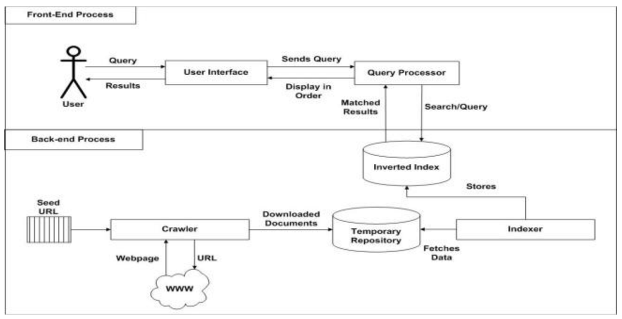


Figure 1. Search Engine Architecture

A basic architecture of search engine shown in Fig 1. It consists of two layers: Font-end and Back-end. Front-end composed of a two main components: User Interface and Query Processor. User Interface is used to interact with the user. Then user interacts with search engine by giving a query which acts as an input to the query processor. The query processor searches the result to get the desired match results and rank them in some order and then give it to user interface to display it to the user. Back-end layer consist of crawler and Indexer. Crawler starts it working with list of seed URL as an initial input. It traverses the WWW to download the corresponding web page and stores them into temporary repository. Indexer fetches the downloaded documents from temporary repository, extract all the keywords, stop words are removed and stem the root word and stores it in an Inverted Index.

Indexing is the process of systematic arrangement of records. An index enables user to find records more quickly. Without them, researcher might have to look through hundreds or thousands of records to locate an individual record. The existing Indexing technique stores the web pages only in the form of keywords present in them. However, ontology based Indexing using contextual senses

[8] has tried to store some context information but they have not considered the importance of the keyword. Keyword alone is not sufficient for retrieving information about the web page. The presence of the keyword in various HTML tags of web documents should also be considered for indexing the web pages.

The proposed contextual based indexing has considered the presence of keywords in various HTML tags of web documents such as head, title, keyword, description, body and link. The weight is assigned to each of these tags and stores it using trie structure. This will help to optimize the speed and performance in finding relevant documents for a search query.

II. RELATED WORK

L. Huilin [1] has proposed an efficient crawling strategy for focused search engine comprising of two components; filtering and link forecasting. The first component filters out the irrelevant documents from previously fetched list of documents and the second component envisage the best matched links form related documents to assign them to crawler for further downloading. Thereafter, does the same for the irrelevant documents in support of another topic.

Another technique proposed is the semantic indexing technique [6], which shows improved effectiveness over the classic word based indexing techniques. Herein a Boolean Information retrieval system adds word semantics to the classic word based indexing. Two of the core tasks of the system, namely the indexing and retrieval components, use a combined word- based and sense-based approach. The key to system is a methodology for building semantic representations of open text, at word and collocation level.

The topic specific information is retrieved by Focused search engine [4] by assembling the web pages for diverse topics for post-processing. The performance of a focused crawler is enhanced because it provides explicit topic specific documents.

A research by HAWK crawler [7] is based on the content of the document and on link analysis. X. Chen, X. Zhang, 'HAWK: A Focused Crawler with Content and Link Analysis' is implemented using user-defined relevant formula, shark-search and Page-Rank.

Soumyadeb Mitra [5] has put forth a procedure to update an inverted index based on merging of the posting lists of terms. He has evaluated jump index, which is an efficient index for join queries. Jump indexes execute insert and lookup operations on number of indexed documents.

Context Based Indexing of Web Document using Ontology [8]: - Another proposed technique is an indexing structure in which index is built on the basis of context of the document rather than on the terms basis using ontology. The ontology-based collection selection method uses context to portray collections and search engines. The context of the documents being collected by the crawler in the repository is being extracted by the indexer using the context repository, thesaurus and ontology repository and then documents are indexed according to their respective context.

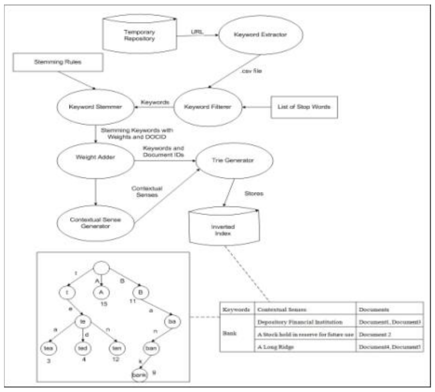
It has been observed from the existing techniques that following technical gaps are there for accessing the index: Focused search engine does not consider the context of the keywords of the user query. Whereas it is essential to focus on the context of the keywords present in the query as well as in the web documents resulting in comparatively smaller searching zone capable of providing more specific web page. Contextual based indexing using Ontology considered terms only in title of the document with maximum frequency. In ontology, user has to pass the context along with query keyword which creates problem with naive users who are not aware of different contextual senses of query.

III. PROPOSED ARCHITECTURE

The existing architecture [12] of search engine shows that the index is built on the basis of the terms of the document and consists of an array of the posting lists where each posting list is associated with a term and contains the term as well as the identifiers of the documents containing the term. The current information retrieval systems use terms to describe documents and search engines. In this work, contextual indexing is being proposed which uses terms with their contextual senses to provide the more relevant results.

The proposed contextual based indexing considers the senses of the keyword. Contextual senses are the different meanings of the same word. Different users type the same query to get the different results according to their interest. For Example, the keyword bank has different contextual meanings like depository financial institution, a long ridge or pile, a stock held in reserve for future use etc. So the proposed contextual based indexing will provide the different senses to the user which will help in generating the more relevant results to the user.

The proposed architecture for indexer consists of following components: Keyword Extractor, Keyword Filterer, Keyword Stemmer, Weight Adder, Contextual Sense Generator, Trie Generator and Inverted Index. Keyword Extractor will extract the keywords from web pages stored in temporary repository and csv(comma separated values) file will be sent to the keyword Filterer. Keyword Filterer matches the list of keywords from csv file with list of stop words and matched words will be eliminated from the list. Keyword Stemmer stems list of keywords to its root from like playing to play etc. Then duplicate keywords will be merged and their weights will be added. Then Trie Structure will be generated for the keywords, documents and senses from contextual sense generator. Inverted Index will be created which consists of keyword, Contextual Sense and Document Ids where the keywords are present within. The proposed indexing architecture is shown in Fig 2.



Keywords	Contextual Senses	Documents
Bank	Depository Financial Institution	Document1, Document3
	A Stock hold in reserve for future use	Document 2
	A Long Ridge	Document4, Document5

Figure 2. Proposed Architecture for Indexer

The detail working of each component is described below:

1) Keyword Extractor: Keyword is the smallest unit of a document which contributes to the meaning of document. Keyword Extractor parses the web documents retrieved from temporary repository. It extracts all the keywords present in web document from different HTML tags such as body, head, title, keyword, description, link and their frequencies in respective tags. The specific weights are assigned to the different HTML tags. Then keyword extractor creates a list of all keywords present in that web document with their corresponding Weight, Percent and Total Frequency. This list is then stored in comma delimited report format.

2) Keyword Filterer: Stop words are the keywords those occur frequently in the web page but do not contribute to the context of web document. The keyword filterer filter out the stop words from the documents. The list of all the stop words is stored in a file, for example, a, an, the, and, it, to, in, of etc. This list is then matched against the list read from the keyword list of document and the words which are common in both the lists are removed from the csv file.

3) Stemmer: Stemming is a process which reduces a word to its stem or root form. It is performed generally by splitting off the suffix of the words. For example, the word banking is reduced to bank by splitting (ing) from the suffix. Thus, the key terms of a query or document are represented by stems rather than by the original words. For this purpose, Porter Stemming Algorithm[14] is used. After stemming has been done, the new result is stored in the file for the next module processing.

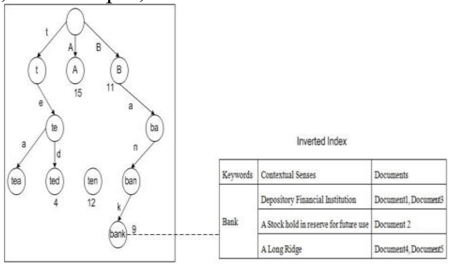
*4) Weight Adder:*It searches the file after stemming to find out the occurrence of a keyword at multiple places in different form of text fonts. For example, the occurrence of keyword "Bank" in the web document can be in different forms such as Bank, bank, BANK. It converts all these occurrences of word to lowercase (bank). It updates the file by the occurrence of that keyword by one and the corresponding weight by the sum of weights of all the earlier occurrences.

5) Contextual Sense Generator: It generates the different senses or meanings of the same keyword. For Example, contextual senses of word colt is a kind of revolver and a young male horse under the age of four. These senses are stored in file to facilitate the user in selecting the proper context of his query keywords.

6) Trie Generator: It generates the data structure in the form of trie which actually produce the index of all these keywords or terms of a web document. Trie is used to store pieces of data that have a key and possibly a value which is any additional data associated with the key. Trie has its origin from the core section of the word **"retrieval",** and this origin hints on its usage: information retrieval systems. This data structure is chosen because it can extract information in a computational complexity of the order of one.

The hierarchical depth of the trie data structure depends on the amount of data stored in it. Each element of data is stored at the highest level of the hierarchy that still allows a unique retrieval. The data structure has been maintained in search-insert fashion, i.e., when a new word is encountered, it get inserted at proper places and accordingly the trie is updated.

Trie is tree type structure which will store the keyword, for example, bank in tree form as shown in Fig 3.



Keywords	Contextual Senses	Documents
Bank	Depository Financial Institution	Document1, Document3
	A Stock hold in reserve for future use	Document 2
	A Long Ridge	Document4, Document5

Figure 3. Trie Structure

Here, node 9 points to the inverted index contain different contextual senses of the node 9 i.e. bank.

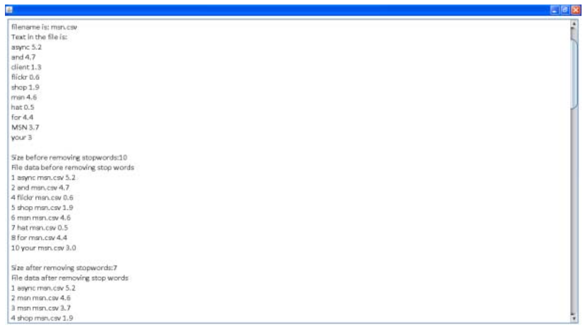
7) Inverted Index: It is created by trie generator as explained in point 6. Each entry of inverted index is a data structure which stores the keyword, meanings and corresponding document ids. By a tree searching algorithm, this inverted index can be explored to acquire a match for a particular query keyword in search-insert fashion.

IV. RESULTS

The proposed indexing algorithm has been implemented to evaluate the performance of the algorithm. This technique is implemented using Java as front-end and Oracle 10g Express Edition as the back-end.

The processing of file includes: removal of stop words, stemming, adding of weights and indexing of file.

a) The implementation of first step of processing the documents i.e. the removal of stop words has been shown in Fig 4.

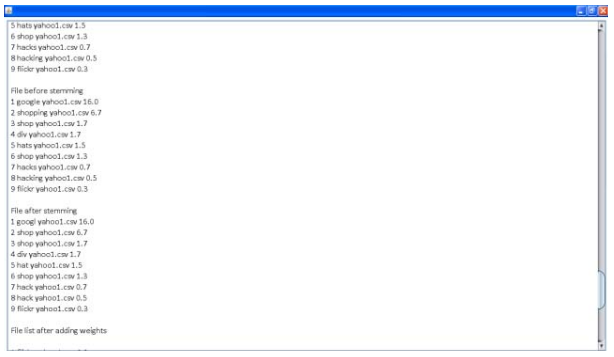


```
filename is: msn.csv
Text in the file is:
async 5.2
and 4.7
client 1.3
flickr 0.6
shop 1.9
msn 4.6
hat 0.5
for 4.4
MSN 3.7
your 3

Size before removing stopwords:10
File data before removing stop words
1 async msn.csv 5.2
2 and msn.csv 4.7
4 flickr msn.csv 0.6
5 shop msn.csv 1.9
6 msn msn.csv 4.6
7 hat msn.csv 0.5
8 for msn.csv 4.4
10 your msn.csv 3.0

Size after removing stopwords:7
File data after removing stop words
1 async msn.csv 5.2
2 msn msn.csv 4.6
3 msn msn.csv 3.7
4 shop msn.csv 1.9
```

Figure 4. Processing of documents (Stop word removal)

The Figure 4 shows that the name of the file selected is ‘msn.csv’ proceeding the list of keywords present in the file and their corresponding weights. It indicates the size of the file before processing and then after the removal of stop words. It is clear from the results that ‘and’, ’for’ and ‘your’ has been removed.

b) The implementation of second step of processing the documents in which stemming on keywords are performed are as shown in Figure5:

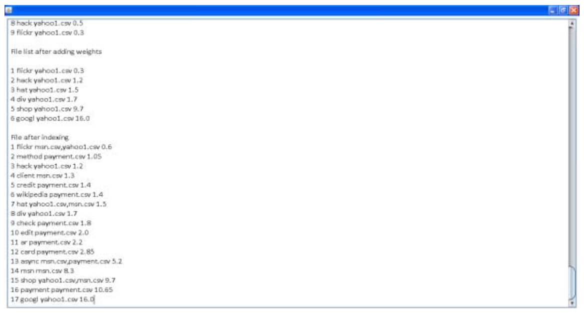


```
5 hats yahoo1.csv 1.5
6 shop yahoo1.csv 1.3
7 hacks yahoo1.csv 0.7
8 hacking yahoo1.csv 0.5
9 flickr yahoo1.csv 0.3

File before stemming
1 google yahoo1.csv 16.0
2 shopping yahoo1.csv 6.7
3 shop yahoo1.csv 1.7
4 div yahoo1.csv 1.7
5 hats yahoo1.csv 1.5
6 shop yahoo1.csv 1.3
7 hacks yahoo1.csv 0.7
8 hacking yahoo1.csv 0.5
9 flickr yahoo1.csv 0.3

File after stemming
1 googl yahoo1.csv 16.0
2 shop yahoo1.csv 6.7
3 shop yahoo1.csv 1.7
4 div yahoo1.csv 1.7
5 hat yahoo1.csv 1.5
6 shop yahoo1.csv 1.3
7 hack yahoo1.csv 0.7
8 hack yahoo1.csv 0.5
9 flickr yahoo1.csv 0.3

File list after adding weights
```

Figure 5. Processing of documents (Stemming)

It has been shown in Fig 5. that stemming is done successfully as all the keywords present in other form are changed to their root form. For example, the keywords hacks and hacking are stemmed to hack. Similarly, all other keywords are also stemmed.

c) The implementation of third step of processing of documents i.e. addition of weights are as shown in Fig 6.

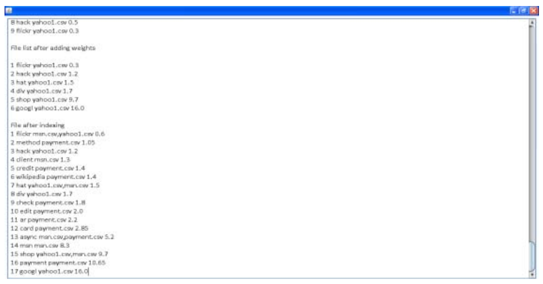


```
8 hack yahoo1.csv 0.5
9 flickr yahoo1.csv 0.3

File list after adding weights

1 flickr yahoo1.csv 0.3
2 hack yahoo1.csv 1.2
3 hat yahoo1.csv 1.5
4 div yahoo1.csv 1.7
5 shop yahoo1.csv 9.7
6 googl yahoo1.csv 16.0

File after indexing
1 flickr msn.csv,yahoo1.csv 0.6
2 method payment.csv 1.05
3 hack yahoo1.csv 1.2
4 client msn.csv 1.3
5 credit payment.csv 1.4
6 wikipedia payment.csv 1.4
7 hat yahoo1.csv,msn.csv 1.5
8 div yahoo1.csv 1.7
9 check payment.csv 1.8
10 edit payment.csv 2.0
11 ar payment.csv 2.2
12 card payment.csv 2.85
13 async msn.csv,payment.csv 5.2
14 msn msn.csv 8.3
15 shop yahoo1.csv,msn.csv 9.7
16 payment payment.csv 10.65
17 googl yahoo1.csv 16.0
```

Figure 6. Processing of file (Addition of weights)

In this Fig 6, it has been shown that the same keyword are added up with their weights and are presented once in the processed document. For example, earlier the keyword shop has occurred three times and after processing the third step it has been replaced with occurrence of shop once with weight 9.7 i.e. (1.7+1.3+6.7) by adding their corresponding weights.

d) The implementation of final step of processing the documents is to create the index. The implementation of the same is as shown in Fig 7.

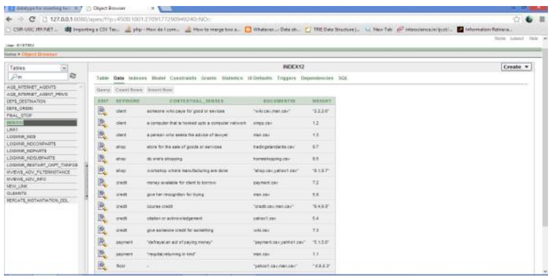


```
8 hack yahoo1.csv 0.5
9 flickr yahoo1.csv 0.3

File list after adding weights

1 flickr yahoo1.csv 0.3
2 hack yahoo1.csv 1.2
3 hat yahoo1.csv 1.5
4 div yahoo1.csv 1.7
5 shop yahoo1.csv 9.7
6 googl yahoo1.csv 16.0

File after indexing
1 flickr msn.csv,yahoo1.csv 0.6
2 method payment.csv 1.05
3 hack yahoo1.csv 1.2
4 client msn.csv 1.3
5 credit payment.csv 1.4
6 wikipedia payment.csv 1.4
7 hat yahoo1.csv,msn.csv 1.5
8 div yahoo1.csv 1.7
9 check payment.csv 1.8
10 edit payment.csv 2.0
11 ar payment.csv 2.2
12 card payment.csv 2.85
13 async msn.csv,payment.csv 5.2
14 msn msn.csv 8.3
15 shop yahoo1.csv,msn.csv 9.7
16 payment payment.csv 10.65
17 googl yahoo1.csv 16.0
```

Figure 7. Processing of documents (File after indexing)

The Fig 7. displays the various fields: S.No., keyword, list of documents in which they appear and the weights of the words in the respective documents. Here shop is occurred in two files i.e. yahoo1.csv and msn.csv with weight 9.7.

e) A data structure designed to store these processing information has been shown in Fig 8.



INDEX12

EDIT	KEYWORD	CONTEXTUAL_SENSES	DOCUMENTID	WEIGHT
	client	someone who pays for good or services	"wiki.csv,man.csv"	"2.2,2.6"
	client	a computer that is hooked upto a computer network	xmpp.csv	1.2
	client	a person who seeks the advice of lawyer	man.csv	1.1
	shop	store for the sale of goods or services	tradingstandards.csv	9.7
	shop	do one's shopping	homeshopping.csv	5.5
	shop	workshop where manufacturing are done	"shop.csv,yahoo1.csv"	"8.1,8.7"
	credit	money available for client to borrow	payment.csv	7.2
	credit	give her recognition for trying	man.csv	5.8
	credit	course credit	"credit.csv,man.csv"	"6.4,6.0"
	credit	citation or acknowledgement	yahoo1.csv	5.4
	credit	give someone credit for something	wiki.csv	7.3
	payment	"defrayal,an act of paying money"	"payment.csv,yahoo1.csv"	"5.1,5.6"
	payment	"requital,returning in kind"	man.csv	1.1
	floor	-	"yahoo1.csv,man.csv"	"0.8,8.3"

Figure 8. Storing the keywords with contextual senses in database

In the above Fig 8, various fields presented are: keywords, contextual senses, document Id and their corresponding weights. Here, the keyword represent word occurred in the document, contextual senses represent different meaning of the same keyword, document ID represent the occurrence of keywords in various documents and weight represent weight of the keyword in the corresponding document. For example, the sixth row in the Figure8 shows the entry of keyword 'shop' with their contextual sense 'workshop where manufacturing are done', the corresponding document IDs 'shop.csv' and 'yahoo1.csv' and with respective weights '8.1' and '8.7' is stored in the database.

V. COMPARISON BETWEEN CONTEXTUAL INDEXING ON BASIS OF QUERY AND ONTOLOGY[8]:-

Indexing Mechanism	Document Approach	Context Extraction	Ambiguity Consideration	Data Structure	System Performance
Ontology Based	Considers the terms with maximum frequency those occurs only in the title tag of HTML document.	Ontology repository is used to get various contexts.	No consideration of ambiguity if user is not aware of the context.	Simple Inverted Index.	Fast access to documents.
Proposed (i.e. contextual Sense based)	Considers the frequency of keywords in various tags such as Head, Title, Keyword, Body, Description, Link etc.	WordNet dictionary is used to extract the various senses.	The various contextual senses are passed to the user to select the specific context so ambiguity will be removed.	Trie type tree Structure with search -insert mode.	Search performance is faster than existing as searching is done in tree type Structure.

Table 1. Comparison between proposed and ontology based Contextual Indexing

In Contextual based Ontology when we search the colt horse specific results according to young male horse will be extracted. In our proposed mechanism when we search colt the different contextual senses are generated such as young male horse under the age of four and a kind of revolver, although user will select two times but the results will be more refined.

VI. CONCLUSION

In this paper, we have implemented an indexing mechanism to store the keyword present in the document with their contextual senses .It also focuses the importance of keywords in different HTML Tags. The mechanism removes the stop words, stems the keyword and after that creates the index. The data structure trie fasten the search for matched results from the Inverted Index. It also helps the user to process the user query with fast and more relevant results.

REFERENCES

[1] L. Huilin, K. Chunhua and W. Guangxing, "Efficiently Crawling Strategy for Focused Searching Engine", Advances in Web and Network Technologies and Information Management, Lecture Notes in Computer Science, 2007, Vol. 4537/2007, 25-36.

[2] Changshang Zhou, Wei Ding, NaYang, "Double Indexing Mechanism of Search Engine based on Campus Net", Proceedings 2006 IEEE Asia-Pacific Conference on Services Computing (APSCC'06).

[3] G.Salton and M.J.McGill , "An Introduction to Modern Information Retrieval" ,McGaw-Hill,1983.

[4] S. Chakrabarti, M. Berg, B. Dom,"Focused crawling: a new approach to topic-specific Web resource discovery", The International Journal of Computer and Telecommunications Networking, Volume 31 Issue 11-16, 1999.

[5] S. Mitra, M. Winslett, Windsor W. Hsu and K. Chen-Chuan, "Trustworthy keyword search for compliance storage", VLDB, 2008, J.17(2), pp 225-242.

[6] Rada Mihalcea and Dan Moldovan, "Semantic Indexing using WordNet senses", in proceedings of ACL workshop on IR and NLP, Hong Kong, 2000.

[7] X. Chen, X. Zhang, " HAWK: A Focused Crawler with Content and Link Analysis", IEEE International Conference on e-Business Engineering. pp- 677-680, 2008.

[8] Parul Gupta, Dr. A.K. Sharma, "Context based Indexing in Search Engines using Ontology", International Journal of Computer Applications(0975-8887), Vol.1-14,2010.

[9] [9]S Büttcher, CLA Clarke , "Indexing time vs. query time: trade-offs in dynamic information retrieval systems", Proceedings of the 14th ACM international in 2005.

[10] Zhiqiang Wang and Ruifan Li , ''An Index Design in Topic-focused Search Engine", Centre for Intelligent Science and Technology Beijing University of Posts and Telecommunications.

[11] S.Chakrabarti, B.Com, P.Raghvan, S.Rajagopalana,D.Gibson, and J.Kleinberg, "Automatic Resource compilation by analyzing hyperlink structure associate text," in Proc 7th World Wide Web Conference , Brisbane, Australia,1998.

[12] S.Brin and L.Page, '' The Anatomy of a Large-Scale Hypertextual Web Search Engine". In: Seventh International World-Wide Web Conference (WWW 1998), April 14-18, 1998, Brisbane, Australia.

[13] Elizabeth Shanthi and R. Nadarajan, "An Index Structure for Fast Query Retrieval in Object Oriented Data Bases Using Signature Weight Declustering", Information technology Journal, vol-8, issue-3, pp.275-283, 2009.

[14] M.F., Porter, "An algorithm for suffix stripping", Program, vol. 14, pp. 130-137. 1980.