Performance and Comparative Analysis of the Two Contrary Approaches for Detecting Near Duplicate Web Documents in Web Crawling

V.A.Narayana
Associate Professor
CMR College of Engg & Tech
Hyderabad, AP

P. Premchand
Professor
Univ College of Engg, OU
Hyderabad, AP

A. Govardhan
Professor
JNTU College of Engg
Hyderabad, AP

ABSTRACT

Recent years have witnessed the drastic development of World Wide Web (WWW). Information is being accessible at the finger tip anytime anywhere through the massive web repository. The performance and reliability of web engines thus face huge problems due to the presence of enormous amount of web data. The voluminous amount of web documents has resulted in problems for search engines leading to the fact that the search results are of less relevance to the user. In addition to this, the presence of duplicate and near-duplicate web documents has created an additional overhead for the search engines critically affecting their performance. The demand for integrating data from heterogeneous sources leads to the problem of near-duplicate web pages. The detection of near duplicate documents within a collection has recently become an area of great interest.

In this research, we have presented an efficient approach for the detection of near duplicate web pages in web crawling which uses keywords and the distance measure. Besides that, G.S. Manku et al.'s fingerprint based approach proposed in 2007 was considered as one of the "state-of-the-art" algorithms for finding near-duplicate web pages. Then we have implemented both the approaches and conducted an extensive comparative study between our similarity score based approach and G.S. Manku et al.'s fingerprint based approach. We have analyzed our results in terms of time complexity, space complexity, Memory usage and the confusion matrix parameters.

After taking into account the above mentioned performance factors for the two approaches, the comparison study clearly portrays our approach the better (less complex) of the two based on the factors considered.

General Terms
Data Mining, Web Mining

Keywords
Near Duplicate Documents, Similarity Score Measure, Confusion Matrix, Storage Space Complexity, Memory Usage Analysis, Computation Time Analysis.

1. INTRODUCTION

Near-duplicate documents can adversely affect the efficiency and effectiveness of search engines. Due to the pair-wise nature of the comparisons required for near-duplicate detection, this process is extremely costly in terms of the time and processing power it requires. Despite the ubiquitous presence of near-duplicate detection algorithms in commercial search engines, their application and impact in research environments is not fully explored. The implementation results of near-duplicate detection algorithms forces trade-offs between efficiency and effectiveness, entailing careful testing and measurement to ensure acceptable performance. This paper describes the experimental results and the performance analysis of the proposed methods by Narayana[19,20,21,22] with the existing approach proposed by Manku[9]. The results and discussion is carried out using the huge document collection of databases to clearly evaluate the performance of the approaches. In addition to this, the well-accepted evaluation metrics are employed here to directly compare the approach with the existing ones. In order to prove the effectiveness, we initially set up the experimental environment and collect the huge collection of documents which are more appropriate to conduct an experimental study, so that only the effectiveness can be evaluated in a fulfilled manner according to expert's view.

2. REVIEW OF RELATED RESEARCH

The discovery of near-duplicates benefits numerous applications including web search engines. Some advantages of near-duplicate detection in web search engines are, helps to perform focused crawling, increase the quality and diversity of query results, and identifies spam [6, 8,10]. Many web mining applications trust on the ability to accurately and efficiently identify near-duplicates. They include document clustering [4], finding replicated web collections [5], detecting plagiarism [12], community mining in a social network site [13], collaborative filtering [3] and discovering large dense graphs [11]. Removal of near-duplicates [1] saves network bandwidth, reduces storage costs and improves the quality of search indexes and it also reduces the load on the remote host that is serving such web pages. Documents that are exact replicas of each other (due to mirroring and plagiarism) can be easily identified by standard check summing techniques [2], but the most difficult part is to identify the near-duplicate documents [9].

Earlier the research on duplicate detection was performed generally in the areas of databases, digital libraries, and electronic publishing. Recently, duplicate detection has been studied for web search tasks, for example, to give more effective and proficient web crawling, document ranking, and document archiving. Numerous duplicate detection techniques have been proposed, that range from manually coded rules to applications of the latest machine learning techniques [7, 8, 16, 17]. Recently, few authors have proposed techniques for near-duplicates detection [9,14,15,16]. Their focus varies from providing high detection rates to minimizing the computational and storage resources. For large collections, some techniques are too pricey computationally to be deployed in their full capacity, whereas some algorithms are very efficient yet very brittle and sensitive to even small changes of the text.

A comparison of the two algorithms namely shingling algorithm [4] and random projection based algorithm [2] was performed by Monika Henzinger [8] on a very large scale set of 1.6B distinct web pages. The outputs showed that none of the algorithms works well for locating near-duplicate pairs on the same site, while both attain high accuracy for near-duplicate pairs on different sites. A combined algorithm has been presented by them which achieve precision 0.79 with 79% of the recall of the other algorithms. Chuan Xiao et al. [7] have presented exact similarity join algorithms with application to near-duplicate detection and a positional filtering principle. It exploits the ordering of tokens in a record and leads to upper bound estimates of similarity scores. They demonstrated the better-quality performance of their algorithms to the existing prefix filtering-based algorithms on several real datasets under a wide range of parameter settings.

Hui Yang et al. [15] have presented DURIAN (DUplicate Removal In lArge collectioN), an improved version of a prior near-duplicate detection algorithm. DURIAN uses a traditional bag-of-words document representation, document attributes ("metadata"), and document content structure to recognize form letters and their edited copies in public comment collections. The outputs show that statistical similarity measures and instance-level constrained clustering can be very helpful for efficiently identifying near-duplicates.

Andrei Z. Broder et al. [4] have created an efficient method to find out the syntactic similarity of files and have applied it to every document on the World Wide Web. Using this method, they have built a clustering of all the documents that are syntactically similar. Possible applications include a "Lost and Found" service, filtering the results of Web searches, updating widely distributed web-pages, and identifying violations of intellectual property rights.

Hui Yang et al. [18] have discovered the use of simple text clustering and retrieval algorithms for locating near-duplicate public comments. They have focused on automating the method of near-duplicate detection, particularly form letter detection, in this domain. They gave a clear near-duplicate definition and explored simple and efficient methods of using feature-based document retrieval and similarity-based clustering to discover near-duplicates. The technique is evaluated in experiments with a subset of a large public comment database collected for EPA rule. Donald Metzler et al. [17] have explored methods for measuring the intermediate kinds of similarity, focusing on the task of finding where a particular piece of information initiated. They mentioned a range of ideas to reuse detection at the sentence level, and a range of ideas for merging sentence-level evidence into document-level evidence. They considered both sentence-to-sentence and document-to-document comparison, and have integrated the algorithms into RECAP, a prototype information flow analysis tool.

Sergey Brin et al. [16] have created a system for registering documents and then detecting copies, either complete copies or partial copies. They illustrated algorithms for such detection, and metrics required for evaluating detection mechanisms constituting accuracy, efficiency and security. They also mentioned a prototype implementation of this service, COPS, and presented experimental results that suggest the service can really detect violations of interest. Ziv BarYossef et al. [14] have mentioned the issue of dust: Different URLs with Similar Text. They proposed original algorithm, DustBuster, for detecting dust; that is, for discovering rules that change a given URL to others that are likely to have similar content. DustBuster mines dust effectively from previous crawl logs or web server logs, without investigating page contents.

Gurmeet Singh Manku et al.[9] have made two research contributions in developing a near-duplicate detection system for a multi-billion page repository. Initially, they verified that Charikar's [2] fingerprinting technique is appropriate for this goal and then they proposed an algorithmic technique for finding existing f-bit fingerprints that differ from a given fingerprint in at most k bit-positions, for small k. This technique is valuable for both online queries (single fingerprints) and batch queries (multiple fingerprints). Jack G. Conrad et al. [6] have determined the extent and the types of duplication present in large textual collections. Their research is divided into three parts. At first they started with a study of the distribution of duplicate types in two broad-ranging news collections consisting of approximately 50 million documents. Then they check the utility of document signatures in addressing identical or nearly identical duplicate documents and their sensitivity to collection updates. Lastly, we have demonstrated a flexible method of characterizing and comparing documents in order to allow the identification of non-identical duplicates. This method has created promising results following an extensive evaluation using a production-based test collection formed by domain experts.

3. EXPERIMENTAL ENVIRONMENT AND SETUP

The proposed near duplicate document detection system is programmed using Java (jdk 1.6) and the backend used is MS Access. The experimentation has been carried out on a 2.9 GHz, i5 PC machine with 4 GB main memory running a 32-bit version of Windows XP.

3.1 Dataset Collection

Here, we have gathered our huge dataset with the aid of a computer program, Wget. Here, we have collected by giving some URLs of the news website like Times of India or India Times. So that, we can get more number of web documents with relevant information. Almost, we have collected 75K documents for our research analysis. With these documents, the proposed research methods get analyzed with the existing algorithm and hence prove the efficiency with the evaluation metrics.

3.2 Wget

Wget can optionally work like a web crawler by extracting resources linked from HTML pages and downloading them in sequence, repeating the process recursively until all the pages have been downloaded or a maximum recursion depth specified by the user has been reached. The downloaded pages are saved in a directory structure resembling that on the remote server. This "recursive download" enables partial or complete mirroring of web sites via HTTP. Links in downloaded HTML pages can be adjusted to point to locally downloaded material for offline viewing. Wget (or just Wget, formerly Geturl) is a computer program that retrieves content from web servers, and is part of the GNU Project. Its name is derived from World Wide Web and get. It supports downloading via HTTP, HTTPS, and FTP protocols. Its features include recursive download, conversion of links for offline viewing of local HTML, support for proxies, and much more.

3.3 Creation of Dataset & Testing

The dataset is created by means of having 75K crawled web pages. Within these, 30K pages are taken out and 25K pages in those 45K pages are transformed into near duplicate web pages with the aid of some intermediate steps. Out of which, 30K web pages are exact duplicates, 25K pages are transformed near duplicates and the remaining 20K pages are the newly involved web pages. Thus, we have a total of 75K web pages for our experimental analysis.

In the testing phase, the fingerprint values and the keywords of the collected exact duplicate web pages are extracted and stored in the database. Then, the remaining web pages are utilized to test the results in the case of finding the exact duplicate, near duplicate and non duplicate web pages.

4. PERFORMANCE EVALUATION METRICS

There are two main considerations when solving an NDD problem: efficiency and accuracy. Applications of NDD typically need to handle a very large collection of documents. A practical algorithm will need to determine whether a document is a duplicate of some other documents in the repository in real-time. As a result, efficiency has been the main focus of existing popular NDD approaches, where various techniques of generating short signatures using hash functions and pruning inverted index searching were invented. In contrast, the problem of how to improve NDD accuracy has received less attention. Thus, in this thesis, we have analyzed our results in terms of time complexity, space complexity, Memory usage and the confusion matrix parameters.

4.1 Space Complexity

The space complexity of a program (for a given input) is the number of elementary objects that this program needs to store during its execution. This number is computed with respect to the size n of the input data. The way in which the amount of storage space required by an algorithm varies with the size of the problem it is solving. Space complexity is normally expressed as an order of magnitude, e.g. O(N^2) means that if the size of the problem (N) doubles then four times as much working storage will be needed. Space complexity is the storage required for the execution of the algorithm. This includes all permanent and temporary storage required by the algorithm, including any necessary stack allocations for recursions.

4.2 Time Complexity

Intuitively, the amount of time an algorithm takes depends on how large is the input on which the algorithm must operate: Sorting large lists takes longer than sorting short lists; multiplying huge matrices takes longer than multiplying small ones. The dependence of the time needed to the size of the input is not necessarily linear: sorting twice the number of elements takes quite a bit more than just twice as much time; searching (using binary search) through a sorted list twice as long, takes a lot less than twice as much time. The time complexity function expresses that dependence. Note that an algorithm might take different amounts of time on inputs of the same size. We can define the size of an input in a general way as the number of bits required to store the input. This definition is general but it is sometimes inconvenient because it is too low-level. More usefully we define the size of the input in a way that is problem-dependent.

4.3 Memory Usage

The memory utilized by the current jobs present in the particular system. Memory is an important resource for your application so it's important to think about how your application will use memory and what might be the most efficient allocation approaches. Most applications do not need to do anything special; they can simply allocate objects or memory blocks as needed and not see any performance degradation. When you are creating an application that can be memory-intensive, it may be useful to monitor the current memory usage. This allows you to modify the behavior of the program as its RAM requirements increase and to predict out-of-memory exceptions.

4.4 Confusion Matrix

In the field of artificial intelligence, a confusion matrix is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class. The name stems from the fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another). A confusion matrix contains information about actual and predicted classifications done by a classification system. Performance of such systems is commonly evaluated using the data in the matrix. The following table shows the confusion matrix for a two class classifier. The entries in the confusion matrix have the following meaning in the context of our study:

- a is the number of correct predictions that an instance is negative,
- b is the number of incorrect predictions that an instance is positive,
- c is the number of incorrect of predictions that an instance negative, and
- d is the number of correct predictions that an instance is positive.

Confusion matrix		Predicted	
		Negative	Positive
Actual	Negative	a	b
	Positive	c	d

4.4.1 True Positive

True positive is defined as the ability of a test to identify the correct one with a condition. A test with a high true-positive rate will nearly always be positive for the object who have the condition (the test has a low rate of false-negative results). The true-positive rate is also known as sensitivity. It indicates the likelihood that the object with a positive test result would actually have the condition for which the test is used. The higher the value of the positive predictive value (for example, 90 percent would be considered a high value), the more useful the test is for predicting that the object has the condition.

4.4.2 False Positive

The false positive rate is the proportion of absent events that yield positive test outcomes, i.e., the conditional probability of a positive test result given an absent event. The false positive rate is equal to the significance level. The specificity of the test is equal to 1 minus the false positive rate. In statistical hypothesis testing, this fraction is given the Greek letter α, and 1 − α is defined as the specificity of the test. Increasing the specificity of the test lowers the probability of type I errors, but raises the probability of type II errors (false negatives that reject the alternative hypothesis when it is true).

4.4.3 True Negative

True negative is defined as the ability of a test to identify correct one without the condition. A test with a high true-negative rate will rarely be wrong about who does not have the condition (the test has a low rate of false-positive results). The true-negative rate is also known as specificity. It indicates the likelihood that objects with a negative test result would not have a condition. The higher the value of the negative predictive value (for example, 99 percent would usually be considered a high value), the more useful the test is for predicting that the object do not have the condition.

4.4.4 False Negative

The false negative rate is the proportion of events that are being tested for which yield negative test outcomes with the test, i.e., the conditional probability of a negative test result given that the event being looked for has taken place. In statistical hypothesis testing, this fraction is given the letter β. The "power" (or the "sensitivity") of the test is equal to 1− β.

4.5 Accuracy

The accuracy of a measurement system is the degree of closeness of measurements of a quantity to that quantity's actual (true) value. That is, the accuracy is the proportion of true results (both true positives and true negatives) in the population. It is a parameter of the test.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

5. PERFORMANCE & COMPARATIVE ANALYSIS RESULTS

5.1 Effective Analysis

The effectiveness of our work is compared with the Manku *et al.'s* work in terms of various evaluation metrics. The experimentation is carried out using 75,000 web pages, which are divided into three categories. In the first set, we have considered the 30,000 web pages and 25,000 pages are the near duplicate of these pages by adding some more keywords in these web pages. In the third set, non-duplicate web pages. In order to construct the database, first set of web pages are taken and it has been preprocessed with a set of operations so that top-10 frequent keywords are extracted. These keywords and their frequency are stored in the database for our work. But, in Manku *et al*.'s work, fingerprint of the web documents and it permutation are stored in the database.

In effectiveness analysis, all the web documents are preprocessed and matched with pages that are already in the database. The first and second set of document should be near duplicate and the third set of documents should be non-duplicate documents. At the same way, we have matched all web documents with the database to find whether the input documents are duplicate or no-duplicate. Then, the evaluation metrics is computed based on the definition given in the previous chapter. The same experiments have been conducted for the various matching threshold. In manku work, the bit difference threshold (k) is fixed like, 2, 3 and 4 and in our work, the actual threshold fixed by our experiments is 19.5043. For fixing the value of k, the SSM values [0,5), [5,10), [10,15) and [15,19.5] is fixed from 1,2,3 and 4 respectively. For the various threshold values, the evaluation metrics are found out and the graph is plotted.

In fig 1, True positive of both the works are computed and plotted as graph. From the graph, we can analyze that our work has obtained the TP of 42,900 which is compared with existing technique that has achieved 42500. This concludes that the number of correctly identified pages as duplicate is high compared with the existing technique. Also, if the matching is varied to the higher values, the behavior of both the algorithms is similar. Both the algorithms improve their performance in identifying the exact duplicate documents.

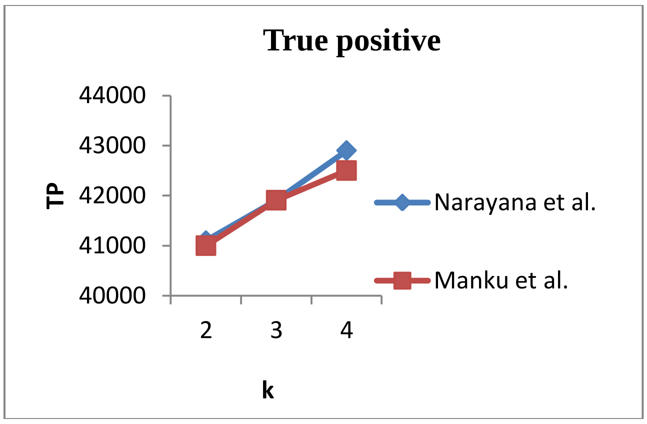

Fig.1. Graph plotted for True positive

Fig 2 is plotted for the false positive of both the algorithms for various matching threshold. When comparing the performance of both the algorithms in terms of FP, the proposed algorithm is better compared with the previous algorithm. Our algorithm obtained 10,443 as FP for the threshold of two, but the existing algorithm achieved 10583. This ensures that non-duplicate documents incorrectly identified as duplicate documents have been reduced for our algorithm compared with the existing technique.

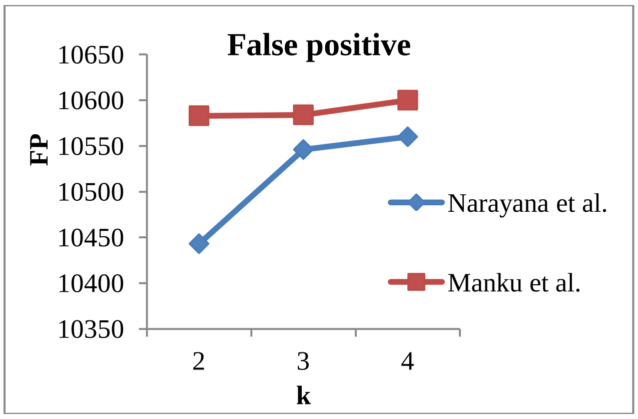

Fig.2. Graph plotted for false positive

Fig 3 is plotted for the true negative of both the algorithms for various matching threshold. When comparing the performance of both the algorithms in terms of TN, the proposed algorithm is better compared with the previous algorithm. Our algorithm obtained 9490 as TN for the threshold of four, but the existing algorithm achieved 9,300. This ensures that non-duplicate documents correctly identified as non-duplicate documents have been improved for our algorithm compared with the existing technique.

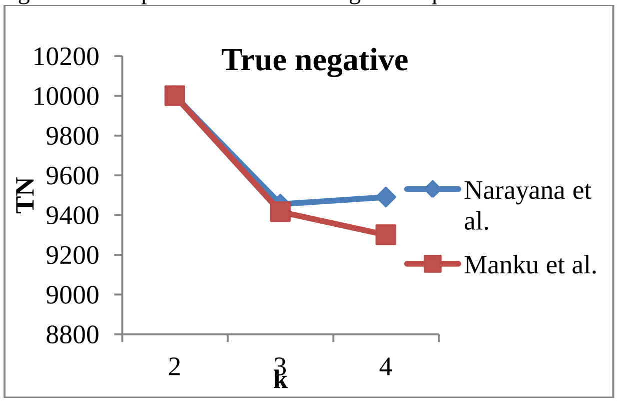

Fig.3. Graph plotted for True Negative

Another evaluation metric utilized in our work is FN that considers the non duplicate documents incorrectly identified as duplicate documents. Based on that, for various matching thresholds, FN value is computed for both the algorithms and it is plotted as graph shown in fig 4. After analyzing this graph, both algorithms achieved the similar value for the first two thresholds. But, for the k value of four, our algorithm has obtained 12050 while the existing algorithm achieved 12600.

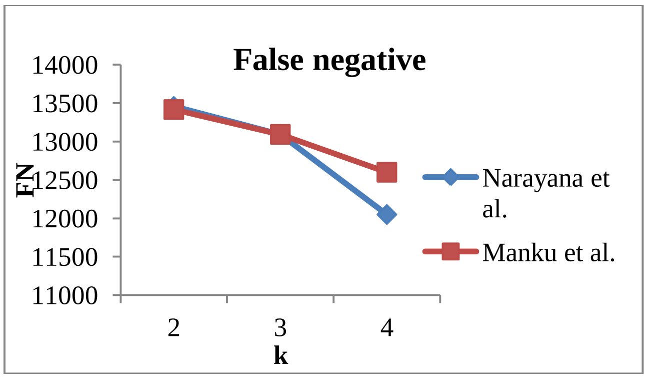

Fig.4. Graph plotted for False Negative

Fig 5 shows the sensitivity of our algorithm with existing technique. From the graph, the performance is increasing when the matching threshold is increased. Here, the performance is similar for the thresholds, 2 and 3. But, for the threshold of four, our algorithm has performed even better of previous algorithm. Our algorithm achieved an improvement of nearly 1% in near duplicate detection process compared with existing technique.

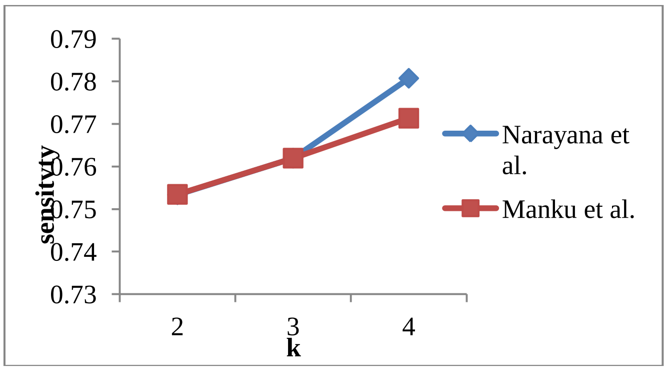

Fig.5. Graph plotted for sensitivity

Fig 6 shows the specificity of our algorithm with existing technique. From the graph, the performance is inreasing when the matching threshold is increased. Here, the performance is different for all the thresholds, 2, 3 and 4. For the threshold of four, our algorithm achieved an improvement of 1% in near duplicate detection process compared with existing technique.

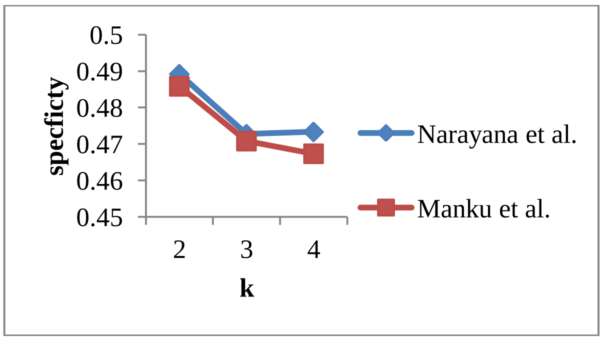

Fig.6. Graph plotted for specificity

Fig 7 shows the accuracy plot of both the algorithms. Here, the performance is better for all the thresholds even with the existing technique. Here, for the smaller thresholds, the performance is slightly improved but, for the large threshold, the performance of our algorithm achieved 1% improvement with the existing technique. And also, this chart suggests that the better threshold for this experiment is k=4 for both the algorithms. Figs 8 and 9 shows the FP and FN rates. From all the graphs, the performance is better for our algorithm and threshold suggested for near duplicate detection process is four.

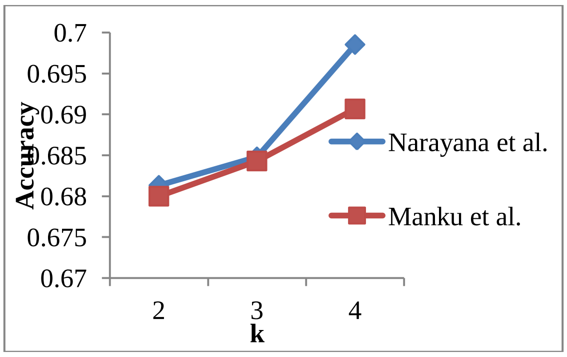

Fig.7. Graph plotted for accuracy

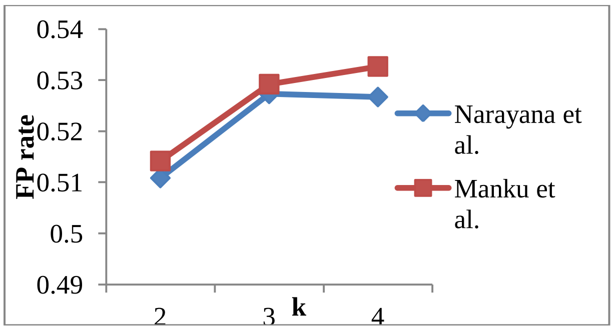

Fig.8. Graph plotted for FP rate

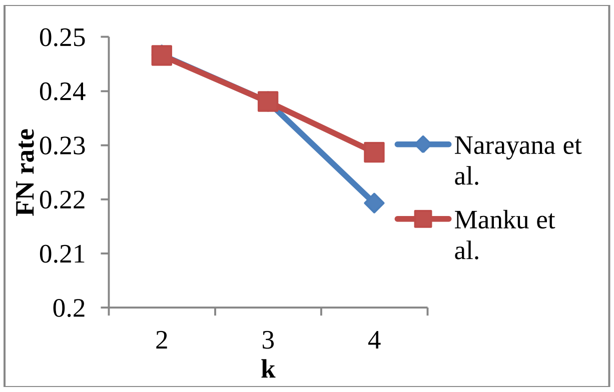

Fig.9. Graph plotted for FN rate

5.2 Scalability Analysis

This section describes the scalability analysis of both the methods. The scalability analysis is needed to find how the algorithms are behaved when the size of the data is continuously changed to higher value. The scalability of both algorithms is compared with computation time needed to database construction, memory space needed to store the fingerprint/keywords and the space complexity.

5.2.1 Computation Time Analysis

Time is one of the most significant parameter that can effectively determine the complexity of the algorithm. In simple terms, it can be defined as the runtime of the algorithm. Time can be employed as an effective measure in investigative studies to determine the better of the n algorithms considered. In our consideration,

- Manku et al.'s fingerprint based approach, the computation time is considered as the time taken to perform exact bitwise match between the query fingerprint and the stored permuted fingerprints. Roughly, for k=3, the query fingerprint has to be matched with nearly 2240 permutations of a single fingerprint. The condition becomes more intricate with *n* fingerprints available in the repository.
- Similarly, in correspondence to our proposed similarity score based approach; the computation time is considered as the time taken to compute the consolidated similarity score between the query and the reference web page.

Fig 10 is plotted for computation time needed for inserting the keywords/fingerprints of all the web pages to the database. Here, whenever the number of pages is increased, the computation time is also increased in the same way for both the algorithms. But, the insertion time of our algorithm is comparatively high compared with the existing algorithm when the number of pages inserted into the database is high. While analyzing the graph, we found that our algorithm took much computation time to insert a set of keywords for the particular web pages. For 5k web pages, our algorithm took 15000 sec but, existing algorithm took only 10500 sec. Due to finding of frequency, our algorithm took slightly high computation time compared with existing technique.

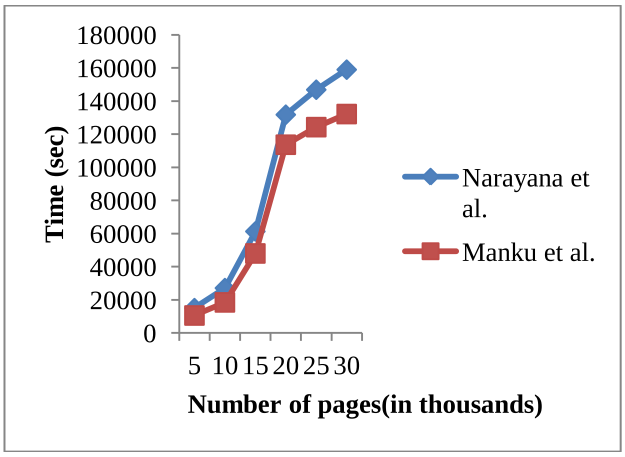

Fig.10. Time complexity Analysis Graph

5.2.2 Memory Usage Analysis

Memory is an important resource for an application, so it's important to think about how an application will use memory and what might be the most efficient allocation approaches.

Most applications do not need to do anything special; they can simply allocate objects or memory blocks as needed and not see any performance degradation. When we are creating an application that can be memory-intensive, it may be useful to monitor the current memory usage. This allows us to modify the behavior of the program as its RAM requirements increase and to predict out-of-memory exceptions.

Fig 11 is plotted for the memory needed to execute both algorithms for various set of web pages. Here, the performance of both the algorithms is almost similar for various set of pages. The only difference we saw from graph is that our algorithm achieved significantly higher memory for 10k pages.

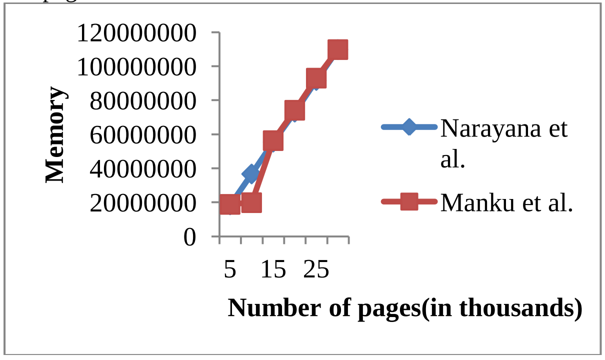

Fig.11. Memory Analysis Graph

5.2.3 Storage Space Complexity Analysis

Another most well-known complexity measure used for delineating an algorithm's complexity is the storage space incurred. With ideal detection accuracy achieved by both the approaches considered, we take up storage space (memory overhead) as another parameter for comparative analysis. Generally, prior to near-duplicate detection, every repository is likely to have hundreds of reference web pages (signatures) that are to be checked with the query web page. So, the term 'storage space' describes the memory space incurred to store those signatures or keywords of all those reference web pages, say *n*.

Here, we have considered 25k web pages and its corresponding information is stored in the database for duplicate detection process. For our algorithm, the top-10 keywords are stored in the database but the existing technique stored fingerprint and its possible permutation. Fig 12 shows the performance of the scalability issue in database construction. Whenever the web pages are increased, the space complexity is also increased. But, the percentage of increasing is less for our algorithm compared with existing technique. From the graph, we can assure that our algorithm is scalable compared with the existing technique. After adding 30k web pages, the space needed for our algorithm is 1*exp10. But, the existing technique is needed 2.6*exp10 which is more than double of our algorithm.

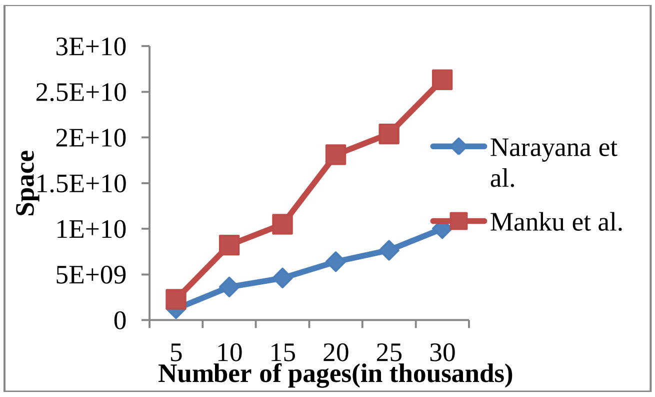

Fig.12. Space Complexity Analysis Graph

6. CONCLUSION

The experimentation has been carried out using the 75k web pages and the effectiveness and scalability are analyzed for our algorithm and the existing technique. From the effectiveness analysis, our algorithm has achieved better accuracy compared with the existing technique. Additionally, the scalability of our algorithm is achieved by our algorithm in terms of space needed to store the extracted web information.

7. REFERENCES

[1] A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan, "Searching the web", ACM Transactions on Internet Technology, vol. 1, no. 1: pp. 2-43, 2001.

[2] M. Charikar. "Similarity estimation techniques from rounding algorithms". In Proceedings of the 34th Annual Symposium on Theory of Computing (STOC 2002), pp: 380-388, 2002.

[3] R. J. Bayardo, Y. Ma and R. Srikant, "Scaling up all pairs similarity search". In Proceedings of the 16th international conference on World Wide Web, pp. 131 – 140, Banff, Alberta, Canada, 2007.

[4] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. "Syntactic clustering of the web", Computer Networks and ISDN Systems, vol. 29, pp. 1157–1166, 1997.

[5] J. Cho, N. Shivakumar, and H. Garcia-Molina. "Finding replicated web collections". ACM SIGMOD Record, vol. 29, no. 2, pp. 355 – 366, June 2000.

[6] J. G. Conrad, X. S. Guo, and C. P. Schriber. "Online duplicate document detection: signature reliability in a dynamic retrieval environment", in Proceedings of the twelfth international conference on Information and knowledge management, pp. 443 - 452 New Orleans, LA, USA, 2003.

[7] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu , "Efficient Similarity Joins for Near-Duplicate Detection", in Proceeding of the 17th international conference on World Wide Web, pp.131-140, 2008.

[8] M. Henzinger, "Finding near-duplicate web pages: a large-scale evaluation of algorithms", in Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 284-291, 2006.

[9] Gurmeet Singh Manku, Arvind Jain, Anish Das Sarma, "Detecting near-duplicates for web crawling", in Proceedings of the 16th international conference on World Wide Web, pp: 141 - 150, 2007.

[10] D. Fetterly, M. Manasse, and M. Najork. "On the evolution of clusters of near-duplicate web pages". In Proceedings of the First Conference on Latin American Web Congress, 2003.

[11] D. Gibson, R. Kumar, and A. Tomkins. "Discovering large dense subgraphs in massive graphs", In Proceedings of the 31st international conference on Very large data bases, pp. 721 – 732, Trondheim, Norway, 2005.

[12] T. C. Hoad and J. Zobel. "Methods for identifying versioned and plagiarized documents", Journal of the American Society for Information Science and Technology, vol. 54, no.3, pp.203–215, 2003.

[13] E. Spertus, M. Sahami, and O. Buyukkokten. "Evaluating similarity measures: a large-scale study in the orkut social network", in proceedings of International Conference on Knowledge Discovery and Data Mining, pp. 678 – 684, Chicago, Illinois, USA, 2005.

[14] Ziv Bar-Yossef, Idit Keidar,Uri Schonfeld, "Do not crawl in the dust: different urls with similar text," in Proceedings of the 16th international conference on World Wide Web, pp: 111 - 120, 2007.

[15] Hui Yang, Jamie Callan, Stuart Shulman, "Next steps in near-duplicate detection for eRulemaking," Proceedings of the 2006 international conference on Digital government research, vol. 151, pp: 239 - 248, 2006.

[16] S. Brin, J. Davis, and H. Garcia-Molina, "Copy detection mechanisms for digital documents", In Proceedings of the Special Interest Group on Management of Data (SIGMOD 1995), pp. 398–409. ACM Press, May 1995.

[17] D. Metzler, Y. Bernstein and W. Bruce Croft. "Similarity Measures for Tracking Information Flow", in Proceedings of the fourteenth international conference on Information and knowledge management, Bremen, Germany, 2005,

[18] H. Yang and J. Callan, "Near-duplicate detection for eRulemaking", in Proceedings of the 2005 international conference on Digital government research, pp: 78 - 86, 2005.

[19] V.A. Narayana, P. Premchand and A. Govardhan, "Effective Detection of Near-Duplicate Web Documents in Web Crawling", International Journal of Computational Intelligence Research, vol.5, no.1 ,pp. 83–96, 2009.

[20] V.A.Narayana, P.Premchand and A.Govardhan, (2010) "Fixing the Threshold for Effective Detection of Near Duplicate Web Documents in Web Crawling". Proceedings of 6th International Conference on Advanced Data Mining and Applications, Chongqing University, China Published in LNCS of SPRINGER in volume 6440/2010 pp 169-180.

[21] V.A. Narayana, P. Premchand and A. Govardhan, "Near-Duplicate Web Page Detection: A Comparative Study of Two Contrary Approaches" Paper published in proceedings of 6th International Conference on Computer Sciences and Convergence Information Technology, Jeju Island, Korea from 29 Nov - 01 Dec 2011. Indexed in IEEE XPLORE. pp 769-776

[22] V.A. Narayana, P. Premchand and A. Govardhan, "To Create A Confusion Matrix in Respect of Threshold Being Fixed for Effective Detection of Near Duplicate Web Documents in Web Crawling" Paper published in proceedings of 6th International Conference on Computer Sciences and Convergence Information Technology, Jeju Island, Korea from 29 Nov - 01 Dec 2011. Indexed in IEEE XPLORE. pp 763-768