Algorithm of the Longest Commonly Consecutive Word for Plagiarism Detection in Text Based Document

Agung Sediyono

Informatics Engineering Department
Universitas Trisakti, Jakarta, Indonesia
Tel:+62215663232 ext 178,
Fax:+6221567001
E-mail: agung@trisakti.ac.id

Ku Ruhana Ku-Mahamud

College of Arts and Sciences
Universiti Utara Malaysia, 06010 Sintok, Kedah
Tel : 04-9284701, Fax : 04-9284753
E-mail : ruhana@uum.edu.my

Abstract

Plagiarism is a form of academic misconduct which has increased with the easy access to obtain information through electronic documents and the Internet. The problem of finding document plagiarism in full text document can be viewed as a problem of finding the longest common parts of strings. Moreover, the detection system has to be capable to determine and visualize not only the common parts but also the location of the common parts in both the source and the observed document. Unlike previous research, this paper proposes a numerical based comparison algorithm that is comparable in the computation time without loosing the word order of common parts. Based on the experiment, the proposed algorithm outperforms the suffix tree in the length of observed paragraph below one hundred words.

1. Introduction

Based on a survey in Pakistan, Canada, UK, and India, it was found that 44.6 % of students used cut and paste style plagiarism from the Internet [1]. Besides creating a sense of awareness to academics and the public, another method to prevent plagiarism is using a detection tool. There are the detection tool of plagiarism and each tool uses different method in finding the part of plagiarism, but most of them use statistical metrics of similarity to determine whether plagiarism has been done or not. Moreover, most of the plagiarism detection are dedicated to detect the text based document, for instants CopyCatch, Eve2, WordsCheck, Glatt, and the famous one TurnItIn [2]. The scope of similarity could be on the scope of documents, paragraphs, or sentences of the observed and source document. To determine the level of similarity, the detection tool of plagiarism uses a percentage of common parts syntactically [3,4,5,6,7,8] or semantically [9,10,11,15].

The aim of the plagiarism detection tool is to help plagiarism assessor in judging plagiarism easily. Therefore, besides determining the suspected document in the first step, the detection system has also to be capable to determine and visualize not only the common parts but also the location of the common parts in both the source and the observed document. In syntactic approach based on text based comparison the common part is found by comparing character by character [6,7]. Therefore, this approach can fulfill the aim of creating the detection tool. The problem of this approach is the computation time. Implementation of the suffix tree algorithm can reduce the time complexity to O(m); however, the suffix tree algorithm looses not only the space complexity [12] but also the location of the common parts in the source document. Meanwhile, in the numerical based approach [3,4,5,8], the common part is found by comparing chunk by chunk of string that has been hashed. Even though, the numerical based approach outperforms in terms of computation time, it has a problem in determining a suitable length of chunk, and determining and locating a longest common part that is required in detection of plagiarism. The problem in determining the location of the common parts is also faced in semantic approach.

Unlike previous research that have problem in determining the location of common parts, this paper proposes a numerical based comparison that can determine not only the common parts but also the location of the common part in both source document and observed document. In this approach, the text of the source document is broken down into the paragraphs, and then each paragraph is broken down into all possible consecutive word. Every consecutive word and its paragraph identifier are saved into the

978-1-4244-2917-2/08/$25.00 ©2008 IEEE

special bloom filter that is designed in [13]. The observed document is treated as the source document, and every consecutive word of the paragraph of observed document is checked into the bloom filter. If it is in the bloom filter, the common part will be found. Based on the experiment, this research outperforms the suffix tree algorithm for length of paragraph below one hundred words in terms of the computation time. Meanwhile the precision of detection is loose because of the false positive of the bloom filter.

The reminder of this paper is organized as follows. Section 2 reviews the related work and section 3 describes the proposed algorithm. Meanwhile, the experimental design is discussed in section 4. The result and analysis is discussed in section 5. Finally, the conclusion and future work are described in section 6.

2. Related Work

There are detection tools of plagiarism. The most of them are dedicated to detect plagiarism in the text based document environment, such as CopyCatch, Eve2, WordsCheck, Glatt, and the famous one TurnItIn [2]. These tools hide the method used in finding similarly common part. Therefore, this paper explores other references that focus on algorithm in finding common parts either in scope of document, paragraphs, or sentences. Based on the previous research, the problem in finding the common parts can be solved using two approaches: syntactically [3,4,5,6,7,8] or semantically [9,10,11,15].

In the syntactic approach, the document is assumed as a composition of ordered chapters, paragraphs, sentences, words, or characters. In this approach, the comparison is done by comparing a document either based on chapter by chapter, paragraph by paragraph, sentence by sentence, word by word, character by character, or part by part in general according to the order. From this point of view, the problem in comparing parts of a document can be viewed as the repetition process of string matching where the pattern is part of the observed document and the text is the source document. Therefore, the syntactic approach can be viewed from the string matching algorithms. Even though the string matching algorithm can be applied to solve this problem it is not suitable in terms of computation time in long texts like in a large size document [12]. To improve this weakness, some researchers try to apply a filter in the first stage and then use text based comparison in finding common parts in the second stage [6,7,8]. Meanwhile, the others use numerical based comparison by converting text into a numeric code in order to speed up the comparison time [3,4,5].

Syntactic approach named *sif* [3] proposes to break the file into equal length of substrings. In this approach, the file is broken up into different combinations which are 50-byte substrings. Then, it converts all 50-byte substrings to fingerprints and compares all fingerprints in one file to the other. If the percentage of the common fingerprint is over the threshold, two files can be assumed similar. Even though the *sif* outperforms the text mode being compared, it loses the syntactic order because the different combination of 50-byte substrings is not the ordering of strings. Meanwhile, instead of breaking the document into the all possible 50-byte substrings, Brin [4] breaks the document into chunks. The important thing is there is a trade off between the accuracy of detection and the size of chunk. If the size of chunk is large, the accuracy of detection is decreased. For example, if the chunk size is a sentence, any word insertion or deletion only a meaningless word like a stop word can change the result of decision. However, if the size of the chunk is small like a word for example, the time of comparison increases and the system deteriorates. Moreover, in [5] the chunk size in a word. Instead of saving all words, it just saves the distinct chunks into an inverted index and the position where the chunks occurs is recorded. This system can overcome the accuracy problem in the [4]'s system. However, the size of inverted index still cannot fit the memory size. Therefore, the system performance is slowed down as long as there is an increase in the inverted index size because of the growth of the source documents. Most importantly though, it has been shown by the experiment that the word chunk has good accuracy in practice; even though, it loses sequencing information between words. A syntactic approach using numerical based comparison has a weakness in determining the location of the common parts. Therefore, other researcher [5] proposes a syntactic approach using text based comparison which is a suffix tree algorithm. In this approach, the observed document is indexed using suffix tree that is implemented in the suffix array data structure. By indexing the observed document, the suffix array can fit into a memory. The candidate documents from the first step are compared to the suffix array to determine the common part

Unlike the syntactic approach that is concerned with word by word, the semantic approach is only concerned with words that have a meaning or words that can represent the concept of a document. On the other hand, this approach tries to use filtering to remove the meaningless words or parts of document that does not contribute to the concept of document. By reducing the number of parts having to be compared, researchers hope to increase the speed of comparison. To find the concept words, researcher looks at the

Latent Semantic Index (LSI) comparing based on terms of rareness. The LSI includes only a part of the terms that can figure the concept of the document. In this way, the computation performance can be increased. Moreover, instead of using LSI to improve the speed of computation, other researcher [9] proposes I-Match using TF.IDF as a concept content of document and use a certain number of top rank terms according to the position of terms in a document and covert it into a hash code like fingerprint. The weakness of this method is the stability of the fingerprint because of the growing size of the corpus [14]. The other detection tool called SNITCH uses the Google Search API in comparing the observed document and the online source document. The observed document is plagiarized if the percentage of plagiarized word to the overall word count for the observed document is higher than the threshold. The weakness of this system is it can not determine the location of the plagiarism, so that the assessor has to check manual to determine the location where the plagiarism has been written [16]. Meanwhile, White and Joy in [11] propose the semantic in a sentence scope. The algorithm is very simple it just collects the semantic terms composing each sentence into a set of semantic sentences. The plagiarism detection is conducted by comparing pair wise of sentences in a text base comparison. The performance of this method does not outperform the previous methods. However, this method is reluctant to the insertion/deletion case and can recognize the author's style in writing.

3. Proposed Algorithm

3.1. Basic Concept

In this paper, the text of the source document is broken down into paragraphs. Each paragraph is labeled by document and paragraph identifier, DocId:ParaId as a location identifier of the paragraph. The text in each paragraph is chopped to be all possible consecutive words. The smallest size of the consecutive word is a single word. This smallest size is also used by [5]. From all consecutive words, it can be figured a triangle of consecutive word for each paragraph. For instance, for the paragraph having m words, there is $P=\{P_1, P_2, \ldots, P_m\}$ as a base members at the lowest level of the triangle. A member except the base members can be derived from the base members. The triangle is built level by level and each level is related to the longest of the concatenated of the base members. For instance, if the level is 2, the members are P_1P_2, P_2P_3, …, $P_{m-1}P_m$. Then, if the level is 3, the members are $P_1P_2P_3$, $P_2P_3P_4$, …, $P_{m-2}P_{m-1}P_m$. Therefore, for level m, a member is equal to P itself.

The text of the observed document is also broken down into paragraph, and then for each paragraph is built a triangle of consecutive word like in the source paragraph. In finding the commonly consecutive words (CCWs), the nodes in the triangle of the observed paragraph are compared to the nodes in the triangle of the source paragraph. This comparison is conducted paragraph by paragraph. By knowing the position of the node in the triangle and also the DocId:ParaId, the location of the CCW can be determined precisely. The longest commonly consecutive word (LCCW) can be found from the longest of the CCWs. Based on the facts that the nodes in the triangle are naturally ordered either in diagonal or vertical direction, the searching process in finding the node that represent a CCW can be conducted by using the binary search method. This binary search is applied either in diagonal search or vertical search. The diagonal search is applied if the start node is in the source. Meanwhile, the vertical search is applied if the diagonal search found the CCW except if the CCW is the base node. By using this approach, the sequential check node by node can be avoided. The illustration of this method can be seen in Figure 1 for *m* equal to five.

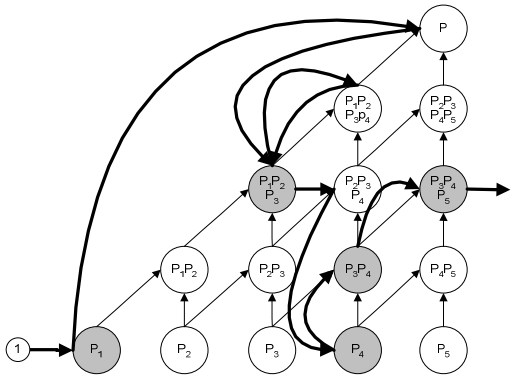

Figure. 1. Binary search in the case of multiple CCW: $P_1P_2P_3$ and $P_3P_4P_5$ for the 5 words paragraph

3.2. Implementation

The implementation tries to map the basic idea discussed in Section 3.1 to the real design so the basic idea can be implemented in the computer system. In this section the preparation of the source and the observed paragraph, and the search phase are described.

Source Paragraphs Preparation

The time and space complexity in building for both the source and the observed paragraph are ½ m(m+1),

where the m is the number of words in the source and observed paragraph. The quadratic characteristic of the time complexity in building the source triangle can be ignored because it can be conducted in the off-line processing. However, the space complexity can be a big problem. Therefore, this paper proposes to use a bloom filter in order to shrink the size of storage of the source paragraphs. The bloom filter has weaknesses such as the false positive and indifference of the location where the key comes from. The false positive can be minimized by choosing an appropriate length of the bloom filter. Meanwhile, the indifference of the location is overcome by implementing special bloom filter that is designed in [13] (see Figure 2). In this bloom filter, the key and its location are hashed separately and then modulo by *l* where l is the length of the bit vector representing the bloom filter. Finally, the bit position of the key and its location are combined to be a final bit position of the bit vector. By this approach, the location identifier (DocId:ParaId) can be processed offline since the location identifier is static.

All nodes of the paragraph triangles are entered into the bit vector of the bloom filter using mechanism that is figured in Figure 2. If the source is a huge document collection that has thousands or even millions of paragraphs, the size of the bit vector is difficult to be implemented into the memory. To overcome this problem, a single bit vector of the bloom filter is broken into multiple bit vectors that have same length. Therefore, there are a set of bit vector of the bloom filter $B=\{B_1,B_2, \ldots, B_b\}$ where *b* is number of the bit vector of the bloom filter and a set of bit position of the location identifier $L =\{L_1,L_2, \ldots,L_b\}$ where $L_i =\{L_{i1}, L_{i2}, ..,L_{ip})$ where *p* is number of paragraphs in B_i. Based on the fact that the hash function take the longest computation time in the bloom filter structure, this paper proposes to hash and modulo only the base member of the triangle and then rehash the bit position of the base member to be bit position of the upper nodes. In [3] the bit position in the upper level is calculated by using simple polynomial hash function like ($\sum P_i \times p^i$) modulo ℓ where p is a prime number, ℓ is upper bounded hash code and P is the bit position in the first level. Meanwhile, this paper uses a simple arithmetic such as ($\sum i\ P_i$) modulo ℓ for m-lv-1 ≤ i ≤ m where *lv* is the level of the node and m is the m^{th} node of the first level node.

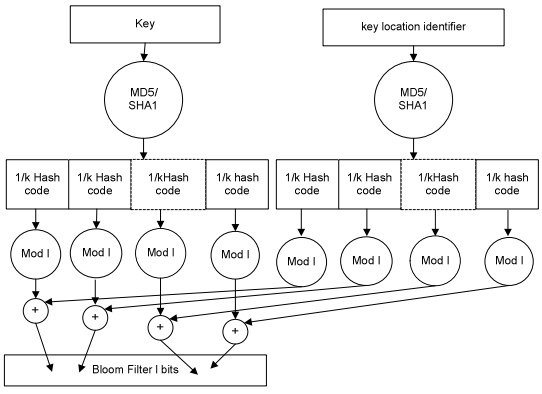

Figure. 2 The bloom filter that can differentiate the location of the key

The node on *lv*= i and *m*= j can be calculated as follows:

$$Pi, j = \sum_{j=1}^{j=i} (j) P_{1,j} \quad (1)$$

If equation 1 is implemented to the triangle of CCWs, it can be proved that the node on *lv*=i and *m*= j can be calculated by equation 2

$$Pi,j = P_{i-1,j-1} + i\ P_{1j} \quad (2)$$

Equation 2 shows that the upper node can be calculated based on the previous calculation of the node below it.

Observed Paragraph Preparation and Search Phase

The observed paragraphs are treated as in the source preparation. The hash and modulo function are used only for the base member. Since the binary search is not visits all the nodes, the bit positions of the upper node are calculated only when the node is visited by binary search. The triangle of the observed paragraph is saved into an array where each element of the array contains the bit position of the bit vector of the bloom filter (see Figure 3). Based on this data representation, finding the nearest lower node is a simple task that is by decreasing the index of array one by one. For

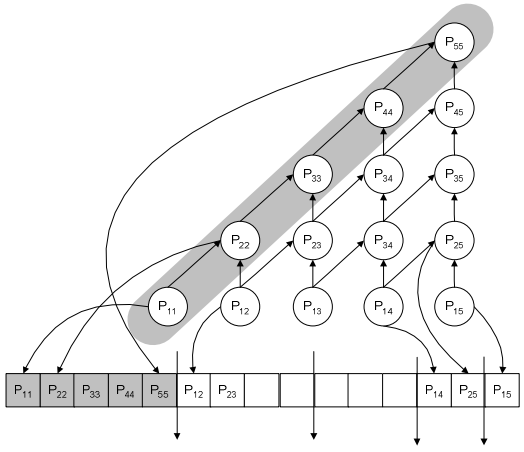

Figure. 3. Array presentation of the observed paragraph

instance, if node P_{44} is to be calculated, the first step is to find the nearest lower node that has been calculated by checking whether P_{33} has been calculated or not. If P_{33} is not calculated yet, it has to go down to node P_{22} and so on.
After observed paragraph has been prepared, the LCCWs can be found by using following algorithm:

```
For all paragraph in the observed Document do
  Prepare the triangle of observed paragraph Pk
  For all member of B do
     Load Bi , Li
     For all member of Li do
       --Search using binary search to find the LCCW
        LCCW(Pk,Bi,Lij)

LCCW(P, B, L)
m ← 1 'm is number of words in paragraph'
while m < upper bound of P
   do diagonal search to find CCW
   if CCW is found
      increase m by 1
      do vertical search to find other CCW
      if CCW is not found
          increase m by 1
   else
       increase m by 1
```

4. Experiment Design

To determine whether the LCCW algorithm outperform or not, the LCCW algorithm will be compared to suffix tree algorithm. The suffix tree algorithm is chosen because of its capability to determine a longest common part and its location as the LCCW does. Generally, the evaluation of algorithm is conducted based on the performance of algorithm which are the precision, the time and space complexity metrics. In here, only the computation time and precision will be used.

The precision of the LCCW algorithm is not comparable to the suffix tree because the suffix tree is exact matching, while the LCCW algorithm is approximate matching because of false positive rate of the bloom filter. However, the precision of the LCCW algorithm is explored to know its characteristic.

The computation time measurement is divided into four categories such as preparation time of pattern, loading time of resource index, loading time of resource, and matching time. The preparation time is the time beginning from loading time of observed paragraph into memory until converting to bit position of the bloom filters. The loading time of resource index is a time beginning from the loading set of paragraphs identity into memory until constructing it to be a linked list data structure. The loading time of resource is a cumulative time loading of source document that is represented in several fixed length of binary data of the bloom filters into the computer's memory. Finally, the matching time is a cumulative time of matching process between observed and source paragraphs.

The time measurement of the suffix tree algorithm follows the LCCW setting. The preparation time is the time beginning from loading observed paragraphs into memory until constructing it to be a suffix tree data structure. The loading time of resource index is none or zero because there is no index in suffix tree. The loading time of resource is a cumulative time of reading text paragraphs of source document related to paragraphs converted into bloom filter in LCCW algorithm. Finally, the matching time is the time needed in the matching process between observed paragraphs and source paragraphs in the source document.

A generalization is obtained by testing the proposed algorithm using the random data test. In this experiment, the random data tests are six data collections, five (FBIS, FT, JWS, LATIMES, ZIFF) from TREC and one from the RFC collection.

5. Result and Analysis

The statistics of RFC and TREC collections are tabulated in Table 1. From this table, it can be known that the size of the bloom filter is bigger than the original text and the biggest ratio is FBIS collection.

Based on Figure 4 it can be argued that the loading time of suffix tree outperform the LCCW algorithm except for LATIMES collection. From Table 1, LATIMES has a smallest paragraphs size so that it has a smallest ratio of the bloom filter size to the original text. From this fact, it can be argued that if we have a collection with a short paragraph we can expect that the loading time of LCCW is lower than the suffix tree algorithm. Based on the Figure 5, it can be concluded that the LCCW algorithm outperform the suffix tree for length of the observed paragraph below one hundred words except for FBIS collection. Based on the average length of paragraph and its standard deviation as shown in Table 1, it can be argued that one hundred words is enough to construct one paragraph

From Table 1, FBIS collection has a biggest ratio of the bloom filter size to the original text. From this finding, it is still open research to increase the performance of the LCCW algorithm by decreasing the loading time, for example by pipelining between loading and matching process or by omitting the stop word in order to reduce the paragraph size.

Tabel 1. Statistic of the testbed

Collection	# of words in the Paragraph		# of Para-graphs	# of Doc.	Size of Collection (Kbyte)	Size of Bloom Filter (Kbyte)	Ratio Bloom To Text
	Av.	STD					
RFC	41.11	42.07	584753	4380	202938	644556	3.18
FBIS	57.85	48.72	1106966	491	493071	1962966	3.98
FT	52.63	39.19	1497288	592	591563	2002030	3.38
JWS	51.57	32.32	1250669	783	568257	1367240	2.41
ZIFF	51.64	38.84	1854136	784	799525	2500096	3.13
LATIMES	38.91	20.15	1624401	729	498360	976600	1.96

Based on the characteristic of the LCCW algorithm, the LCCW algorithm is suitable to be implemented in an environment of plagiarism detection whereby the comparison can be conducted paragraph by paragraph. This condition is properly implemented in the distributed resources where the comparison is not in the whole document bases but in the relevant paragraph bases according to the concept terms of document collection.

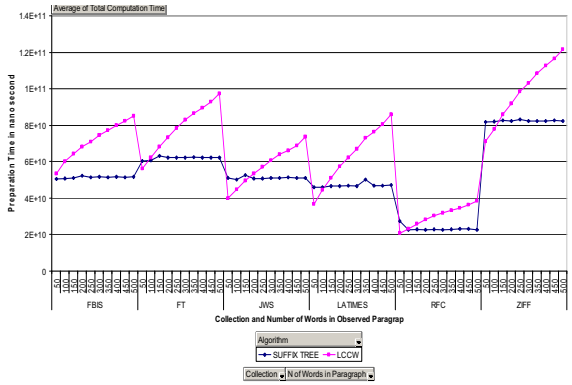

Figure 5. The Total Computation Time Comparison between LCCW and Suffix Tree algorithm versus number of words in the observed paragraph for several document collections.

The precision of the LCCW algorithm is shown in Figure. 6. Based on the Figure 6, it can be concluded that the precision of the algorithm increases if the density of the bloom filter decreases. Decreasing of the bloom filter is equal to decreasing of the false positive rate of the bloom filter. However, Decreasing of the false positive rate will increase the size of the bloom filter so that the loading time is increasing too.

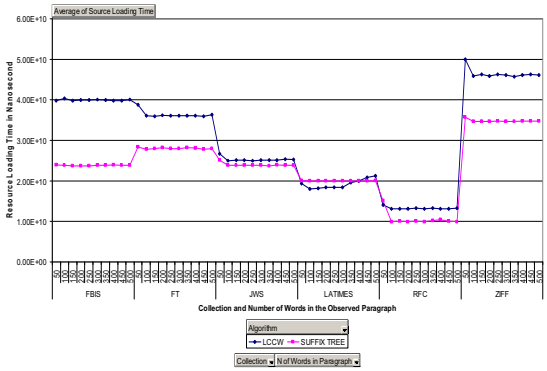

Figure 4. The Loading Time Comparison between LCCW and Suffix Tree algorithm versus number of words in the observed paragraph for several document collection

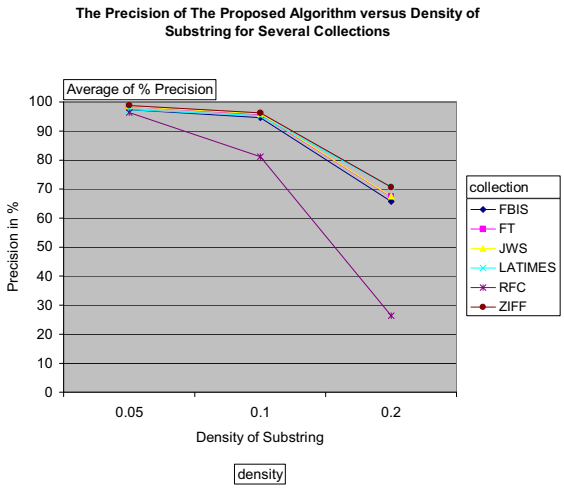

Figure 6. The precision characteristic of the LCCW algorithm versus the density of the Bloom Filter.

6. Conclusion and Future Work

Unlike a numerical based comparison, in the previous research, that is lost the location of the common parts, the proposed algorithms called the LCCW can find not only the common parts but also the location of the common parts. Therefore, the LCCW can fulfill the aim of the creation of the plagiarism detection tool. Moreover, the LCCW is also reluctant to insertion or deletion of words in document. Based on the experiment, it can be concluded that the proposed algorithm outperform the Suffix Tree algorithm in case of the length of the observed paragraph below one hundred words. Therefore, the LCCW algorithm is suitable to be implemented in an environment of plagiarism detection whereby the comparison can be conducted paragraph by paragraph.

The weakness of the proposed algorithm is the loading time. There is a possibility that has to be explored in depth in order to reduce a loading time, for instants by selecting only the relevant paragraphs that will be loaded or by pipelining between loading and comparing paragraph. A selecting relevant paragraph needs an appropriate filtering method so that the reduction of the detection precision can be minimized. Meanwhile, building pipelining between loading and comparing process needs a selecting the appropriate size of the bloom filter that has to be fragmented so that the overall time can be minimized.

References

[1] Jafri,S. B.A., Khalid, A. and Atiq-us-Salam: *Plagiarism : Opportunities and challenges in age of the Internet*. Retrieved September 27,2004 from http://www.basitali.com/plagiarism.html (2002)

[2] Lukashenko, Roman et al. Computer-based plagiarism detection methods and tools: An overview. International Conference on Computer System and Tools-CompSysTech'07. (2007).

[3] Manber, U.: Finding similar files in a large file system. ACM Journal (1994)

[4] Brin, S., Davis, J., & Garcia-Molina, H.: Copy detection mechanism for digital documents. ACM Journal on SIGMON, San Jose, CA USA (1995)

[5] Shivakumar, N. & Garcia-Molina, H. SCAM: A Copy detection mechanism for digital documents. ACM Journal (1995)

[6] Monostori, K., Zaslavsky, A., & Schmidt, H.: Parallel and distributed document overlap detection on the web. *Workshop on applied parallel Computing. PARA2000, Bergen, Norway* (2000)

[7] Zaslavsky, Arkady et al.: Using copy-detection and text comparison algorithms for cross-referencing multiple editions of literature works. *s of the 5th European Conference on Research and Advanced Technology for Digital Libraries* (2001)

[8] Finkel, R.A, Zaslavsky, A., Monostori, K. & Schmidt H.: Signature extraction for overlap detection in documents. *Proceedings of the 25th Australian Computer Science Conference, Monash University, Melbourne* (2002)

[9] Chowdhury, et al.: Collection statistics for fast duplicate document detection. *ACM Transaction on Information System Vol 20 No 2* (2002)

[10] Zhang, T., Damerau F.,&Jonhson D.: Text chunking based on a generalization of winnow. Journal of Machine Learning Research 2 (2002)

[11] White, D.R. & Joy, M.S.: Sentence-based natural language plagiarism detection. *ACM Journal on Education Resource in Computing, Vol 4 No.4.* (2004)

[12] Baeza-Yate, R. & Ribeiro-Neto, B.: *Modern information retrieval*. Addison Wesley (1999)

[13] Sediyono, A. & Ruhana: A new architecture of bloom filter for string matching. KMICE08: International Conference on Knowledge Management. Langkawi, Malaysia (2008)

[14] Conrad, J.G., Guo Xi S. & Schriber C.P.: Online duplicate document detection : Signature reliability in a dynamic retrieval environment. *CIKM, New Orlean, Louisiana, USA*. (2003)

[15] Broder, A., Glassman, S., Manasse, S, & Zweig, G.: Syntactic clustering of the web. *In Proceedings of the Sixth International World Wide Web Conference (WWW6'97). Santa Clara, CA.* (1997)

[16] Niezgoda, S. and Way, Thomas. SNITCH: A Software tool for detecting cut and paste plagiarism. SIGCSE'06. Houston, Texas, USA. (2006)