# Efficient Algorithm for Near Duplicate Documents Detection

**Gaudence Uwamahoro [1] and Zhang Zuping[2]**

**[1] School of Information Science and Engineering, Central South University
Changsha, 410083, China**

**[2] School of Information Science and Engineering, Central South University
Changsha, 410083, China**

**Abstract**

Identification of duplicates or near duplicate documents in a set of documents is one of the major problems in information retrieval. Several methods to detect those documents have been proposed but their relevance is still an issue. In this paper we propose an algorithm based on word position which provides a reduced candidate size to search in and increases efficiency and effectiveness for partial documents relevance. In our experiments the results show that during search process for the query the candidate size has reduced up to 12% of the size of set of documents which leads to a decreased time in searching. The results also have shown a higher accuracy thus helping help the user not to waste time on waiting for a query and getting unwanted documents.

***Keywords:*** *Inverted Index, Near-Duplicate Document, Partial Document, Document Relevance, Duplicates Detection.*

## 1. Introduction

The Internet as source of knowledge provides users with access to the abundant information on current research in different areas. To provide users with documents that satisfy their needs is a goal of information retrieval. But the relevance of documents containing that information is a major issue. The relevance must be both efficient and effective since many queries may need to be processed in short time and effectively since the quality of ranking determines whether the search engine accomplishes the goal of finding relevant information. There are several kinds of documents containing the targeted information including academic publications. In order to rapidly respond to the user's query, inverted index as inherent file structure is proposed. In this structure, one term records the identifiers of the documents containing that term, frequencies and sometimes position of the term in the document.

Digital documents are easy to modify using operations such as insertion, deletion and substitution which is the reason most of documents are duplicates and near duplicates to others. In those documents more paragraphs or sentences of one document appear in other documents and make it to be seen as near duplicates. Documents with similar partial contents relate to the same area. It takes time to get the information related to what the user needs and sometimes the results don't match the query. Several approaches to detect duplicates and near duplicate documents have been proposed but they still need improvement. Lack of speed in searching for a query and relevance ranking is an obstacle in research especially in information retrieval which makes it necessary for a powerful algorithm to be used to make the searching process faster and provide better results. Inverted index data structure with ability to increase the speed and effectiveness of search in documents collection is used to solve those problems using the ability of recording for each terms occurred in each document and its position in that document [1] In this paper our method "phrase query" is used and we are interested in showing how the query is processed efficiently and effectively. Some researchers have proposed a method of using inverted index where they first search all terms of query in the documents and then apply intersection algorithm which takes time to process. In our method, we propose the use of inverted index and recursion. The query processing time will decrease and accuracy will be established with more attention.

## 2. Related works

Many researchers in information retrieval have worked on duplicates detection. Recently, research on duplicates detection has focused on issues of efficiency and effectiveness. Duplicates and near duplicate documents detection is an interesting subject in current research in information retrieval.

### 2.1 Duplicates and near duplicates detection

It is easy to make a copy of a document to produce its duplicate. Minor changes like delete, insert and substitution on the document make that document to be a near document as in [2, 3]. There are several approaches to detect those documents like edit distance, fingerprinting, shingling, checksumming and bag of word. Similarity measures also are used as in [4, 5]. Shingles are the most

features used by many researchers where more shingles shared in two documents imply more likely that the documents are near duplicates. Most of the methods used have the same shortcoming of being efficient only for small size documents. The research continued and the new approach called fingerprinting to remediate to that drawback has been proposed in [6, 7]. Heintze and Manku invented the fingerprinting method where every shingle is fingerprinted [8] and documents are near duplicates if one of the fingerprints matches. Charikar in [9] proposed a method based on random projection of words in documents to detect near duplicate documents and improved the overall precision and recall. Xiao, W, L and J in [10] proposed new filtering technique by exploiting the ordering information in prefix filtering. Lakkaraju, P. Gauch, S., Speretta and M, proposed a method based on conceptual tree where they presented each document as a tree [11]. Research on partial copy detection is done in [12], in that research the method of calculation of sentence level similarity by comparing word overlap invented.

## 2.2 Inverted index

The inverted index also called inverted file is an efficient data structure used recently by search engines for indexing large volumes of text documents with increasing speed and provides efficiency searching in documents collection. An inverted index contains a list of documents that contain the index term for every term. It is inverted in the sense of being the opposite of document file that lists for every document the index terms it contains, hence the name "Inverted index". Inverted index is a powerful technique used in information retrieval because of its efficiency and effectiveness in document relevance as it has been used in duplicates detection [13, 14, 15]. Many researchers have used inverted index as in [16] where the method proposed uses sentence based on inverted index but the word position data is dropped and considers only the existence of each word in each sentence. In [17, 18, 19] inverted index has been used in document detection based on a sentence level. The search method proposed in this paper is based on word position in the document. A sentence level and recursion to achieve document relevance effectively and efficiently is proposed. The words position will lead to the location of terms in the document. A posting list as one of the main components of inverted index shows documents in which the term appears frequencies and its position. The Intersection of the posting lists can be used to find sentences that appear in different documents. The space used by the inverted index varies in range of 5-100% of the total size of the document indexed. That range is due to the implementation of inverted index, where some try to reduce the size of documents using compression methods like Delta coding, Golomb-Rice and Elias gamma and some store positions whereas others don't.

There are several variations on inverted indexes. The simplest form of inverted list stores just the documents that contain each word. If you want to support phrase query you need to store word positions for each document and also it is possible to add word frequency for each document. The typical example of inverted index of a set of four documents is shown in table 1. Those documents are:

*d1: Old people like green tea than porridge*
*d2: Children like hot maize porridge or cold soya porridge*
*d3: Some like it hot others like it cold*
*d4: Mothers keep maize porridge in big pot*

Table 1: Inverted index from documents

| **Token** | **Documents** | **Word position in document** |
|---|---|---|
| Children | d2 | d2, 1 |
| cold | d2 | d2, 7 |
| green | d1 | d1, 4 |
| hot | d2,d3 | [d2, 3], [d3, 4] |
| in | d4 | d4,5 |
| it | d3 | d3, 3, 7 |
| keep | d4 | d4, 2 |
| like | d1, d2, d3 | [d1, 3], [d2, 2], [d3, 2] |
| **maize** | d2, d4 | **[d2, 4], [d4, 3]** |
| mothers | d4 | d4, 1 |
| old | d1 | d1, 1 |
| or | d2 | d2, 6 |
| people | d1 | d1, 2 |
| **porridge** | d1, d2, d4 | **[d1,7], [d2,5], [ d2, 8], [d4, 4]** |
| some | d3 | d3, 1 |
| soya | d2 | d2, 8 |
| tea | d1 | d1, 5 |
| than | d1 | d1, 6 |

When looking for matches for query "maize porridge", the posting lists of those two words of query are considered, and intersection of two lists is used as follows:

- *maize: [d2, 4], [d4, 3]*
- *porridge: [d1, 7], [d2, 5], [d2, 8], [d4, 4]*

From the lists above we see that maize porridge appears in document $d_2$ where the word maize comes on position four and porridge on position five in the same document. The document $d_4$ also matches the query where maize is on position 3 and porridge on position four in the same document.

## 3. Word Positional Based Approach for Document Selection (WPBADS).

We propose a new approach which is based on term position in inverted index, phrase query and recursion to make the text document search fast and effective. The user will be able to submit the query to the system, and the system will display all the documents having the parts that match the query. Our method comes to support the improvement of document relevance ranking by decreasing searching process time and increase accuracy during partial document copy detection in order to reach the goal of relevant information. Recursion is one of the methods used in our approach. It is a method where the solution to a problem depends on the solution to smaller instances of the same problem.

Our method focuses on word position and recursion. Storing a mapping from words to their locations in documents make inverted index efficient and flexible and provides a good full search. Manipulating well inverted index can speed up and increase effectiveness of searching. With word position, we have to consider the order and position of each term in the query to find all documents that match the query. Before indexed, all documents in collection must be preprocessed by tokenization, stemming and removal of all stopwords. In our method we only consider tokenization and stopwords removal. After preprocessing, all tokens are indexed in vocabulary and a posting list that shows for each term in vocabulary and the list of documents it appears in is then established. As the vocabulary contains many terms, positions are very important to locate the term in each document. Not only word position in document is needed, but also the order of the terms of query is the key in the search process used in our method. Our method supports phrase query. When a user submits the query, the system will search in the vocabulary and will display all documents that match the query by order of relevance and score. The top high relevant documents are presented to the user.

### 3.1 Minimize candidate size for fast searching

Most of the methods used before search in every document each term of the query use intersection algorithm. Those methods increase the time of searching. We realized that even when a document contains a term of the query it doesn't mean it must be selected to serve the searching of query. We have also realized that all documents without the first term of query during searching do not serve for search of next term of query in a set of documents. There is no chance to get other next terms of the query in documents without the first term and that is the reason all documents that don't contain the first term are no longer used to find next terms of the query. The proposed method WPBADS allows selecting a document with importance during the search of a certain query and that leads to the decreasing in time because of the size of documents to search in decreases during searching process. To find the second term, the system must search from documents that have the first term. There is one condition to find the second term which is being next to the first term in document. After getting documents that contain those two terms, we concatenate them to get the new string which will be the base of our algorithm determined by the first two terms in the same order as in the query. The importance of considering the base is that if we consider only one condition of being next to the second, we can get wrong results because the second term can be in different documents next to the second term but the previous term of second term (first term) is different from the first term in the query. The documents used for searching in are the ones that contain only the base.

To continue our search for our query, WPBADS will be based on the previous base. The next term of the query to search must be next to the base and as a result the system displays all documents that contain the base and term to search. The next search continues to the next term of the query but we must concatenate the base with the previous term used to get the new base. Base is changed as we continue the search.

### 3.2 Algorithm description

Let the collection of documents $D_n$ be an $n$-tuple made of n documents, where n is an integer number. Therefore $D_n=<d_1, d_2, d_3 \ldots d_n>$, where $d_{i \mid n\geq 1}$ is the $i^{th}$ document and documents may have different sizes. Hence for each document $d_{i \mid n\geq i,}$ we associate its size $S_i$ such that we now have the couple $(d_{i \mid i<n}, S_{i\mid n\geq i})$. The tuple $D_n$ becomes $D_n=<(d_{i \mid i<n}, S_{i\mid n\geq i})>$, the total size of $D_n$ being $S_T = \sum_1^n S_{i\mid n\geq i}$ Let $UQ_k$ be the user query made of $k$ strings, where k is an integer number (number of terms of the query). Therefore $UQ_k=UQ_1+ UQ_2+ UQ_3+ UQ_4+ UQ_5+\ldots+UQ_k$. The proposed algorithm is described as follows: we query the presence of $UQ_{h \mid 1\leq h\leq k}$ string in every document, the first round of search having started *with* $UQ_1$. If the string is in a document, then the document is kept for the next round of search of $UQ_{h+1 \mid 1\leq h\leq k}$. The search contains k number of rounds at most since each string must be found in each document that had been kept. Hence for each round we may keep $j$ documents where $j$ is an integer number less than $n$. The algorithm is as follows: Initially we query the presence of $UQ_1$ for each document in $D_n$. For subsequent searches, let $IS_{j \mid 1\leq j\leq n}$ be the tuple of $j$ documents in which the $UQ_{h \mid 1\leq h\leq k}$ had been found at $h^{th}$ round. Therefore, we keep the tuple $IS_{j \mid 1\leq j\leq n,}$ and search

$UQ_{h+1 \mid 1 \le h \le k}$ in it. Therefore, if the tuple dimension is not zero for all rounds, the search goes on until $h=k$. The string to search is divided into terms and all documents are represented by their roots in a list. The algorithm searches a particular term in all the documents, if the term is not found then the document is deleted in the list of documents having that term.

### 3.3 Time and space complexity

The time complexity is measured by the number of elementary operations carried out during execution of program and space complexity is the computer memory used by our algorithm. In our algorithm, the operations considered are the ones done by binary search on the input size n used to search term in a document. The amount of work done during a single execution before and after loop is constant. The time of our algorithm is proportional to the number of time the loop executed. It is possible to reduce running time of our algorithm by reducing the number of candidate size to search in. That leads to the increasing speed of algorithm.

Using binary search in the proposed algorithm after each iteration, the input size to search in is decreased and it is less than the one for previous iteration. That it is why the time complexity of our algorithm is logarithmic and is $O$ ($log_2$ $n$). Since each comparison binary search uses halves of the search space, search process will never use more than $O$ (log $N$) comparisons to find the target term in a document. It is the same calculation for its space complexity. The more space we allocate for the algorithm, the faster it runs. The work space cannot exceed the running time. We know that writing in each memory cell requires at least a constant amount of time.

Thus if we let *T(n)*, time complexity and *S(n)* space complexity of our algorithm, then *S(n) = O(T(n))*.The space complexity of our algorithm is then *O (1)*.The function of our algorithm is logarithmic. As the number of documents to search in increases the time used to search in is decreased. The algorithm WPBADS is listed in table 2.

We are based on a collection of 8 documents in table 3 with different sizes and those documents are preprocessed by tokenizing their text, removing all stopwords, punctuations and lowercase all letters. Let 1391 bytes be the size of the collection of 8 documents with different sizes. Let 100 bytes be the size of $d_1$, 109 the size of $d_2$, 188 bytes the size of $d_3$, 171 bytes the size of $d_4$, 181 the size of $d_5$, 329 the size of $d_6$, 146 bytes the size of $d_7$ and 167 bytes the size of $d_8$. *The user query is "freshwater tropical fish tank and saltwater tropical tank".*

Table 2: WPBADS algorithm

***Algorithm:*** WPBADS

***Input***: D a set of documents to search in

User query: phrase containing terms

***Output*:** Documents that match the query presented by order of relevance

List of Documents $D = \{d_1, d_2, d_3, d_4, \ldots\ldots, d_n\}$

List of Query terms $T = \{t_1, t_2, t_3, t_4, \ldots\ldots\ldots, t_n\}$

***FOR*** each node $t_i$ in $T$

Get the term $t_i$

***FOR*** each node $d_i$ in $D$

FOUND = searchTerm ($t_i$ in $d_i$)

***IF*** Not FOUND ***THEN***

DELETE $d_i$ in $D$

***ELSE***

Pos ←get the position of $d_i$

***IF*** Pos is not 0 and Pos is different to $d_i$.position + 1 THEN

DELETE $d_i$ in $D$

***ELSE***

$d_i$.position ← Pos

***END IF***

***END IF***

***NEXT*** node in $D$

***NEXT*** Node in $T$

***FUNCTION*** searchTerm Parameters root: pointer to the root, term: string

***BEGIN***

IF root.info == term ***THEN***

Return (root)

***ELSE***

***IF*** root.info > term ***THEN***

Return searchTerm (Right Son of root, term)

***ELSE***

Return searchTerm (Left Son of root, term)

***END IF***

***END IF***

***END***

The proposed algorithm follows different steps and the number of those steps depends on the number of terms of query. We have used Vector Space Model to represent the text document. In that model a document is represented by a vector of keywords extracted from the document with associated weights representing the importance of keywords in the document within the whole documents set.

Table 3: Ranking of documents in order by score

*d1: tropical fish include fish found tropical environments word including freshwater salt water species*
*d2: fishkeepers use term tropical fish to refer requiring freshwater saltwater tropical fish referred marine fish*
*d3: tropical fish popular aquarium fish due bright coloration fishkeepers use term tropical fish refer particularly requiring freshwater tropical saltwater tropical fish referred marine fish*
*d4: freshwater fish coloration typically derives iridescence salt water fish generally pigmented marine tropical fish interest fishkeepers fish live close relation coral reefs*
*d5: articles library contains large sections cichlids beta fish aim offer articles targeting beginners experienced tropical fish keepers training tropical fish disease aquariums general*
*d6: saltwater aquarium complicated handle basic freshwater tropical fish tank invested time efforts rewarded possibility keep remarkably beautiful fascinating saltwater fish species find freshwater tropical fish tank saltwater tropical fish tank beginner guides help pet fish way hopefully aquarium beginner guides help learn basics*
*d7: ac tropical fish started out site tropical aquarium fish keepers has grown include areas such coldwater species set maintenance saltwater aquarium*
*d8: find freshwater tropical fish tank and saltwater tropical fish tank beginner guides help pet fish way hopefully aquarium beginner guides help learn basics buying fish*

To weigh a term the $tf \times idf$ method is used where the term frequency $tf_{i,j}$ and document frequency $df_j$ are the main factor for weighting the term. The weight of term $j$ in document $I$ is: $w_{ij} = tf_{i,j} \times idf_j = tf_{i,j} \times log\ N/df_j$ where N is the number of the documents collection and $idf_j$ is the inverse document frequency. The term weight helps us for documents ranking. The ranking function is necessary to measure similarity between document vectors and the user query. Cosine similarity measure is used to determine angle between query and document vectors. The similarity between query $Q$ and document $D_i$ is given by:

$$sim(Q, D_i) = \frac{\sum_{j=1}^{v} w_{q,j} \times w_{i,j}}{\sqrt{\sum_{j=1}^{v} w_{q,j}^2 \times \sum_{j=1}^{v} w_{i,j}^2}}$$

Where $W_{q,j}$ is the weight of term $j$ in the query and is defined in the same way as in $w_{i,j}$ ($tf_{q,j} \times idf_j$). In our documents set the table 4 shows the results:

Table 4: Ranking of documents in order by score

| D.ID | R.Q(score) | D.R.Sc (traditional) | | R.D.P.Us.Sc using WPBADS | |
|---|---|---|---|---|---|
| | | Order1 | D.ID | Order2 | D.ID |
| $d_1$ | 0.511890 | 1 | $d_6$ | 1 | $d_8$ |
| $d_2$ | 0.578541 | 2 | $d_8$ | 2 | $d_6$ |
| $d_3$ | 0.604367 | 3 | $d_3$ | 3 | $d_3$ |
| $d_4$ | 0.361961 | 4 | $d_2$ | 4 | $d_2$ |
| $d_5$ | 0.367404 | 5 | $d_1$ | 5 | $d_1$ |
| $d_6$ | 0.695353 | 6 | $d_7$ | 6 | $d_4$ |
| $d_7$ | 0.406181 | 7 | $d_5$ | 7 | $d_5$ |
| $d_8$ | 0.622375 | 8 | $d_4$ | 8 | $d_7$ |

The following abbreviations are used in table 4 above. RQ represents result to the query by score which is similarity to the query; D.R.Sc represents documents ranking in order by score. This is the traditional method where documents are presented to the user by their score to the query. R.D.P.Us.Sc is used to represent relevant documents presented to the user in order by score using WPBDS. The similarity measure like cosine is used to know how documents are similar to the query and high relevant documents are presented to the user by order of score. The typical example shown is the order 1 in the table 4 above. Represent relevant documents considering only order of score traditionally as in order 1 doesn't interest the user who needs documents containing the information he needs.

Using WPBADS for partial document detection, the order of high relevant documents presented to the user is different from the one in order 1. The user judges the relevance of retrieved documents according to what he wants. The user will be interested in order by score in order 2 column where high relevant documents contain more information he wants. For partial documents, documents are near similar when they contain more query terms in the same order they are presented in the query. In our method, the size of documents to search in is decreased during search process and that leads to a higher efficiency because the size of the documents to search in is decreased. That will have effect on effectiveness of the results wanted**.** The methods used before like inverted index, and others considered as traditional methods they use intersection algorithm after they search every term of the query in each document in a collection, and that takes time. The proposed method provides a higher efficiency as the size of documents to search in is reduced according to the number of terms in the query and that leads to a reduced time of query processing as shown in figure 1**.**

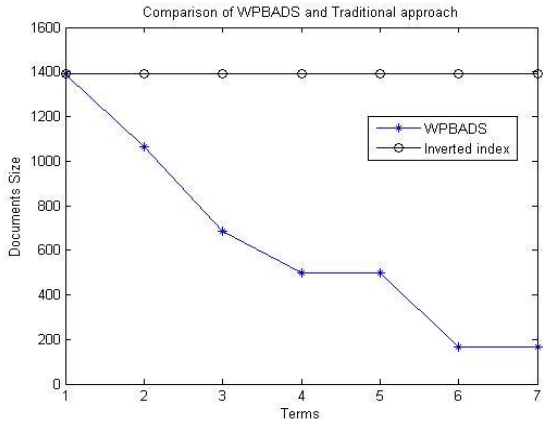


Figure 1: Comparison of WPBADS, inverted index and traditional approaches

## 4. Conclusion

Although research on duplicate documents detection is going on until now, efficiency and effectiveness for document relevance is still an issue and needs improvement. In this paper we proposed an efficient algorithm for near duplicate documents detection by exploiting position and order of term in documents. With this method we show that words position plays great role in documents relevance where it is the base for documents candidate selection. The results show that the proposed method provides effectiveness and high efficiency by reducing the documents size to search in up to 12% and that leads to the decreased computation time in partial document detection. In future we intend to investigate compression methods in our method for the query efficiency.

### Acknowledgments

We are grateful to the support of the National Natural Science Foundation of China (Grant No. 60970095, M1121008) and Research Fund for the Doctoral Program of Higher Education of China (Grant No. 20120162110077).

## References

[1] Ajik Kumar Mahapatra, Sitanath Biswas, “Inverted index Techniques”, International Journal of Computer Science Issues, Vol. 8, Issue 4, No. 1, July 2011.

[2] De Carvalho, M. G., Laender, A. H. F., Goncalves, M. A., & da Silva, A. S., “A genetic programming approach to record deduplication”, IEEE Transactions on Knowledge and Data Engineering, Vol. 24, No. 3, 2012, pp. 399–412.

[3] Valls, E. & Rosso, P., “Detection of near-duplicate user generated contents: the SMS spam collection”, in Proceedings of the 3rd international workshop on search and mining user-generated contents, 2011, pp. 27–34

[4] Yung-Shen Lin, Ting-Yi Liao, Shie-Jue Lee, “Detecting near-duplicate documents using sentence-level features an supervised learning”, 2012, pp. 1467-1476

[5] B. Karthikeyan, V. Vaithiyanathan, C. V. Lavanya, “Similarity Detection in Source Code Using Data Mining Techniques”, European Journal of Scientific Research ISSN 1450-216X Vol.62, No.4, 2011, pp. 500-505.

[6] Gurmeet Singh Manku, Arvind Jain and Anish Das Sarma, “Detecting near-duplicates for web crawling”, in Proceedings of the 16th international conference on World Wide Web, Banff, Alberta, Canada, 2007, pp. 141 – 150

[7] M. Charikar, “Similarity Estimation Techniques from Rounding Algorithm”, in Proc. of 34th Annual Symposium on Theory of Computing (STOC), 2008, pp. 380-388

[8] N. Heintze, “Scalable document fingerprinting”, in Proc. of the 2nd USENIX Workshop on Electronic Commerce 216, 1996, pp. 191–200

[9] M. S. Charikar, “Similarity estimation techniques from rounding algorithms”, In Proceedings of 34th Annual ACM Symposium on Theory of Computing, (Montreal, Quebec, Canada, 2002, pp. 380–388

[10] C. Xiao, W.Wang, X.Lin, J.X.Yu, “Efficient Similarity Join for Near Duplicate Detection”, Beijing, China, 2008

[11] Lakkaraju, P. Gauch, S., Speretta, M., “Document similarity Based on Concept Tree Distance”, Proceedings of Nineteeth ACM conference on Hypertext and Hypermedia. Pitteburgh, PA, USA, 2008, pp.127-132

[12] Metzler, D., Bernstein, Y., Croft, W.B., Moffat, A., Zobel, J., “Similarity Measures for Tracking Information Flow”, In: The 14th ACM Conference on Information and Knowledge Management (CIKM 2005), 2005, pp.517–524

[13] Zobel, J and Moffat, “Inverted files for text search engines”, ACM Computing Surveys, Vol. 38, No. 2, article, 2006, pp. 1-55

[14] Justin Zobel, Alistair Moffat, and Ron Sacks- Davis, “An efficient indexing technique for full text databases”, 1992, pp.352–362

[15] Ajik Kumar Mahapatra, Sitanath Biswas “Inverted index Techniques”, International Journal of Computer Science Issues, Vol. 8, Issue 4, No. 1, 2011.

[16] James P. Callan “Proximity Scoring Using Sentence-Based Inverted Index for Practical Full-Text Search”, 2008

[17] Yerra, R., and Yiu Kai, NG., "A sentence-Based Copy Detection Approach for Web Documents", Lecture Notes in Computer Science, Springer Berlin / Heidelberg, Vol. 3613, 2005, pp.557-570.

[18] Allan, J., Wade, C., Bolivar, A., “Retrieval and novelty detection at the sentence level”, in: Proc. SIGIR-2003, the 26th ACM Conference on Research and Development in Information Retrieval, Toronto, Canada, ACMPress, New York, 2003, pp. 314–323.

[19] Li, X., Croft, B.: “Novelty detection based on sentence level patterns”. In: Proc. CIKM-2005. ACM Conf. on Information and Knowledge Management, ACM Press, New York, 2005.

**First Author:** Gaudence Uwamahoro received Master Degree of Engineering in Computer Science and Technology from Central South University in 2010**.** She is currently working towards her Ph.D. Degree at the School of Information Science and Engineering, Central South University, China. Her research interests include information system, database technology and data mining.

**Second Author:** Zuping Zhang received the Ph.D. degree in Information Science and Engineering, Central South University in 2005. He is now a Professor in School of Information Science and Engineering, Central South University. His current research interests include information fusion and information system, parameter computing and biology computing.