International Journal of Database Theory and Application
Vol.8, No.4 (2015), pp.59-70
http://dx.doi.org/10.14257/ijdta.2015.8.4.07

Efficient Pairwise Document Similarity Computation in Big Datasets

[1]Papias Niyigena, [1]Zhang Zuping*, [2]Weiqi Li and [1]Jun Long

[1]*School of Information Science and Engineering, Central South University, Changsha, 410083, China*
[2]*School of Electronic and Information Engineering, Xi'an Jiaotong University Xian, 710049, China*
*papiasni@yahoo.fr, * zpzhang@csu.edu.cn, liweiqi@stu.xjtu.edu.cn, jlong@csu.edu.cn*

Abstract

Document similarity is a common task to a variety of problems such as clustering, unsupervised learning and text retrieval. It has been seen that document with the very similar content provides little or no new information to the user. This work tackles this problem focusing on detecting near duplicates documents in large corpora. In this paper, we are presenting a new method to compute pairwise document similarity in a corpus which will reduce the time execution and save space execution resources. Our method group shingles of all documents of a corpus in a relation, with an advantage of efficiently manage up to millions of records and ease counting and aggregating. Three algorithms are introduced to reduce the candidates shingles to be compared: one creates the relation of shingles to be considered, the second one creates the set of triples and the third one gives the similarity of documents by efficiently counting the shared shingles between documents. The experiment results show that our method reduces the number of candidates pairs to be compared from which reduce also the execution time and space compared with existing algorithms which consider the computation of all pairs candidates.

Keywords: *Pairwise similarity, Performance, Shingles, Document similarity Algorithm*

1. Introduction

The resemblances of different documents have been a critical issue for long time. Nowadays communication technology, especially the use of the internet, has given facilities to easily access different source of documents. An explosive growth of data on the internet brings challenges due to the excessive cost in storage and processing. The need of copying a part of documents or articles of a paper or simply get its duplicate is very common in academia for example students or researcher to get more publication, in magazines and newspapers; and in project sponsorship organization where two people may want to get a fund to the same project. There is a need to know if the document in consideration has a resemblance to other existing documents, knowing that documents with very similar content provide little or no new information to the user. The computation of pairwise document similarity is to sort out, in a corpus, all documents compared one to another and give the similar document by comparing one document to the rest of the documents in the corpus. It is often both impractical and extremely tedious and expensive to evaluate the pairwise similarity of documents given the high number of documents to be compared to. The problem of computing pairwise documents similarity in the corpus is particularly challenging because of a potentially high cost of computation which may be required in such a process [1, 17]. For large document collections and query sets, this can quickly become impractical. The importance of document similarity

ISSN: 2005-4270 IJDTA

is recognized in many publications especially nowadays where access to information is nearly unrestricted and a culture for sharing without attribution is a recognized problem [5, 14].

To tackle the problem of pairwise computation, we represent documents in a corpus by a matrix M×N where M is the Universal set of all shingles; N is the set of all documents. In this matrix, rows represent shingles (elements), and columns represent documents (sets). A given cell in the matrix has the value 1 if the shingle in the row M belongs to the document of the column N, and value 0 otherwise. This will result in a sparse matrix where we will have more zeros than ones.

Given the number of available documents to compare, it becomes a complex task to manually specify the resemblance of one document to the rest of other documents in the corpus. The approach of learning-based has been proved to perform well as this task is to classify each pair and determine if either they match or not match. This approach comes with a cost because the high effectiveness of learning-based approaches comes with poor efficiency [10]. Our goal is to significantly reduce the computational time without compromising on the quality of retrieved documents.

The similarity between two documents is a number between 0 and 1 where if the number is close to 1, the documents are roughly the same. If we consider resemblance of documents by considering the documents like set of string [17], the similarity of two documents A and B will be given by Jaccard similarity, expressed by the following formula 1:

$$\mathrm{Sim}(\mathrm{A},\mathrm{B}) = \frac{|\mathrm{A} \cap \mathrm{B}|}{|\mathrm{A} \cup \mathrm{B}|} \quad (1)$$

Formula 1 describes shingles in common compared with all shingles in *A* and *B*.

A key process of document similarity system is the similarity computation. To deal with a large number of calculations, most of the previous works employed some type of sampling or hashing strategies to reduce complexity and save storage space, but their accuracy may be affected in some scenarios. For example, the minwise hashing algorithm may not generate good results when the resemblance between two documents is close to 0.5 [2]

We propose a new aspect to improve the detection of similar or near similar documents by reducing the computation time for big datasets. To achieve this objective, we combine fast retrieval data structure and use an efficient shingling manipulation algorithm which can be used in searching process. The method we are proposing uses shingling to represent documents and reduces the number of candidates pairs by heuristically limit the files to compare if their comparison doesn't give any contribution to the comparison performed before.

Hashing is commonly the technique used in Natural Language Processing (NLP) and information Retrieval (IR) tasks that are used in order to achieve faster word retrieval. The problem of computing pairwise document similarity can be succinctly defined as follows: for every document d in collection C, compute the top k documents for similar documents according to a particular term weighting model.

It was observed that the introduction of special structure based on the hashing technique, known as shingling or chunks can substantially improve the efficiency of deciding about the level of similarity of two documents by observing the number of common shingles. According to Alzahrani survey on plagiarism detection methods [3], common plagiarism detection techniques rely on character based method to compare the suspected document with the original document. Identical string can be detected exactly or partially using character matching approach. Chim et al. [6] consider text comparison based on word n-grams, where the suspected text is split into two set of trigrams to be compared. A number of common trigrams are considered in order to detect potential plagiarism cases. Mishra et al [7] introduced a plagiarism detection system from Stanford

Digital Library project named COPS which detects documents overlap relying on string matching and sentences.

There were other methods used to detect document resemblance or near document similarity. The cluster-based method is one of the information retrieval techniques that are used in many fields such as text summarization [7], text classification, and plagiarism detection [8]. Zini et al. have shown the similarity of documents using a cluster-based method [9]. He mentioned that the fingerprinting techniques mostly rely on the use of k-grams since the process of fingerprinting divides the document into grams of k-length, the fingerprinting of two documents can be composed in order to detect plagiarism. Fingerprinting was used for document clustering to summarize a collection of documents and build a fingerprint model for it [9].

Charikar proposed a locality sensitive hashing scheme for comparing documents as expressed in [11]. He proposed an improvement to the algorithm based on sequence matching, which determines the location of duplicates parts in documents. Algorithms based on shingling are commonly utilized to identify duplicates or near duplicates because of their abilities to perform clustering tasks in linear computational complexity [11].

Through this literature review, we show that efforts have been made to detect the similarity between text documents. These methods obtain good result but some fail when the duplicate or near duplicate document part is modified by rewording or changing some words, and the main drawback for the method proposed by Alzahrani et Al. [3] is that it fails to consider individual words and takes the whole sentence as one part. The main drawback for algorithms based on shingling is the big number of shingles produced and increases the time complexity to near quadratic. This number of shingles also affects efficiency because the system has to resort to the external storage because mostly they cannot fit in the main memory.

The main difference between our proposed method and these techniques is presented in two points: first our proposed method reduces the number of shingles to be compared; for example the shingles belonging only in one file are discarded. The second point corresponds to comparison mechanism: heuristically our method discard unnecessary files in comparison process because their comparison cannot bring any information regarding the similarity of documents we are looking for. This reduces the time complexity which before was quadratic to almost linear complexity. Our work could be leveraged by these applications for improved performance and higher accuracy.

2. Proposed Method

Documents are represented in a characteristic matrix. This matrix represents sets in a way to ease their manipulation and computation. In a characteristic matrix, sets are represented in columns and elements are represented in rows. The document is represented as a set, where tokens or shingles are its elements. To represent documents in a characteristic matrix, columns of the matrix correspond to the documents and the rows correspond to the elements of the universal set of shingles [11]. The matrix has 1 in row ***r*** and column ***c*** if the element for row r is a member of the set for column ***c***, otherwise the value in position (***r***, ***c***) is 0. Table 1 shows the representation of documents in the characteristic matrix.

In a large corpus with thousands of documents, this representation of documents as the set becomes impractical and their computation might be huge [7, 11]. We are presenting a new method to save space where we only consider the cells in the matrix having value 1 which reduce the computation time where the number of pairs to compare is reduced.

To avoid the sparse matrix, we use a data structure like relation, which can manage efficiently up to millions of rows and which can ease counting and aggregation operations

that will be needed in the documents resemblance computation. This relation contains all shingles of the universal set and their respective document.

Table 1. Documents Representation by Sparse Matrix

	doc1	doc2	doc3	doc4	doc5	doc6	…		doc*n*
shing1	0	1	0	0	1	0	…		1
shing2	1	0	0	0	1	0	…		0
shing3	0	0	1	0	0	0	…		0
shing4	1	0	0	1	0	1	…		1
shing5	0	1	1	0	0	1	…		0
shing6	0	0	0	0	0	0	…		1
shing7	1	1	0	1	0	1	…		1
shing8	0	0	0	0	0	0	…		0
shing9	0	0	0	0	0	0	…		0
shing10	0	0	0	0	0	0	…		0
shing11	0	0	0	0	0	0	…		0
shing12	0	0	0	0	0	0	…		0
…	…	…	…	…	…	…			
shing*n*	0	0	0	0	0	0	…		0

2.1 Reducing Computation Time

Pair-wise documents similarity result is an upper triangular matrix because the similarity between two documents is symmetric. This means $SIM(A,B) = SIM(B,A)$. We can take advantage of this property to reduce the number of computation while we are looking for the similarity of documents in a corpus [12]. By sorting the files in decreasing order of documents number of shingles, at given time in the process we get the solution before we scan all documents, smaller documents being incorporated in bigger documents. Here we can reduce the time complexity from O (n) to O (log n).

The relation to be created in order to represent the set of all shingles in the collection has two attributes. Its row represents a pair (***a***, ***b***) such that ***a*** is a shingle in the document ***b***. In Fig. 2 this relation is represented by List of Tuples. Our objective is to find the triple (***u***, ***v***, ***w***) such that there is a link from ***u*** to ***v*** (which means that ***u*** document has ***v*** shingle)and a link from ***v*** to ***w*** (which means that ***v*** shingle belongs to document ***w***). Here, we essentially want to take the natural join of doc/shingle with itself, and we can achieve that by taking our relation as two relations *L1* and *L2*. That is, for each tuple t1 of *L1* and each tuple *t2* of *L2* check if their v component is the same [13]. The idea behind implementing natural join can be observed if we look at the specific case of joining $R(A,B)$ with $S(B,C)$. We must first find tuples that agree on their B components. That is the second component from tuples of R and 1st component tuples of S. We shall use B value of tuples from either relation as the key.

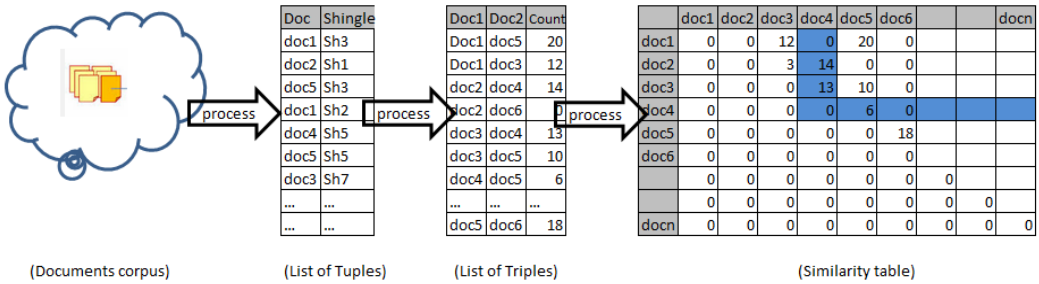
For each tuple (a,b) of R, produce the key-value pair $[b(r,a)]$. For each tuple (b, c) of S, produce the key-value pair $[b,(s,c)]$. Each key value b will be associated with a list of pairs that are either of the forms (R,a) or (S,c). The output for key b is (*b*, [(*a1*, *b*, *c1*), (*a2*, *b*, *c2*)…]) that is, b associated with the list of tuples that can be formed from a R-tuple and S-tuple with a common b value. After this process, we count the number of

common shingles between (*R*, *a*) and (*S*, *c*) as expressed by the relation List of Triples in Fig. 1. The pairwise similarity is represented as an upper triangular matrix where rows and columns represent documents and cell represent the number of shared shingles. As expressed in Fig. 1 by Similarity Table, the pairwise similarity of any document is given by its intersection cell with all other documents in the collection. In the next sections, we propose the algorithms to perform this process.

Before applying our algorithm, text preprocessing should be carried out. Stopwords are frequently occurring, insignificant words that appear in documents and they are useless to use in the similarity of documents [13]. Stopwords lists and stemming algorithms are two commonly used information retrieval techniques for preprocessing text documents. Action that make up the process of text-refinement in documents starts from extracting lexical units (tokenization), and further text-refinement operations are: elimination of the words in stopwords-list, identification of multi-word concept and bringing concept to the main form by stemming [13]. The goal of stemming is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

2.2 Create the Relation for Pairwise Computation

The most effective way to present documents as a set is to construct from the document the set of short strings that appear within it [12]. The documents that share pieces as short as sentences or even phrases will have many common elements in their sets, even if those sentences appear in different orders in the two documents. K-shingle for a document is any substring of length k found within the document.



Doc	Shingle
doc1	Sh3
doc2	Sh1
doc5	Sh3
doc1	Sh2
doc4	Sh5
doc5	Sh5
doc3	Sh7
...	...
...	...

Doc1	Doc2	Count
Doc1	doc5	20
Doc1	doc3	12
doc2	doc4	14
doc2	doc6	0
doc3	doc4	13
doc3	doc5	10
doc4	doc5	6
...	...	...
doc5	doc6	18

	doc1	doc2	doc3	doc4	doc5	doc6			docn
doc1	0	0	12	0	20	0			
doc2	0	0	3	14	0	0			
doc3	0	0	0	13	10	0			
doc4	0	0	0	0	6	0			
doc5	0	0	0	0	0	18			
doc6	0	0	0	0	0	0			
	0	0	0	0	0	0	0		
	0	0	0	0	0	0	0	0	
docn	0	0	0	0	0	0	0	0	0

Figutre 1. General Process of the Proposed Method

To start the process, our method reads the corpus. Two cases may appear: the corpus fit in memory or the corpus is too big to fit in memory [1]. In our method, we concentrate the efforts in the first option where corpus can fit in the memory of the system. The Fig. 1 shows the full process of our method: the methods start by reading the corpus which will be tuned in a data structure like a relation. After that, it computes the intersection between documents by counting the common tuples and finally build a table to show the pairwise similarity.

The basic approach for computing resemblance is expressed as a set of intersection problem. The reduction to a set intersection problem is done via a process called shingling. In shingling each document *D* gets an associated set *Sd*. This is performed as follows: we view each document as a sequence of tokens. Tokens may be letters, words or lines. We assume that we have a parser program that takes an arbitrary document and reduce it to a canonical sequence of tokens. Any duplicates would be removed. When we have the two sets of shingles for two different documents, we could calculate the Jaccard coefficient, which gives a similarity score between two sets. The coefficient is defined as the size of the intersection, the number of shingles present in both, divided by

the size of the union, the number of shingle present in either document. This is given by the following formula, considering document *A* and document *B*:

$$\mathrm{Sim}(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (2)$$

The way to process documents in a corpus are so varied, application and language dependent. Here, a document is represented by a set of token which will be divided and grouped to produce a set of shingles according to a number specified by the user. If the given number is small, we will have a good recall ratio while picking the big value will provide a good precision ratio. The important thing to remember in picking the value is that it should be picked large enough that the probability of any given shingle appearing in any given document is low. Our objective is to represent all documents in the corpus in different vectors, which will allow us to represent the document in a form of a relation of pair (*Dc*, *Sh*), that says, the document *Dc* has the shingle *Sh* as shown by Fig. 2.

	Shingle	Doc
1	child, caus, sever, chang, famili, none	cv722_7110.txt
2	friend, sitcom, dialogu, allow, littl, room	cv321_12843.txt
3	exley, cool, calcul, action, slight, hint	cv156_10481.txt
4	investig, succe, attempt, origin, funni, complet	cv071_12095.txt
5	audienc, best, kind, children, entertain, eleg	cv885_12318.txt
6	stone, wasn, likeabl, brook, right, idea	cv829_20289.txt
7	mcnaughton, decid, give, someth, memor, consequ	cv681_9692.txt
8	miss, cross, rosemari, herman, begin, date	cv384_17140.txt
9	lot, great, thing, act, amaz, oscar	cv503_10558.txt
10	immens, morrow, also, good, play, younger	cv973_10066.txt
11	stage, grand, stori, butler, learn, continu	cv862_14324.txt
12	world, know, kill, black, junki, even	cv936_15954.txt
13	can, not, stress, enough, attent, unlik	cv986_13527.txt
14	father, help, oliv, platt, local, drama	cv181_14401.txt
15	scene, import, becom, appar, crucial, scene	cv278_13041.txt

Figure 2. Shingle by Document Relation

The idea behind this is to find the triple (*U*, *V*, *W*) such that there is a document *U* having *V* shingle, and that shingle *V* is also in document *W*. We look for each tuple *t1* of L_1 (ie. Each tuple of Doc, Shing) and each tuple t_2 of L_2, see if their shingle component is the same. Note that these components are the second component of t_1 and the first component of t_2. If these two components agree, then produce a tuple schema (V_1, V_2, V_3). This tuple consists of the first component of t_1, the second component of *t1* (which must equal to the first component of t_2) and the second component of t_2. The algorithm will scan the relation and for each document in the corpus, it identifies the intersection between the shingles of documents and the relation created. The results give all documents having in common one or more shingles. We can easily create a triple (V_1, V_2, V_3) such that, V_1 is the document concerned, V_2 and V_3 is a tuple of the result of applying the intersection between document and relation. To be able to produce this relation, two algorithms are proposed: the first one creates the relation of all shingles and the documents they belong to while the second algorithm creates the relation of triples to be used in evaluation of similarity.

Create_Shingle_Relation Algorithm

//This algorithm construct a relation of tuple Document Shingle, where document has shingle
Input: The path of the corpus (Set of documents and the number of n-grams
Output: A set of tuple (Doc, Shingle)

```
1   LTuple = Empty list
2   LDocument = Empty list
3   Reader : The corpus reader
4   Text: The empty text
5   //Get the list of all documents names from corpora
6   LDocument = Reader.GetDocumentName(Path_Of_Corpus)
7   For each D in LDocument
8           Text = Reader.GetDocumentText(D)      //Read the text of document D from disk
9           Text = RemoveStopWords(Text)
10          Text = RemovePonctuation(Text)
11          Text = Steemer(Text)
12          LShingle = DivideInnGrams(Text, n)
13          For each LS in LShingle
14                  Tp = produce_Tuple(D, LS)
15                  LTuples .Add(Tp )
16          End for
17  End for
18  Return (LTuples)
```

The execution time of this algorithm is the time used to generate the list of all shingle for every document in the corpus (from line 12 to 16). As this operation will be performed for every document, the execution time depends to the number of document in the corpus (d) times number of all shingles in the corpus (n), which is O($d * n$). Traditionally, the shingling algorithms have a high computational complexity in time because they are O (n^2) (n is the number of documents in the collection).

The following algorithm creates the list of triples (a, b, c) such that a is the document having b as shingle and the shingle b belongs to the document c. Actually this is the bottleneck of other methods because the worse case may be quadratic. The main objective of this algorithm is to reduce the candidates' pairs to be considered in documents similarity computation, knowing that the big number of candidates pair affect efficiency. Given that the similarity of two documents is symmetric, the algorithm is improved such that after comparing a document, we do not need to recompute its similarity by any other document. This is achieved by keeping the track of document we have finished comparing so that we compare only the difference of shingles remaining.

Create_Triples Algorithm

//Construct a relation of triples, where two document share a shingle
Input: The list of tuples created by the previous algorithm
Output: A list of triples (a, b, c) such that a is the document having b as frequent term and b also is in the document

```
1   ListTriples = Empty set
2   DocShingles = Empty set
3   SetShingles = Empty set
4   For each D in ListOfDocument
```

```
5       SetShingles = ListTuples - ListTriples.Shingles //this is to consider the symmetric of
6   pairs
7       If SetShingles ≠ ø then
8           DocShingles = D.Shingles ∩ SetShingles
9           For each triple in DocShingles
10              tripleVar = (D, triple.doc, triple.Shingle)
11              ListTriples.Add(tripleVar)
12          End For
13      End if
14  End for
15  Return(ListTriples)
```

This algorithm produces the list of triples. Line 5 executes the reduction of the set of pairs to compare, by reducing the triples of documents we have already finished. This implies that the number of files to compare is less than the number of files in the corpus. Thus, we do not need to perform the Cartesian product, but files having shingles to be compared will enter the process. The time complexity is O(*n*log *m*) (n is the number of documents and m is the set of shingles). Actually, the naïve algorithm has the time complexity of O(n^2) where n is the number of shingles in the collection.

2.3 Similarity Calculation

The last step of our method is to produce the final relation of triples (*a*, *b*, c) such that a is the document in the corpus, b is another document in the corpus, both documents share c numbers of shingles. This relation provide all parameters we need to compute the Jaccard Similarity because it represents the intersection between document a and document b.

Create_Shingle_Relation Algorithm

```
//Construct a relation of tuple Document Shingle, where document has shingle
Input: The list of all shingles of the corpus and the Document to compare
Output: A set of triple (Doc1, Doc2, Count) where Doc1 and Doc2 have in common
Count Shingles
LComp = The empty set
ListResult = the empty set
LdcShingle = The empty set
LdcShingle = Read the shingles of Document D
LComp = LdcShingle ∩ LTuples        //Here we consider only shingles in document D
ListSh = Add (Lcomp, D)        //Here, all shingles in LComp having same shingle as D are combined
ListSh.SortBy(D)
For each doc in ListSh
    N = Count the number of shingles having doc
    Tpl = create_Triple (D, doc, N)
    ListResult.Add(Tpl)
End for
Return ListResult
```

We do remember that the list is sorted by the document, which will reduce the time complexity. In the algorithm, a document is produced as a candidate if it matches with any other document (from line 5 to line 8). After the identification of files to be compared, the algorithm starts counting the singles in common. The time complexity of

this algorithm is O (*n*), where n is the number of pairs to count. This is an improvement to the execution time of other methods because they have to go through all the candidates' pairs in the collection. The relation produced by the algorithm is shown by Figure 3.

DocId	DocShared	NbrShingles
cv005_29443.txt	cv008_29435.txt	1
cv162_10424.txt	cv163_10052.txt	2
cv022_12864.txt	cv176_12857.txt	2
cv162_10424.txt	cv231_10425.txt	2
cv163_10052.txt	cv231_10425.txt	2
cv162_10424.txt	cv248_13987.txt	2
cv163_10052.txt	cv248_13987.txt	2
cv231_10425.txt	cv248_13987.txt	2
cv115_25396.txt	cv274_25253.txt	601
cv258_5792.txt	cv282_6653.txt	11
cv202_10654.txt	cv290_11084.txt	2
cv307_25270.txt	cv374_25436.txt	2
cv258_5792.txt	cv380_7574.txt	11
cv282_6653.txt	cv380_7574.txt	18
cv343_10368.txt	cv426_10421.txt	1

Figure 3. Computing Intersection of Documents

3. Experimental Evaluation

In this section, we present our experimental results. Our proposed method Best-Join algorithm to efficiently compute pairwise similarity in the corpus has been compared with All-Pairs algorithm.

Our algorithm is compared to All-Pairs to measure its efficiency. We have chosen this algorithm (All-Pairs) because it doesn't resort to approximation or discarding of frequent features. Its contribution is to scale exact method to larger datasets. In other algorithms, the problem is solved approximately by applying either a sketching function based on mini-wise independent permutations in order to compress documents vectors whose dimensions correspond to distinct n-grams or any other probabilistic method.

We measured both the size of the candidate pairs and the running time for all experiment. All-pairs is using cosine similarity while Best-Join use Jaccard similarity. The data set used was downloaded from Reuter's corpus and we have collected other small documents also to check the efficiency and effectiveness of our method on small and average documents. All stop-words were removed and all words in the data set were stemmed.

3.1 Reducing Candidates Pairs

The main drawback of pairwise document similarity computation is the big number of documents to be compared. Documents are represented in w-shingles to be able to highlight the intersection between them which will be the cornerstone to compute their similarity. Reducing the number of shingles involve reducing execution time and memory to be used by the process of similarity computation.

The Table 2 shows how our method reduces the number of shingles to be used in the similarity computation. Here we are comparing with the size produced by All-Pair algorithm using a group of (10, 20, 50, 100, 200, 350, 500, 600, 900 and 1000) documents. The results show that our method performs better than All-Pairs, it reduces the candidate pairs up to 71% in average. Fig 4 shows the trend of reduction of candidates' pairs between All-Pairs and our proposed method.

Table 2. Candidate Size Comparison

n	*All-Pairs*	*Best-Join*	*Reduced Shingles*	*Reduction %*
10	3,917	2,063	1,854	47%
50	18,597	5,421	13,176	71%
100	36,522	12,054	24,468	67%
200	70,507	19,579	50,928	72%
350	121,879	28,461	93,418	77%
500	174,175	39,823	134,352	77%
600	216,195	52,820	163,375	76%
750	276,766	70,798	205,968	74%
900	332,013	84,962	247,051	74%
1,000	383,106	115,499	267,607	70%

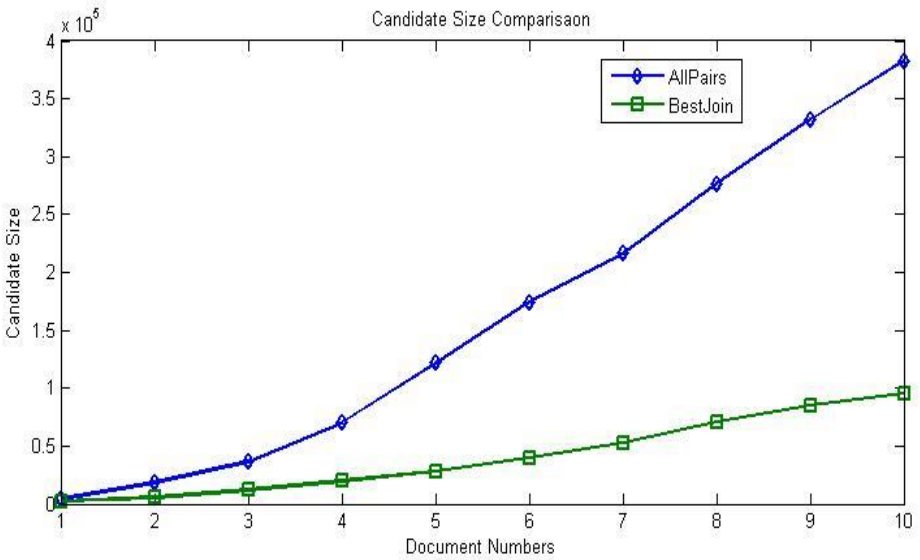


Figure 4. Candidates Pairs Comparison

3.2 Time Complexity Evaluation

As shown before, our method depends on the size of shingle. We should note that, by increasing the length of a shingle, we increase also the space of storage and the number of pairs to be compared which can hinder the efficiency of the method. For our method, the average of eight tokens by shingle was good enough.

The Fig. 5 shows the running time of both algorithms. It's clear that our method outperform the All-Pairs algorithm. One reason is the number of candidates' pairs generated by both algorithms. Another reason, many candidates are quickly discarded in our method than All-Pairs. This reduces dramatically the execution time for our method if compared with All-Pairs.

Two cases are to be considered: comparing a given document to the rest of the documents in the corpus and pair-wise document resemblance, which consist in putting in evidence all documents having a certain resemblance given a threshold.

Dealing with the first case, the time used by this method to perform is O (*n*), where n is the number of shingles in the document. It is the time used for computing the intersection

between the document shingles and the time for counting the shingles shared by each document in the intersection.

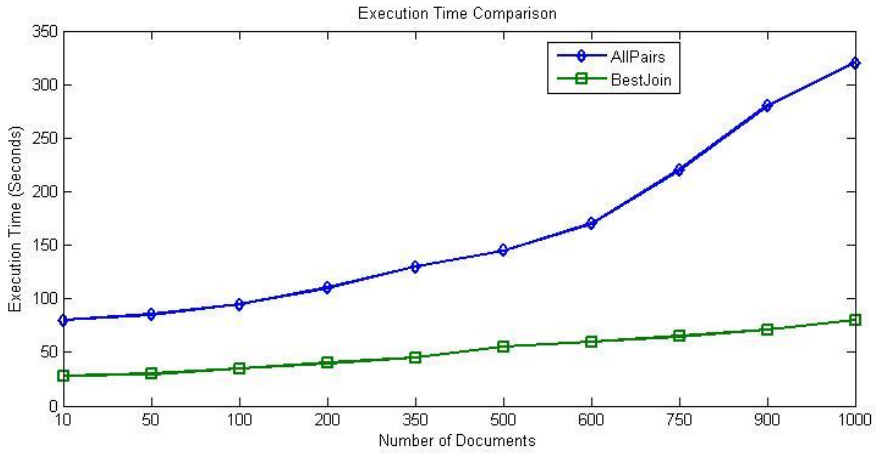


Figure 5. Comparison of Running Time

For the second case, the naïve algorithm uses quadratic time. In our method, we do not need to perform the Cartesian product which is the time and space consuming. The time complexity is O(n log m) where n is the number of documents and m is the universal set of shingles.

As we have explained before, the representation of a corpus as a matrix is space consuming. In our approach, we have adopted the representation for the matrix as a list of tuples. This has an advantage of being indexed and sorted which will dramatically improve the access and search time and save the space used by matrix (by construction a lookup index data structure)

3.3 Effectiveness

We evaluate the effectiveness of our method using three general testing parameters that are commonly used in Information Retrieval (IR). These are Precision, Recall, and F-Measure. Precision is the ratio of number of relevant documents retrieved to the total number of irrelevant and relevant documents. The recall is the ratio of the relevant document retrieved to the total number of the relevant document in the corpus. F-Measure is an effectiveness measure based on recall and precision. It has the advantage of summarizing effectiveness in a single number.

Let A be the number of relevant documents retrieved, B be the number of relevant documents not retrieved and C be the number of irrelevant documents retrieved. The Precision (PR) and Recall (RC) are given by the formula:

$$PR = \frac{A}{A + C} \quad (3)$$

$$RC = \frac{A}{A + B} \quad (4)$$

$$F - Measure = \frac{2 \times RC \times PR}{RC + PR} \quad (5)$$

Our method was tested according to a set of 1000 documents in which 6 pairs of documents were near similar. We have tested by varying the number of w-shingles

because we have seen that it's the number of the length of shingle which affects precision and recall. Table 3 shows the precision, recall and F-Measure ratio of our method. We can notice that with the single size 8, we obtain a good ratio of Precision and Recall.

Table 3. Effectiveness Evaluation by Precision, Recall and F-Measure

Shingle Size	*Precision%*	*Recall%*	*F-Measure%*
4	25	100	40
5	35	83	50
6	40	66	50
7	50	66	57
8	75	50	60
9	100	66	80

4. Conclusion

In this paper, we propose efficient similarity join algorithms by exploiting the ordering of documents and shingles in the records. The algorithm provides efficient solutions for pairwise documents similarity in a corpus. The algorithm alleviates the problem of quadratic growth of candidate pairs when the size of data grows. We have identified that the Cartesian product is the main factor because it implies to consider all document in the corpus and all shingles produced, and we have proposed a method to overcome the issue. Our method uses heuristic techniques to reduce the number of pairs to be compared, and the experimental results indicate that it is possible to reduce the number of candidate shingles to be compared by organizing files in corpus in decreasing order of their size. We have applied our new approaches on 1000 text documents as a sample, and have seen that our approach outperforms other exact search algorithms by reducing the number of shingles to be compared (the average reduction of candidate size in our approach is 71%) thus reducing the computation time.

Our evaluation shows that our method is able to evaluate the similarity of documents in a corpus in O(n log m) time, m being the number of documents. In the future, we will extend our method to incorporate blocking strategies by reducing the candidate pairs for comparison. This can be achieved by representing the sets of documents by their subset, which will reduce the number of shingles and respectively the number of pairs to compute.

Acknowledgements

Project supported by the National Natural Science Foundation of China (Grant No. 61379109, M1321007) and Research Fund for the Doctoral Program of Higher Education of China (Grant No. 20120162110077).

References

[1] C. C. Aggarwal, W. Lin and P. S. Yu, Searching by Corpus with Fingerprints, ACM 348 **(2012)**.
[2] X. Yuan, J. Long, H. Zhang, Z. Zhang and W. Gui, "Optimazing a Near-duplicate Document Detection System with SIMD Technologies", Journal of Computational Information Systems, Binary Inforation Press, 3846 **(2011)**.
[3] S. M. Alzahrani, N. Salim and A. Abraham, Understanding Plagiarism Linguistic Patterns, Textual Features, and Detection Methods, Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions, vol. 42, no.133, **(2011)**.
[4] A. H. Osman, N. Salim, M. S. Binwahlan, R. Alteeb and A. Abuobieda, "An improved plagiarism detection scheme based on semantic role labeling", Applied Soft Computing, Elsivier, **(2012)**.

[5] S. A. Ksouri, M. S. Hidri and K. Barkaoui, "A Parallel Comparator of Documents", Proceedings of the 24th International Workshop on Database and Expert Systems Applications, **(2013)**; Prague, Czech Republic.
[6] H. Chim and X. Deng, Efficient Phrase-Based Document Similarity for Clustering, in: IEEE transaction on knowledge and data engineering, vol.20, no.127, **(2008)**.
[7] R. Mishra and A. Choubey, Discovery of Frequent Patterns from Web Log Data by using FP-Growth algorithm for Web Usage Mining, in: International Journal of Advanced Research in Computer Science and Software Engineering, vol.2, no.311, **(2012)**.
[8] D. Ceglarek, IARIA. Linearithmic Corpus to Corpus Comparison by Sentence Hashing Algorithm SHAPD2, Proceedings of the Fifth International Conference on Advanced Cognitive Technologies and Applications, **(2013)**; Valencia, Spain.
[9] M. Zini, M. Fabri, M. Moneglia and A. Panunza, "Plagiarism detection through multilevel text comparison", Proceedings of the Automated Production of Cross Media Content for Multi-Channel Distribution, **(2006)**; Leeds, UK.
[10] P. Kaur, S. S. Khurmi and G. S. Josan, "Analysis for Classification of Similar Documents among Various Websites using Rapid Miner", **(2014)**, pp.465.
[11] M S. Charikar, ACM, "Similarity estimation techniques from rounding algorithms", Proceedings of the thirty-fourth annual ACM symposium on Theory of computing, (2002); New York, USA.
[12] J. Seo, C.-S. Ock, H.-G. Cho, "A Unified Approach for Computing Document Similarity with Fingerprinting and Alignments", in: Proceedings of the International Conference on Computer and Information Technology, **(2012)** ; Chengdu.
[13] Jannik, Strotgen, M. Gertz and C. Junghans, "An Event-Centric Model for Multilingual Document Similarity", Proceedings of ACM SIGIR, **(2011)**; Beijing, China.
[14] A. Shahbazi and J. Miller, "Extended Subtree: A New Similarity Function for Tree Structured Data", in: IEEE Transactions on knowledge and data engineering, **(2014)**.
[15] Y. Peng and C. Zhang, "Web Information Extraction and its application", Proceedings of IEEE CCIS, **(2011)**.
[16] R. T. Selvi and E. G. D. P. Raj, An approach to Improve Precision and Recall for Ad-hoc Information Retrieval using SBIR Algorithm, in: World Congress on Computing and Communication Technologies, IEEE, **(2014)**, pp.137.
[17] C. Xiao, W. Wang, X. Lin, J. X. yu and G. Wang, "Efficient Similarity Joins for Near Duplicate Detection", in: ACM Transactions on Database Systems, **(2012)**, pp.131-140.
[18] T. Elsayed, J. Lin, D. W. Oard, "Pairwise Document Similarity in Large Collections with MapReduce", in: Proceedings of ACL -08: HLT, **(2008)**; June Columbus, Ohio, USA.
[19] C. Yao, J. Bu, C. Wu and G. Chen, "Semi-supervised spectral hashing for fast similarity", ELSEVIER Neurocomputing, vol.101, no.52, **(2012)**.

