Efficient Near-Duplicate Document Detection Using Consistent Weighted Sampling Filter

Xinpan Yuan[1], Songlin Wang[1], Cheng Peng[1], and Chengyuan Zhang[2]
(Corresponding author: Cheng Peng)
School of Computer Science, Hunan University of Technology[1]
Zhuzhou 412007, China
School of Information Science and Engineering, Central South University, Changsha, China[2]
(Email: 2561838940@qq.com)
(Received Oct. 9, 2018; Revised and Accepted Feb. 7, 2019; First Online July 16, 2019)

Abstract

Near-duplicate document detection is a problem of pursuing data pairs whose similarities are higher than the specific threshold (*e.g.*, 0.7) from the large database. Recently,Consistent Weighted Sampling algorithm (or weighted min-wise hash) and its related hashing algorithms have achieved great performances in near-duplicate detection. However, there are a large number of comparisons for data pairs, which may spend a lot of computation time and affect the performance. This paper proposes a fast consistent weighted sampling filtering algorithm to greatly reduce the calculation time by terminating the unnecessary comparisons in advance. We have proved that the filter is correct and effective through the experiment on the two synthetic data-set (UNIFORM, GAUSSIAN) and a real data (FUNDS).

Keywords: Consistent Weighted Sampling; Filtering; Near-duplicate Document Detection; Similarities

1 Introduction

Explosive information growth of Web leads to a huge amount of similar information on the Web. Research shows that 80% -90% of the data is redundant in backup and archival storage systems, and this rate is still increasing, which is a big waste [24]. These similar documents consumed a lot of storage and computation resources and reduce the efficiency of web search engines [27]. Some duplications come from plagiarism and illegal proliferation. Near-duplicate document detection [16, 22] in intellectual property protection and information retrieval has important applications. The main problem of near-duplicate detection is to find data pairs with similarity greater than the threshold from the large database.In the case that the database is very large or that the similarity computations between the pairs are very costly.

Traditionally, when comparing the similarities of two texts, most of them measure the similarity of texts into the distances from which textual feature vectors are calculated. Common text similarity measurement algorithms have cosine similarity [3], Euclidean distance [13], edit distance [18] and Jaccard coefficient [28], *etc.* These algorithms are only suitable for short texts or when the amount of data is relatively small, and cannot handle long texts and massive text data. In the face of the similarity measure of massive text data, most scholars generate K hash codes or fingerprints from K independent sample outputs, and then estimate the similarity between texts by counting the number of fingerprints equal. Such algorithms are collectively referred to as hash similarity metrics. The most representative is Minwise Hash.

Minwise Hashing [3] (or Minhash) is a Locality Sensitive Hashing, and is considered to be the most popular similarity estimation methods. Many LSH schemes have been proposed, divided between metric-driven [1, 5, 6, 14] with the goal of approximating a given distance metric, and data-driven [19, 25] where the hash functions are learned to optimize performance on a task such as classification.

Minhash keeps a sketch of the data and provides an unbiased estimate of pairwise Jaccard similarity. The algorithm is widely used for near-duplicate web page detection and clustering [9, 12] set similarity measures [2] nearest neighbor search [7] large-scale learning [11] etc. Minwise Hash can quickly and efficiently estimate the similarity of two collections. Minwise Hash requires K (commonly, K=1000) independent random permutations to deal with the datasets [26] It denotes π as a random permutation function: π: $\Omega \rightarrow \Omega$. The similarity between two non-empty sets S_1 and S_2 is defined as:

$$P_r(\min(\Pi(S_1)) = \min(\Pi(S_2)) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = J(S_1, S_2).$$

It can effectively solve the problem of the time and space complexity of solving the similarity of massive data,and the generated feature fingerprints can be used for the next comparison and is widely used However, the

Minwise Hash algorithm does not consider the weight of elements in the set. When a deeper understanding of the words in the text is required, the words should be given corresponding weights. Matching of page titles should be considered more important than matching of other elements in search engines. Besides, the titles, keywords, and abstracts of a document are more important than others. In a sampling algorithm, words or phrases are selected with a low frequency of occurrence in the corpus, because common terms or phrases may represent idiomatic or spurious repetitions.

Manasse [8] introduced the concept of consistent weighted sampling (CWS), which focuses on sampling directly from some well-tailored distribution to avoid any replication. This method, unlike previous ones, could handle real weights exactly. Going a step further, Ioffe [10] proposed weight minhash scheme (WMH) which was able to compute the exact distribution of min-wise sampling leading to a scheme with worst case $O(d)$, where d is the number of non-zeros. Later, Shrivastava [20] provided an exact weighted min-wise hashing with same property as WMH but significantly faster than WMH.

Recent advances based on the idea of densification (Shrivastava & Li, 2014a; c) have shown that it is possible to compute k min-wise hashes, of a vector with d nonzeros, in mere (d; k) computations, a significant improvement over the classical $O(dk)$. These advances have led to an algorithmic improvement in the query complexity of traditional indexing algorithms based on min-wise hashing [21].

To find data pairs with similarities higher than the threshold in the big data set, the usual method of neardu-plicate detection go through the following steps:

1) Feature extraction of data in the dataset;

2) Clustering based on characteristics of the data to form data pairs worth measuring similarity;

3) Computing the similarity value of pairs using the similarity measure function (*e.g.* Minwise hash or CWS).

In this paper, our main contributions include: We design a threshold filter base on CWS and propose a faster weighted hash similarity measurement algorithm, in order to quickly and accurately calculate the similarity in large-scale datasets.

The rest of the paper is organized as follows: Section 2 discusses the related works. Section 3 describes the weighted sampling algorithm in detail. Section 4 introduces a faster similarity measurement over threshold. Experimental evaluations are presented in Section 5. Finally, Section 6 gives conclusions.

2 Feature Representation

TF-IDF is a commonly used text weighting technique for information retrieval and data mining. TF indicates the frequency with which feature item appears in document. IDF [15] represents the quantification of the distribution of feature items in the document set. TF is term frequency and IDF is inverse document frequency.

TF-IDF is a statistical method for assessing the importance of a word for a document set or one of the documents in a corpus [17]. The importance of a word increases proportionally with the number of times it appears in the file, but at the same time it decreases inversely with the frequency with which it appears in the corpus. The main idea is that if a word or phrase has a high frequency of TF in an article and rarely appears in other articles, the less the document contains the feature item and the larger the IDF, the word or phrase has a good classification ability.

For the term t_i appearing in the document d_j, its word frequency TF can be expressed as:

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}},$$

where $n_{i,j}$ is the number of occurrences of the word in the file, and the denominator is the sum of the occurrences of all the words in the document. The reverse document frequency IDF [23] can be expressed as:

$$IDF_i = \log \frac{|D|}{|j : i \in d_j| + 1}.$$

Where $|D|$ represents the DBLP: `conf/bigdataconf/LuoNH13` total number of files in the corpus. $|j : i \in d_j|$ indicates the number of documents containing the word t_i. So the TF-IDF of term t_i can be expressed as: TF-IDF=TF×IDF, Therefore, a document can be represented by the weighted set S={TF-IDF$_1$, TFIDF$_2$, $\cdots$, TF-IDF$_n$} after the above processing.

3 Document Clustering

The main role of document clustering is to form pairs of documents that should be matched in document set. Then, the similarity estimation is performed on the formed document pairs. The process consists of three steps [4] as shown in Figure 1.

Step 1. Calculating a sketch for every document. Generate a list of all the shingles and the documents they appear in, sorted by shingles value.The sketch is expanded into a list of ⟨shingles value, document ID⟩ pairs. The list is then sorted using the split, sort, and merge method.

Step 2. Generating a list of all the pairs of documents that share any shingles, along with the number of shingles they have in common. To do this, taking the file of sorted ⟨shingle, ID⟩ pairs and expand it into a list of ⟨ID, ID, count of common shingles⟩ triplets by taking each shingle that appears in multiple documents and generating the complete set of ⟨ID, ID, 1⟩ triplets for that shingle. Then, applying the divide, sort, merge procedure (adding the counts for

matching ID - ID pairs) to produce a single file of all triplets sorted by the first document ID.

Step3. Examining each ⟨ID, ID, count⟩ triplet and decide if the document pair exceeds our threshold for resemblance. If it does, making clusters of the document pair.

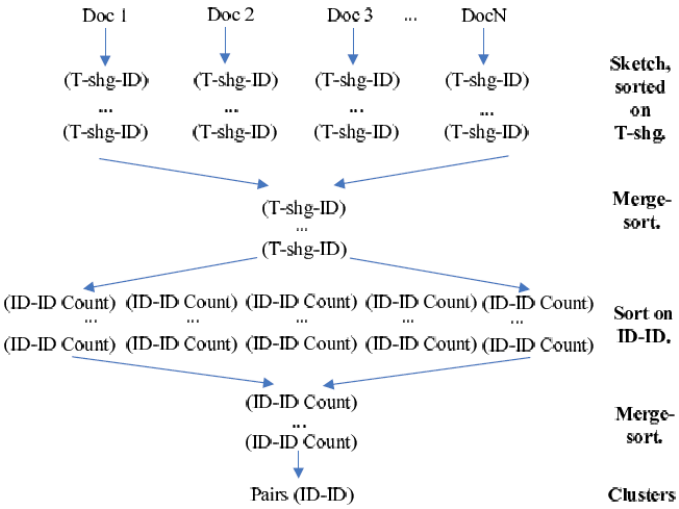


Figure 1: The cluster process

4 Weighted Sampling Algorithm

The consistent weighted sampling (CWS) [8] is a sampling scheme, sampling representatives from a weighted set such that for any non-empty weighted sets S_1 and S_2, the probability that the two choose the same sample is equal to the Jaccard similarity:

$$R = Pr[sample(S) = sample(T)] = \frac{\sum_x min(S(x), T(x))}{\sum_x max(S(x), T(x))} \quad (1)$$

Where $sample(S)$ is a representative sample value of pair (x, y) with y satisfies $0<y\leqslant S(x)$, where $S(x)$ is the weight of any element x.

When x is selected to the sample output pair, the probability of x selection is proportional to $S(x)$. and y is new generated value of $sample()$, is uniformly distributed between 0 and $S(x)$, and finally only the value of y plays a decisive similarity role, as the sample of the weightings S. As shown in Algorithm 1 is the sampling algorithm steps.

As shown in Figure 2, by sampling K from the set S, we can get $Sample(S) = \{p_1, p_2, \cdots, p_k\}$, where p_i is i-th sampled value of set S by Sequence i.

Given two weighted sets S_1 and S_2, perform K times independent samplings to generate values: $Sample(S_1)=p_{1,1}$, $\{p_{1,2}, \cdots, p_{1,k}\}$ and $Sample(S_2)=\{p_{2,1}, p_{2,2}, \cdots, p_{2,k}\}$. the CWS similarity estimator:

$$R = \frac{1}{K}\sum_{i}^{K} 1\{p_{1,i} == p_{2,j}\}. \quad (2)$$

Algorithm 1 Sampling algorithm

1: Input: Given a non-empty weighted set S, $S(x)$ is the weight of any element x
2: Output: $pair(x, y)$
3: **Step1:** GenerateSampleSequences$(x, k, salt)$
4: (1) Generate a number of points on the interval(0, ∞), each point is $2k, k \in (0, \infty)$;
5: (2) Take a k such that $x \in (2^{k-1}, 2^k]$;
6: (3) Generate a random number random within the range of [0, 1] and calculate sample $= 2^k \divideontimes random$;
7: **while** sample $> 2^{k-1}$: **do**
8: save $sample$;
9: $sample = sample \divideontimes random$;
10: **end while**
11: **Step2:** Active indices: (y, z) = **ActiveIndices** $(x, S(x), salt)$
12: According to the sequences obtained in Step 1, determine the greatest value $y \leq S(x)$ and the least value $z > S(x)$ in expected constant time. That is, z=the first right value of the sequence greater than $S(x)$; y=the first left value of the sequence less than $S(x)$.
13: **Step3:** Uniform sampling
14: (1) Define $h_{max} = 0$, $x_{max} = null$, $y_{max} = 0$;
15: (2) for each x in $S(x)$:
16: a. Random variable β is within the range of [0, 1];
17: b. $cdf_z(a) = a^z + a^z z \ln(1/a) = \beta$;
18: c. $h = cdf^{-1}(\beta)$.
19: (3) Take the value x and y with the maximum h_{max} of all h, and return $(x_{max}y_{max})$.
20: After the above three steps, the final pair value of (x_{max}, y_{max}) is the first sampled value of set S by $Sequence_1$.
21: End

5 Faster Similarity Measurement Over Threshold

Most practical applications scenes such as near-duplicate detection, clustering, and nearest neighbor search, only care about the similarity is greater than a certain high threshold [30] $T0$ (*e.g.* $T0$=0.8) between pairwise data, as shown in Figure 3. In this application scenario, in order to quickly and accurately obtain the similar data over the specific threshold, we design a filter of CWS to terminate the unnecessary matching process in advance.

5.1 Observation Threshold

At first, the filter divides the entire comparison process of for each $p_{1,i} = p_{2,i}$ in Equation (2) into several subprocesses with observation points $\{k_1, k_2, \cdots, k_j, \cdots, k_n\}$, $(0 < k_i \leqslant K, 0 < i \leqslant n)$. The similarity of Equation (2)

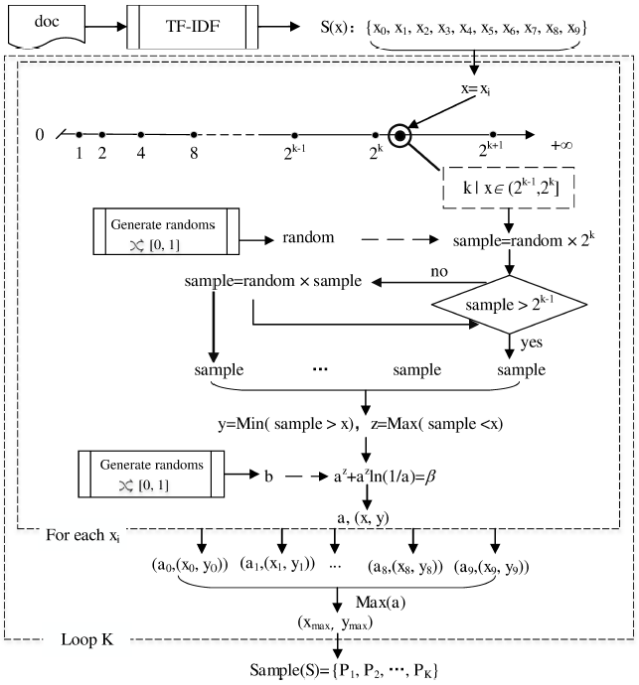


Figure 2: The sampling process

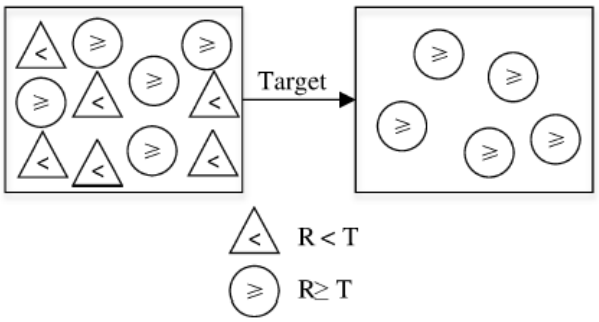


Figure 3: A filter with similarity $R \geq T$

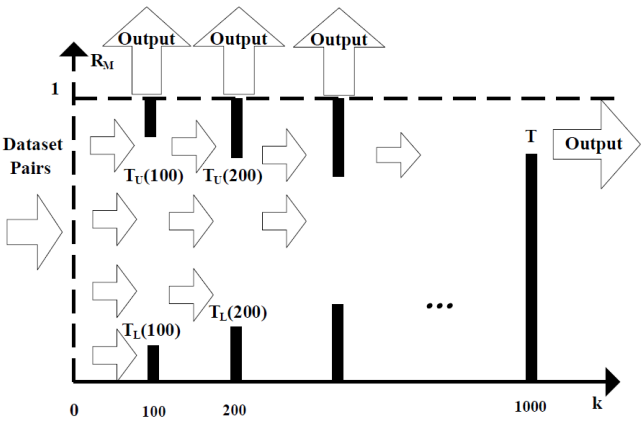


Figure 4: Schematic diagram of filtering process

at the j-th observation point is different from the similarity with argument K, because the more sampling times, the accuracy of similarity is higher. Therefore, in the case of a small k_j, the similarity of the calculations is meaningless. However, We can set the threshold to be higher than the threshold T when the number of samples is K, and set another threshold to filter out the data pairs that do not need to be compared in advance.

As shown in Figure 4, the filter defines the lower bound threshold $T_L(k)$ and the upper bound threshold $T_U(k)$ for any k observation, then the filter can output the pairwise data that exceeds the real similarity over T, when the point of observation at k is higher than $T_U(k)$. And in advance filtering the pairwise data, when the point of observation at k is less than $T_L(k)$, thereby speeding up the comparison process. The $T_U(k)$ and $T_L(k)$ can be found by hypothesis testing and small probability events.

5.2 How to Setting Threshold At K Observation

The total number of comparisons K is reduced to k, and the random variable X at the observation point k is equal to the number of equal terms of $Sample(S_1)$=$\{p_{1,1}, p_{1,2}, \cdots, p_{1,k}\}$ and $Sample(S_2)$=$\{p_{2,1}, p_{2,2}, \cdots, p_{2,k}\}$. The definition of X is shown in Equation (3).

$$X = \sum_{i}^{k} 1\{p_{1,i} = p_{2,j}\} \tag{3}$$

Obviously, X obeys the binomial distribution $X \sim B(n, R)$; the probability function $\Pr(X=m)$ of random variable X is:

$$Pr(X = m) = \binom{k}{i} R^m (1-R)^{K-m} \tag{4}$$

The probability $\Pr(X \leqslant m)$ is: $Pr(X \leq m) = \binom{k}{1} R^1 (1-R)^{k-1} + \binom{k}{2} R^2 (1-R)^{k-2} + \cdots + \binom{k}{i} R^i (1-R)^{k-i} + \cdots + \binom{k}{m} R^m (1-R)^{k-m}$.

Then $\Pr(X \leqslant m)$ and $\Pr(X > m)$ is:

$$Pr(X \leqslant m) = \sum_{i=0}^{m} \binom{k}{i} R^i (1-R)^{k-i}$$

$$Pr(X > m) = \sum_{i=m+1}^{k} \binom{k}{i} R^i (1-R)^{k-i}$$

First make assumptions and use appropriate statistical methods to determine the probability of hypothesis. If the probability is high, although the assumption is not necessarily correct, if the possibility is small, the assumption is absolutely wrong.

The threshold is expressed as T and small probability is expressed as e. The method of test is as following:

1) Hypothesis.

Hypothesis $\mathcal{H}1$: The similarity R is greater than threshold T.

Hypothesis $\mathcal{H}2$: The similarity R is equal or less than threshold T.

The obvious is the $\mathcal{H}\mathbf{1} = \neg\mathcal{H}\mathbf{2}$, and $\neg\mathcal{H}\mathbf{1} = \mathcal{H}\mathbf{2}$.

2) Test hypothesis $\mathcal{H}1$: Test if the probability $\Pr(X \leqslant m)$ of variable X is small enough to be called a small probability event e? If

$$\sum_{i=0}^{m} \binom{k}{i} T^i (1-T)^{k-i} = e$$

Then the hypothesis $\mathcal{H}\mathbf{1}$ is wrong, that is $\neg\mathcal{H}\mathbf{1}$ is correct, and $\mathcal{H}\mathbf{2}$ is correct.

The same reason can be tested hypothesis $\mathcal{H}\mathbf{2}$. If

$$\sum_{i=m+1}^{k} \binom{k}{i} T^i (1-T)^{k-i} = e$$

Then the hypothesis $\mathcal{H}\mathbf{2}$ is wrong, that is $\neg\mathcal{H}\mathbf{2}$ is correct,and $\mathcal{H}\mathbf{1}$ is correct.

But how to setting threshold at k observation?

Let the total comparison number K=1000 (unrelated parameter), the observation comparison number k=100, the small probability e=1E-5, the threshold T=0.8. there are $\Pr(X \leqslant m)$ and $\Pr(X > m)$ as shown in Table 1 and Figure 5.

Table 1: Probability distribution of $\{X \leq m\}$ and $\{X > m\}$when $T = 0.8, k = 100$

m	$P_r\{X \leqslant m\}$	$P_r\{X > m\}$	m	$P_r\{X \leqslant m\}$	$P_r\{X > m\}$
5	$1.29E-61$	1	55	$4.22E-09$	1
10	$6.48E-53$	1	60	$1.29E-06$	0.999999
15	$1.57E-45$	1	65	0.000147	0.999853
20	$4.89E-39$	1	70	0.006059	0.993941
25	$3.09E-33$	1	75	0.087475	0.912525
30	$5.04E-28$	1	80	0.440538	0.559462
35	$2.49E-23$	1	85	0.871494	0.128506
40	$4.09E-19$	1	90	0.994304	0.005696
45	$2.4E-15$	1	95	0.999981	$1.87E-05$
50	$5.18E-12$	1	100	1	$2.04E-10$

As shown in Table 1, we can observe that m=60 and m=95 are located between a small probability and not a small probability, so we can define m=95 as m_{uper} and m=60 as m_{lower}. According to the small probability e=1E-5, if there is R>95% at the observation point of k=100, then we can completely determine R>80%. The same reason, if there is R<60% at the observation point k, then we can completely determine R<80%, so so we can filter out the unnecessary comparison process in advance.

$$\begin{aligned} T_U(k) &= m_{uper}/k \\ T_L(k) &= m_{lower}/k \end{aligned}$$

We try to give a formal description from the perspective of conditional probability. Let event $A = \{R_K > T\}$, $B = \{R_k > T_U(k)\} = \{X > m_{uper}\}$, our goal is $Pr(A|B) = \frac{Pr(A|B)}{Pr(B)=1}$. If $\Pr(B)\to 0 \Rightarrow \Pr(A|B)=1$. The same reason, let event $C=\{R_K<T\}$, $D=\{R_k<T_L(k)\}=\{X<m_{lower}\}$. If $\Pr(D)\to 0 \Rightarrow \Pr(C|D)=1$.

With threshold T and observation points$\{k_1, k_2, \cdots, k_j, \cdots, k_{n-1}, k_n\}$, $(0 < k_i \leqslant K, 0 < i \leqslant n)$, there is lots of $T_U(k)$ and $T_L(k)$ to be settled into the filter, as shown as Figure 6.

The calculation process of the consistent weighted sampling filtering algorithm proposed in the paper is shown in Algorithm 2.

Algorithm 2 The calculation process of the weighted sampling filtering algorithm

Input: Weighted set $\{(S_1, S_2), (S_3, S_4), \cdots, (S_{2n-1}, S_{2n})\}$
Output: The similarity R of the set pairs is greater than the threshold T $\{(S_{2i-1}, S_{2i}), (S_{2i-1}, S_{2i}) | R(S_{2i-1}, S_{2i}), (1 \leqslant i \leqslant n)\}$
(1) Setting parameters Similarity threshold T, observation point $\{k_1, k_2, \cdots, k_i, \cdots, k_{n-1}, k_n\}$, Number of samples K, Small probability e;
(2) Using the CWS algorithm to generate corresponding fingerprints, for example Sample$(S_1) = \{p_{1,1}, p_{1,2}, \cdots, p_{1,K}\}$ and Sample$(S_2) = \{p_{2,1}, p_{2,2}, \cdots, p_{2,K}\}$;
(3) At the observation point $k = k_i$, Calculate the upper and lower thresholds $T_U(k) = m_{uper}/k, T_L(k) = m_{lower}/k$;
(4) For each fingerprint pair, compare the first k fingerprints, calculate the similarity $R(k)$ at the observation point;
if $R(k) \leqslant T_L(k)$ **then**
    The fingerprint pair is excluded in advance, and the corresponding weighted set pair;
else if $R(k) \geqslant T_U(k)$ **then**
    Output fingerprint pairs in advance, and corresponding weighted set pairs;
else
    $i++; k = k_i$; Go to (3) until $k_i = K$
end if

6 Experimental Evaluation

6.1 Experimental Dataset

We select the data pair set from the FUNDS set as the source data set. **FUNDS:** Text application for the Natural Science Foundation project, approximately 100,000 documents. Considering that real-world data sets are usually distributed between Gaussian and uniform distribution, in order to make the experimental results more reliable, we synthesize datasets of uniform distribution and Gaussian distribution from the FUNDS dataset. Finally we synthesize Gaussian, uniform and FUNDS datasets,

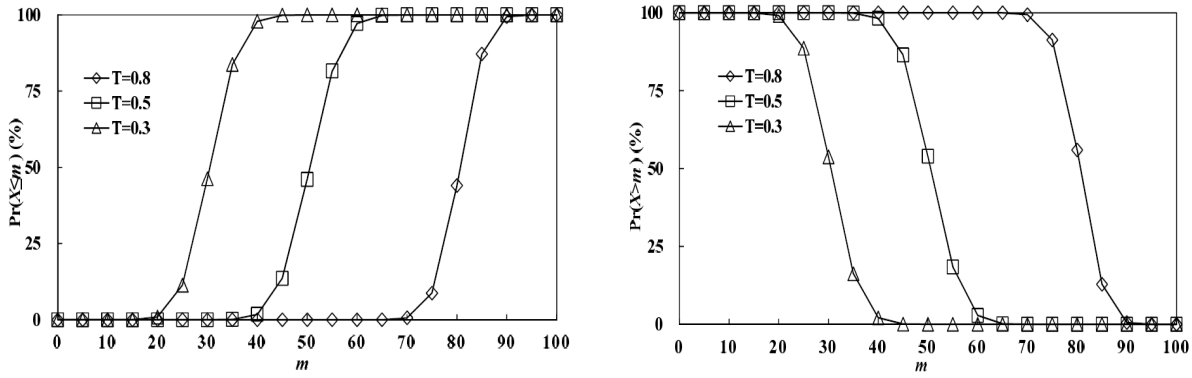


Figure 5: (a) Pr($X>m$) and (b) Pr($X\leqslant m$) varies on m

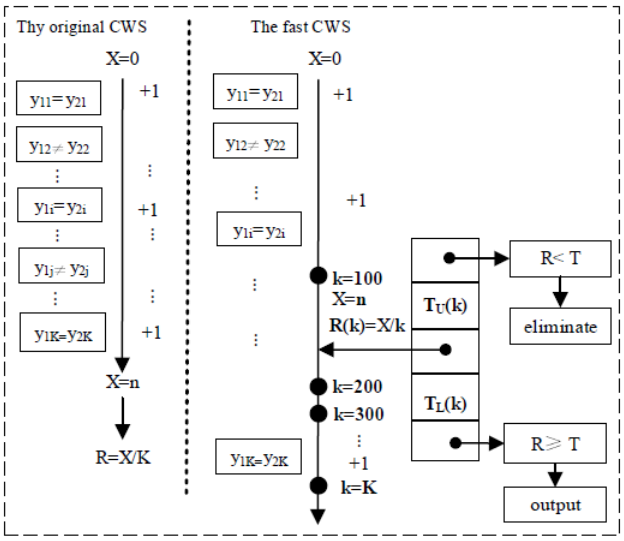


Figure 6: The fast weighted hash similarity measure filter

each with 1000 data pairs. Finally we synthesize three data sets, each with 1000 data pairs, respectively:

1) **UNIFORM** distribution dataset;
2) **GAUSSIAN** distribution dataset;
3) **FUNDS** dataset. The similarity distribution of the data set is shown in Figure 7.

6.2 Precision and Recall of Filter

According to the small probability theory, the filter does not change the estimation accuracy of the CWS algorithm. By Looking at the possibility of testing correctly filtering through actual data, the accuracy and recall of $T_U(k)$ filter is:

$$Accuracy(T_U(k)) = \frac{|R_k > T_U(k)| \bigcap |R_k > T|}{|R_k > T_U(k)|}$$

$$Recall(T_U(k)) = \frac{|R_k > T_U(k)| \bigcap |R_k > T|}{|R_k > T|}$$

Where T is the similarity threshold, K is the number of samples, R_k is the similarity at the observation point k (*e.g.* k=100), R_K is the similarity estimate of CWS (*e.g.* K=1000). And the $T_L(k)$ filtering accuracy and recall rate:

$$Accuracy(T_L(k)) = \frac{|R_k < T_L(k)| \bigcap |R_k < T|}{|R_k < T_L(k)|}$$

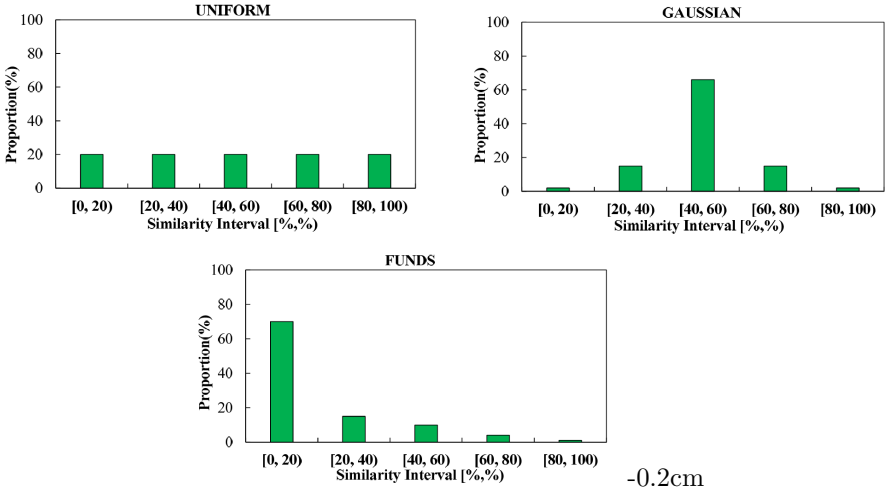
$$Recall(T_L(k)) = \frac{|R_k < T_L(k)| \bigcap |R_k < T|}{|R_k < T|}$$

As shown in the Figure 8, let the observation point k=100, the threshold T=0.8, 0.5, and 0.3, and we have following discussion.

1) Both $T_U(k)$ and $T_L(k)$ have a dividing point of accuracy, for example, when T=0.8 the accuracy of $T_U(k)$ is low in the interval [0,90], in [90,100] will certainly filter success, accuracy is 100%. The dividing point is also the dividing point between non-small probability and small probability.

2) For UNIFORM and GAUSSIAN dataset, when the accuracy is 100%, the recall is not high. It means that filtering accuracy is high, but there are still omissions, so we need to continue setting thresholds at subsequent observation points (*e.g.* k=200, 300 and so on). However, it is easy to observe that certain $T_L(k)$ values in the actual data set (FUNDS) can achieve high accuracy and recall rates because there are a large number of low similarity data in the FUNDS.In this way, a large number of data can be filtered out at the first threshold, thus reducing the time of comparison.

3) So how do we set the threshold at the observation point? As shown in the Figure 8, the setting of the observation point threshold is not related to the distribution of the data set, and determined by the principle of small probability. It can be clearly seen from any picture that the thresholds are the same. The dividing point is the best position to set the threshold.



-0.2cm

Figure 7: The similarity distribution of datasets

On the one hand, it guarantees 100% accuracy and on the other hand increases the recall rate as much as possible.

6.3 Filter Rate

As show in Figure 9, for different data distributions, there is no threshold to ensure 100% accuracy and recall rate, however, the remaining data can be filtered at subsequent observation points (for example, k = 200, 400, 600, *etc.*), and the overall recall rate will be 100%. let's look at the amount of filtering, described in terms of the proportion of the filtered data to the total data. And the filter rates of $T_U(k)$ and $T_L(k)$ are:

$$\begin{aligned} FilteringRate(T_U(k)) &= \frac{|R_k > T_U(k)|}{N} \\ FilteringRate(T_L(k)) &= \frac{|R_k < T_L(k)|}{N} \end{aligned}$$

Where N is the total number of pairs.

Let the observation point k=100,200,400,600,800, the threshold T=0.8, 0.5, and 0.3. As shown in the Figure 9, Different data sets have different amounts of filtering under the influence of $T_U(k)$ or $T_L(k)$. For uniform and Gaussian data sets, $T_L(k)$ plays a major role in filtering when the threshold is high ($T = 0.8$). When the threshold is low (T=0.3), $T_U(k)$ plays a major filtering role. But for datasets (such as FUNDS) that are mostly low similarity data, when the threshold is $T = 0.8$, $T_L(k)$ can filter out more than 90% of the data at the observation point k = 100. Even at the threshold of $T = 0.3$, it can filter out about 50% of the data.

In most practical applications, only high similar data is concerned, that is, a large threshold is set, but there is a large amount of low similarity data in the data set, so that the filter can play a greater role.

6.4 Time Cost

Based on the above analysis of the filtration rate, we can expect that the calculation time will be greatly reduced. CWS has to complete the comparison of 100 million data pairs (each CWS sampling K=1000), so it consumes almost the 1011 ($100\times106\times K$) comparisons in total.

As shown in the Figure 10, abscissa describes different distributed data sets, and the ordinate is CPU time. We still choose three typical thresholds of T=0.8, 0.5, and 0.3 to measure time. The time consumed by different thresholds for the same data set is different, depending on the filtering rate. At the same time, different dataset distributions will produce different filtering rates, for example, it can be clearly observed that the FUNDS data set has a high filtering rate, so its calculation consumes the least amount of time. **Figure 10: CPU time cost of three distribution dataset**

Experiments show that the algorithm can guarantee 100% accuracy and increase the recall rate as much as possible. The remaining filtered data can be used at subsequent observation points (for example, k = 200, 400, 600, *etc.*), and the overall recall rate is 100%. For a large number of low similarity real data, accompanied by high threshold queries, the filter reduces the 85% of comparison, compared with the original CWS algorithm.

6.5 Application

Known from the beginning of January 2018, the document's number of the National Natural Resources Fund of China is about 1.2 million. The pairs number of clustering was about 100 billion to check the similarity. According to the consistent weight sampling algorithm (do not use the optimization algorithm proposed in this paper), it takes about 5000s to complete the near-duplication detection of 100 billion pairs. However, the time taken by the CWS Filter algorithm proposed in this paper is reduced to about 280s. This will greatly reduce the time

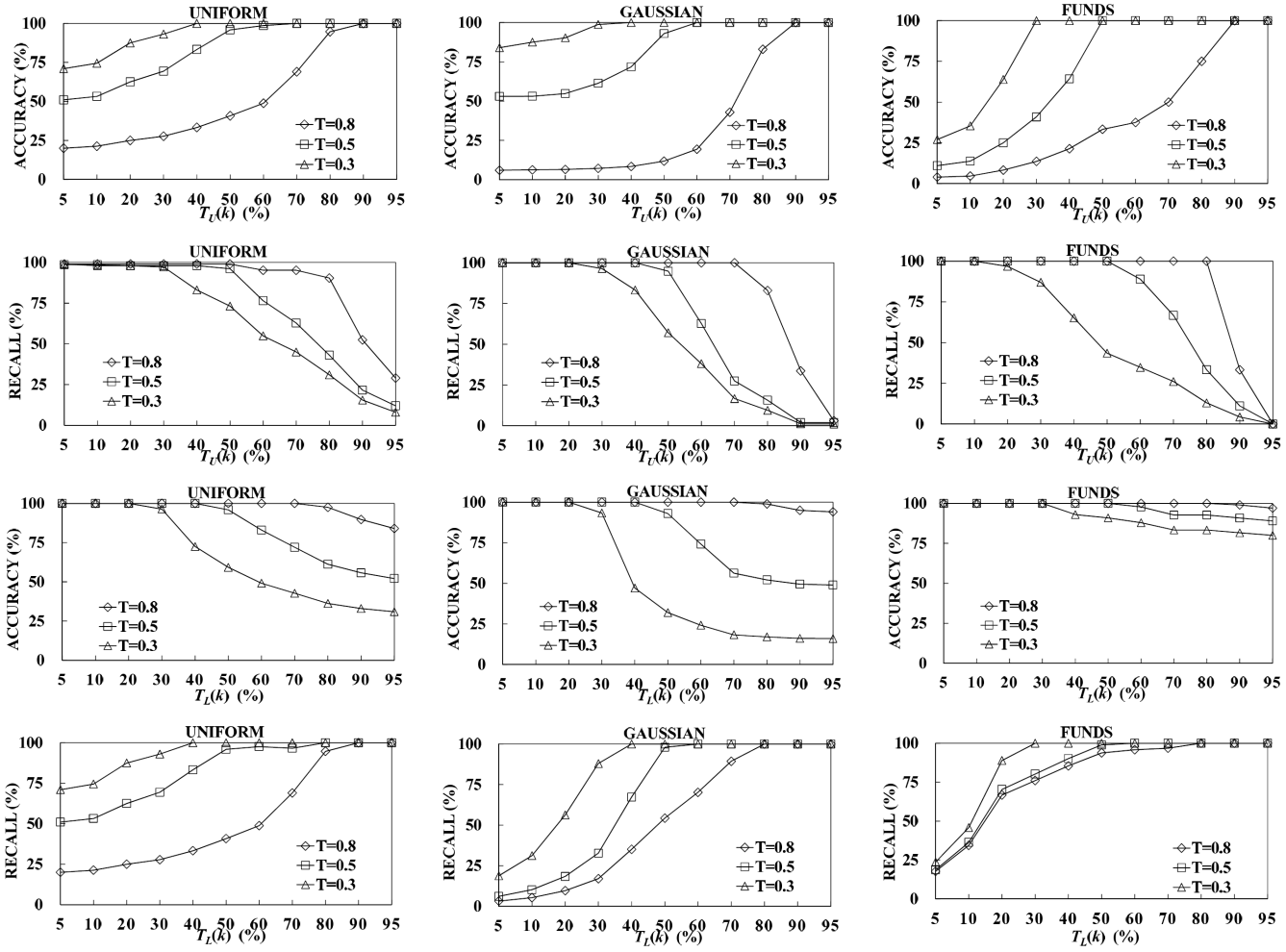


Figure 8: Accuracy and recall of three distribution dataset

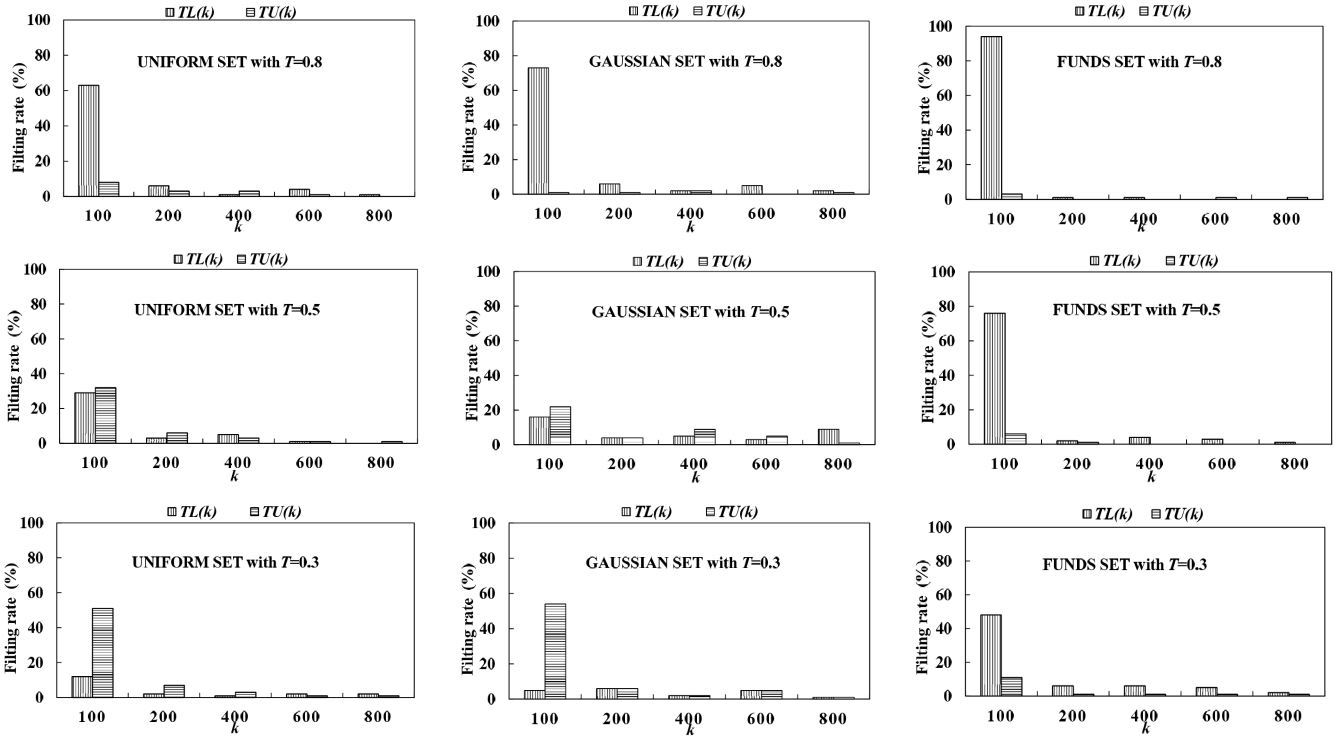


Figure 9: Filtering rate of three distribution dataset

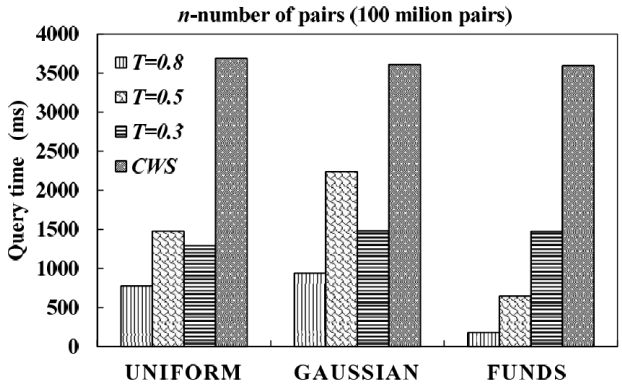


Figure 10: CPU time cost of three distribution dataset

consumption compared to the previous calculations.

7 Conclusions

In this paper, we combine binomial distribution with small probability event and propose a fast consistent weighted hash similarity measurement over threshold. It greatly reduces the calculation time by terminating the unnecessary comparison in advance. Our experimental results are based on the two synthetic dataset (UNIFORM, GAUSSIAN) and a real data(FUNDS), which proves that the filter is effective and correct.

Acknowledgments

This research was funded by National Natural Science Foundation of China [61402165, 61871432,61702560], the Key R & D programs of Hunan Province (No.2016JC2018) and Natural Science Foundation of Hunan Province [2018JJ2099,2018JJ3691].

References

[1] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Communications of the ACM*, vol. 51, no. 1, pp. 117–122, 2008.

[2] R. J. Bayardo, Y. M. Ma, and R. Srikant, "Scaling up all pairs similarity search," in *Proceedings of the 16th International Conference on World Wide Web (WWW'07)*, pp. 131–140, 2007.

[3] T. Bohman, C. Cooper, and A. M. Frieze, "Min-wise independent linear permutations," *The Electronic Journal of Combinatorics*, vol. 7, 2000. (`https://www.combinatorics.org/ojs/index.php/eljc/article/view/v7i1r26`)

[4] D. Brodic, A. Amelio, and Z. N. Milivojevic, "Clustering documents in evolving languages by image texture analysis," *Applied Intelligence*, vol. 46, no. 4, pp. 916–933, 2017.

[5] M. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings on 34th Annual ACM Symposium on Theory of Computing*, pp. 380–388, 2002.

[6] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the 20th ACM Symposium on Computational Geometry*, pp. 253–262, 2004.

[7] X. K. Feng, J. T. Cui, Y. F. Liu, and H. Li, "Effective optimizations of cluster-based nearest neighbor search in high-dimensional space," *Multimedia Systems*, vol. 23, no. 1, pp. 139–153, 2017.

[8] B. Haeupler, M. S. Manasse, and K. Talwar, "Consistent weighted sampling made fast, small, and easy," *CoRR*, vol. abs/1410.4266, 2014.

[9] M. R. Henzinger, "Finding near-duplicate web pages: A large-scale evaluation of algorithms," in *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 284–291, 2006.

[10] S. Ioffe, "Improved consistent sampling, weighted minhash and L1 sketching," in *The 10th IEEE International Conference on Data Mining*, pp. 246–255, 2010.

[11] B. Joshi, F. Iutzeler, and M. R. Amini, "Large-scale asynchronous distributed learning based on parameter exchanges," *International Journal Data Science and Analytics*, vol. 5, no. 4, pp. 223–232, 2018.

[12] J. P. Kumar and P. Govindarajulu, "Near-duplicate web page detection: An efficient approach using clustering, sentence feature and fingerprinting," *International Journal of Computational Intelligence Systems*, vol. 6, no. 1, pp. 1–13, 2013.

[13] L. H. Lee, C. H. Wan, R. Rajkumar, and D. Isa, "An enhanced support vector machine classification framework by using euclidean distance function for text document categorization," *Applied Intelligence*, vol. 37, no. 1, pp. 80–99, 2012.

[14] P. Li, A. Christian, K. Ning, and W. H. Gui, "B-bit minwise hashing for estimating three-way similarities," in *Advances in Neural Information Processing Systems 23*, pp. 1387–1395, 2010.

[15] J. Long, Q. F. Liu, X. P. Yuan, C. Y. Zhang, and J. F. Liu, "A filter of minhash for image similarity measures," *JACIII*, vol. 22, no. 5, pp. 689–698, 2018.

[16] X. Luo, W. A. Najjar, and V. Hristidis, "Efficient near-duplicate document detection using fpgas," in *Proceedings of the IEEE International Conference on Big Data*, pp. 54–61, 2013.

[17] P. D. Qin, W. R. Xu, and J. Guo, "A novel negative sampling based on TFIDF for learning word representation," *Neurocomputing*, vol. 177, pp. 257–265, 2016.

[18] D. C. Reis, P. B. Golgher, A. S. Silva, and H. F. Laender, "Automatic web news extraction using tree edit distance," in *Proceedings of the 13th International Conference on World Wide Web (WWW'04)*, pp. 502–511, 2004.

[19] R. Salakhutdinov and G. E. Hinton, "Semantic hashing," *International Journal of Approximate Reasoning*, vol. 50, no. 7, pp. 969–978, 2009.
[20] A. Shrivastava, "Exact weighted minwise hashing in constant time," *CoRR*, vol. abs/1602.08393, 2016.
[21] A. Shrivastava, "Optimal densification for fast and accurate minwise hashing," in *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, pp. 3154–3163, 2017.
[22] J. H. Wang and H. C. Chang, "Exploiting sentence-level features for near-duplicate document detection," in *Proceedings of 5th Asia Information Retrieval Symposium on Information Retrieval Technology, (AIRS'09)*, pp. 205–217, 2009.
[23] L. Wu, Y. Wang, J. B. Gao, and X. Li, "Deep adaptive feature embedding with local sample distributions for person re-identification," *Pattern Recognition*, vol. 73, pp. 275–288, 2018.
[24] Z. Y. Wang, Y. Lu, and G. Z. Sun, "A policy-based de-duplication mechanism for securing cloud storage," *International Journal of Electronics and Information Engineering*, vol. 2, no. 2, pp. 70–79, 2015.
[25] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Advances in Neural Information Processing Systems 21*, pp. 1753–1760, 2008.
[26] X. P. Yuan, J. Long, Z. P. Zhang, Y. Y. Luo, H. Zhang, and W. H. Gui, "F-fractional bit minwise hashing," *Journal of Software*, vol. 7, no. 1, pp. 228–236, 2012.
[27] C. Y. Zhang, Y. Zhang, W. J. Zhang, and X. I. Lin, "Inverted linear quadtree: efficient top K spatial keyword search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 7, pp. 1706–1721, 2016.
[28] K. Zheng, H. Su, B. L. Zheng, S. Shang, J. J. Xu, J. J. Liu, and X. F. Zhou, "Interactive top-k spatial keyword queries," in *IEEE of 31st International Conference on Data Engineering (ICDE'15)*, pp. 423–434, 2015.

Biography

Xinpan Yuan. He was born in Hunan Province, China. He received Ph.D. degree in computer science from the Central South University, in 2012. Currently, he is a lecturer in School of Computer Science of Hunan University of Technology, China. His research interests include Information Retrieval, Data Mining and NLP.

Songlin Wang. He was born in Henan Province, China. He received B.S. degree in computer science from Hunan University of Technology, in 2017. Currently, he is a master student in School of Computer Science of Hunan University of Technology, China. His current research interests include Information Retrieval and NLP.

Cheng Peng. He was born in Hunan Province, China. He received Ph.D. degree in computer science from the Central South University, in 2013. Currently, he is a lecturer in School of Computer Science of Hunan University of Technology, China. His research interests include software and Big Data.

Chengyuan Zhang. He was born in Hunan Province, China. He received PhD degree in computer science from the University of New South Wales. Currently, he is a lecturer in School of Information Science and Engineering of Central South University, China. His main research interests include information retrieval, query processing on spatial data and multimedia data.