Near Duplicate Text Detection Using Frequency-Biased Signatures

Yifang Sun, Jianbin Qin, and Wei Wang*

The University of New South Wales, Australia
{yifangs,jqin,weiw}@cse.unsw.edu.au

Abstract. As the use of electronic documents are becoming more popular, people want to find documents completely or partially duplicate. In this paper, we propose a near duplicate text detection framework using signatures to save space and query time. We also propose a novel signature selection algorithm which uses collection frequency of q-grams. We compare our algorithm with Winnowing, which is one of the state-of-the-art signature selection algorithms. We show that our algorithm acquires much better accuracy with less time and space cost. We perform extensive experiments to verify our conclusion.

Keywords: near duplicate text detection, Winnowing, k-stability, collection frequency.

1 Introduction

It is now a common practice to use electronic documents in business communications (e.g., Web pages, word documents) and personal life (e.g., emails), as these digital documents are easy and cost-effective to store, retrieve and share. Given a collection of such documents, it is often needed to find documents that are nearly duplicate from a given query document either *completely* or *partially*. We call this the *near duplicate text detection* problem, and it has wide applications such as copyright enforcement, plagiarism detection, and version control.

To scale to large collection of document, the prevalent method for near duplicate text detection is based on *signatures*: a set of signatures are extracted and indexed for the documents at indexing time, and at query time, the query document's signatures are produced in the same manner; this generates a set of candidate texts which will be finally compared with the query document. However, many of the existing methods, such as $\pmod p = 0$ scheme [13], local maximum [2], spotSig [19] and I-match [7], are based on heuristics that cannot even guarantee 100% detection of exact copies.

In this paper, we first propose a general framework for the problem based on the Winnowing-family algorithms [17], which have the *locality* property that *exact* copies exceeding a certain length are guaranteed to be detected. In order to quantify the ability to detect *near* duplicate copies (i.e., copying with a small

* This work is partially supported by ARC DP Projects DP130103401 and DP130103405.

X. Lin et al. (Eds.): WISE 2013, Part I, LNCS 8180, pp. 277–291, 2013.

amount of errors). we propose a novel and useful concept, k-stability. We compute the k-stability for all Winnowing-family algorithms and the result reveals that the original Winnowing algorithm trades quality (i.e., recall) for better efficiency. We then proposed a simple yet effective variation of the Winnowing algorithm, named frequency biased Winnowing, which achieves both good efficiency and high quality. We also consider candidate text generation methods as well as optimizations to further reduce the number of similarity computations. We experimentally evaluated our method in a plagarism detection benchmark, and our method is shown to achieve higher recall with the superior time and indexing space efficiency than the method based on the original Winnowing.

The rest of the paper is organized as follows: Section 2 gives the problem definition and notations. Section 3 introduces our proposed framework. Section 4 analyzes the k-stability of Winnowing-family algorithms and proposes an improved signature selection method based on collection frequencies. Section 5 introduces the candidate text generation method and Section 6 gives an effective improvement by eliminating unnecessary computations. Section 7 shows experimental results. Related works are introduced in Section 8 and Section 9 concludes the paper.

2 Problem Definition and Notations

We first give the formal definition of near duplicate text detection problem.

Definition 1 (Near Duplicate Text Detection). *Given a collection $\mathcal{C}$ of documents and a query document Q, a near duplicate text detection algorithm will return the best near duplicate text of Q in $\mathcal{C}$, indicated by $d \in \mathcal{C}$ and the* start *and* end positions *of the text in d (denoted by pos_{start} and pos_{end}), respectively.*

Although the precise definition of near duplicate is application-dependent, in most cases they are evaluated by a similarity function, which returns high scores when two text strings share a large portion of *identical* or *highly similar* substrings [18].

Note that the above problem definition is general enough to support several important applications. For example, the *near duplicate document detection* problem [17] can be deemed as a special case where the starting and ending positions are always the beginning and the end of the documents, respectively. For another example, the *text reuse problem* [18,23] can be solved by issuing multiple near duplicate text detection queries, each with a sentence as the query document.

Notations. All array indexes start from 1. Given a string T, $len(T)$ denotes its length. T has $len(T) - q + 1$ q-grams, which forms its *q-gram set* and is denoted as $grams_T$. The cardinality of a (multi-)set S is denoted by $|S|$. Given a q-gram g in a document, pos_g denotes the offset of its first character in the document.

3 A Framework of Near Duplicate Text Deteciton

Obviously, the naïve algorithm which performs character-to-character comparison between Q and every document $d \in \mathcal{C}$ is too costly and does not scale well

with the size of the document collection. Existing works are mainly based on selecting a small set of candidates $\mathcal{C}' \subseteq \mathcal{C}$ by extracting and matching document *signatures* [18]. The most prevalent form of signatures are q-grams, which are substring of q characters. A document of length l will generate $l - q + 1$ overlapping q-grams, and usually only a subset of them will be selected as the signatures of the document by a *signature selection* process.

We capture such approaches in a general framework as follows:

- In the *indexing phase*, for each document d in $\mathcal{C}$, a set of signatures is selected from its q-grams. The signature selection method (denoted as SelectSigs in Algorithm 1 and discussed in Section 4) could be any algorithm that will be introduced in Section 8, including *Winnowing* [17] and our *frequency biased Winnowing*. An inverted index, I, is then built that maps a signature to each of its occurrences (identified by document ID and the position within the document).
- In the *query processing phase* (See Algorithm 1), a set of signatures $\mathcal{S}_q$ is selected using the same signature selection method SelectSigs(Line 3). All the occurrences of each signature are collected via probing the index I, and then grouped by document (Lines 4–6). For each document returned, several candidate texts will be generated by the GenCandTexts function (to be discussed in Section 5) and stored in $CAND$ (Lines 7–8). Similarities between the query and each candidate text will be calculated (Lines 9–10) and the one achieving the maximum similarity will be returned (Line 11).

Algorithm 1. Query(Q)

```
1  CAND ← ∅;
2  Initialize sim and G to be empty hashtables;
   /* select Q's signatures                                          */
3  S_q ← SelectSigs(Q);
   /* find and group all occurrences of Q's signatures by document   */
4  for each signature s ∈ S_q do
5      for each pair (d_i, pos_j) ∈ I[s] do
6          G[d_i] ← G[d_i] ∪ {pos_j};
   /* generate candidate texts for each candidate document           */
7  for each d_i ∈ G do
8      CAND ← CAND ∪ GenCandTexts(G[d_i]);
   /* find the best candidate text                                   */
9  for each candidate c_i ∈ CAND do
10     sim[c_i] ← CalcSim(Q, c_i);
11 return arg max_{c_i ∈ CAND} sim[c_i];
```

The function CalcSim computes the similarity of a candidate text c_i against the query Q. In this paper, we consider one-sided Jaccard of q-gram multisets of c_i and Q, or

$$sim(c_i, Q) = \frac{|grams_{c_i} \cap grams_Q|}{|grams_Q|} \quad (1)$$

We break ties by favoring the shortest text.

4 Signature Selection Algorithms

While it is possible to select all the q-grams of a document as its signatures, this usually results in too many comparisons in practice due to the existence of some frequently occurring q-grams. On the other hand, selecting very few q-grams tends to miss many of the query results, or limit the flexibility of the algorithm (e.g., can only detect near duplicate sentences [23] or documents [15]). Hence, the signature selection process is a trade-off between efficiency, space and effectiveness (specifically recall). While many heuristic selection methods exists (such as [19,18,23]), we consider the Winnowing-family algorithms [17], as it has the guarantee that exact copy of substrings exceeding a certain length will always be detected.

In this section, we first briefly introduce and analyze Winnowing-family algorithms, including the original Winnowing method, and then identify a novel concept of k-stability, which is essential to quantifies the probability that a near duplicate text will be detected under the Winnowing-family algorithms. We then point out the limitation of the original Winnowing algorithm due to a dilemma between high stability and low efficiency. Finally, we propose a simple yet effective alternative Winnowing-based algorithm, named Frequency Biased Winnowing, that achieves a better trade-off than the original Winnowing method.

4.1 Winnowing-Family Algorithms

A Winnowing-family algorithm firstly calculates $f(g_i)$ for all the q-grams g_i in the document using an *injective function* $f(x)$. It then uses a sliding window of size w to select signatures. Within each window, it selects the q-gram g_{min}, such that $f(g_{min})$ is the smallest in the window, as the signature. If there is a tie, then the *rightmost* occurrence will be selected.

Example 1. Let $q = 3$ *and* $w = 4$. *Consider the document "*`abcdedcba`*", whose* q*-grams are:* {`abc`, `bcd`, `cde`, `ded`, `edc`, `dcb`, `cba`}. *Assume* $f(x) = (c_1 \cdot 7^2 + c_2 \cdot 7 + c_3) \bmod 23$, *where* c_i *indicates the ASCII code of the* i*-th character of the* q*-gram. Then the corresponding values are* $\{1, 14, 4, 15, 20, 7, 17\}$.

Given $w = 4$, *we have four windows: (**1**, 14, 4, 15), (14, **4**, 15, 20), (**4**, 15, 20, 7), and (15, 20, **7**, 17). q-grams corresponding to underlined bold numbers are signatures selected in each window. Thus,* `abc`, `cde` *and* `dcb` *with values* 1, 4 *and* 7 *will be selected as the signatures of the document.*

The original Winnowing algorithm [17] belongs to this Winnowing-family by using a random hash function with a sufficiently large codomain as $f(x)$. Later in Section 4.3 we will propose another frequency biased instance of the Winnowing-family algorithms.

Winnowing-family algorithms hold an important property named *locality*. An algorithm is l-*local* if, for any two identical strings with length at least l, they will always have at least one identical signature and thus will be guaranteed to be detected by the algorithm. This property is essential to detect *exact* copying. Consider two identical strings of length $l = w + q - 1$ where w is the window size. It is obvious that a Winnowing-family algorithm will always select the same minimum-valued q-gram in the windows as signatures. Therefore, all the Winnowing-family algorithms are $(w + q - 1)$-local.

4.2 k-Stability of Winnowing-Family Algorithms

While the locality property of Winnowing-family algorithms is essential for *exact* duplicate text detection, it does not help to analyze the performance of the algorithm for *near* duplicate text detection, which is arguably the more common and difficult case. To this end, we propose a novel concept named k-*stability*, which capture the ability for a Winnowing-family algorithm to detect text with small (or k) errors.

Definition 2 (k-stability of Winnowing-family algorithms). *Given a Winnowing-family algorithm M, consider randomly and independently changing k q-grams in a window W, which results in W'. The k-stability is the expected probability of that the signatures of W and W' are the same under the algorithm M.*

Obviously, the k-stability depends on content of the window W. In order to get a general, closed-formula characterization for an algorithm, in the following, we compute the k-stability for a window where its constituent q-grams are randomly and independently selected from the entire document collection (e.g., the distribution q-grams in the window are the same as those in the collection).

First, we establish the following Lemma.

Lemma 1. *For any discrete random variable X with possible values $\{x_1, x_2, \ldots, x_n\}$, the following equation holds for sufficiently large $t > 0$:*

$$\sum_{i=1}^{n} (p(x_i) \cdot (F(x_i) - \frac{p(x_i)}{2})^t) \approx \frac{1}{t+1}$$

where $p(x_i)$ is the probability mass function and $F(x_i)$ is the cumulative distribution function, i.e., $F(x_i) = \sum_{j=1}^{i} p(x_j)$.

Proof. Without loss of generality, we assume $x_i < x_{i+1}$. We also additionally define x_0, such that $x_0 < x_1$, and $F(x_0) = p(x_0) = 0$.

Consider a function

$$F_c(y) = \begin{cases} 0 & \text{, when } y < x_0 \\ \frac{a-b}{x_i - x_{i-1}} y + \frac{b \cdot x_i - a \cdot x_{i-1}}{x_i - x_{i-1}} & \text{, when } x_{i-1} \le y < x_i,\ i \in [1, n] \\ 1 & \text{, when } y \ge x_n \end{cases}$$

where $a = \sqrt[t+1]{(t+1)\cdot(F(x_i)-\frac{p(x_i)}{2})^t \cdot F(x_i)}$, and $b = \sqrt[t+1]{(t+1)\cdot(F(x_i)-\frac{p(x_i)}{2})^t \cdot F(x_{i-1})}$.

Since $F_c(y)$ is monotonous, bounded and right continuous, there must exist a random variable Y such that $F_c(y)$ is the cumulative distribution function of Y.

Then for any $x_1 \le x_i \le x_n$, we have:

$$\int_{x_{i-1}}^{x_i}(p_c(y)\cdot F_c^t(y))dy = \lim_{x\to x_i^-}\frac{1}{t+1}F_c^{t+1}(y)\Big|_{x_{i-1}}^{x} = p(x_i)\cdot(F(x_i)-\frac{p(x_i)}{2})^t$$

where $p_c(y)$ is the probability density function of Y. Then we have:

$$\sum_{i=1}^{n}(p(x_i)\cdot(F(x_i)-\frac{p(x_i)}{2})^t) = \sum_{i=1}^{n}\int_{x_{i-1}}^{x_i}(p_c(y)\cdot F_c^t(y))dy$$
$$=\frac{1}{t+1}F_c^{t+1}(x_n) - \frac{1}{t+1}F_c^{t+1}(x_0) - o(\frac{1}{t}) \approx \frac{1}{t+1} \qquad \square$$

Theorem 1. *The k-stability of a Winnowing-family algorithm with window size w is approximately* $\frac{w-k}{w+k}$.

Proof. Assume we randomly and independently pick w q-grams from the collection to form a window W, and another k q-grams to form a set S_{new}. We will then randomly and independently pick k q-grams from W and replace them with q-grams in S_{new}. We name these k q-grams as S_{old} and the rest q-grams as S_{rest}.

Apparently, the signature of W will not change after we subsitute k q-grams, only when the signature sig is in S_{rest}. In Winnowing-family algorithms, this indicates that $f(sig)$ is the rightmost samllest value among all the $w+k$ picked q-grams. The probability of this event can be estimated using Lemma 1:

$$Pr = \sum_{i=1}^{n}\left(p(x_i)\cdot(\sum_{j=1}^{i-1}p(x_j)+\frac{p(x_i)}{2})^{w+k-1}\right)$$
$$= \sum_{i=1}^{n}\left(p(x_i)\cdot(F(x_i)-\frac{p(x_i)}{2})^{w+k-1}\right) \approx \frac{1}{w+k}$$

There are $w-k$ q-grams in S_{rest} and we need to consider each of them. Thus for the event "signature in W is not changed" will happen with probability

$$\binom{w-k}{1}\cdot\frac{1}{w+k} = \frac{w-k}{w+k} \qquad \square$$

Note if we change one character, it will affect at most q q-grams. This observation straightly leads us to the following corollary.

Corollary 1. *Assume a Winnowing-family algorithm with gram length q and window size w. if we change m characters in a window, the signatures of it will remain the same, in the worst case, with probability* $p = \max(0, \frac{w-mq}{w+mq})$.

Table 1. Stability of Winnowing-family algorithms

Setting		m			
		1	2	3	4
$q = 50$	Worst Case	33.33%	0%	0%	0%
$w = 100$	Average	52.25%	25.70%	12.51%	6.11%
$q = 4$	Worst Case	94.67%	89.61%	84.81%	80.25%
$w = 146$	Average	94.77%	89.82%	85.12%	80.67%

Remark 1. *By letting $m = 0$, the Winnowing-family algorithms have 0-stability of 100%, which agrees with the locality property. So in this sense, we can deem k-stability as an extension of the locality property.*

Stability Analysis for the Original Winnowing Algorithm. Table 1 shows the probabilities of the signature in a window remaining the same after changing m characters, in worst case (i.e., in Corollary 1) as well as on average. With the typical setting of Winnowing from [17], where $q = 50$ and $w = 100$, changing even few characters will bring a significant decreasing to its stability, as well as the robustness of a near duplicate text detection method based on Winnowing.

However, from Corollary 1, as as showing in Table 1, we know that with the same locality of $(w + q - 1)$, a smaller q is much more preferable (e.g., $q = 4$) with respect to stability. Unfortunately, Winnowing cannot benefit from such q's. When q is smaller, the average occurence of q-grams is higher due to the reduction of distinct q-grams in the corpus. Then a random hash function $f(x)$ will have more chance to select a frequently-occurring q-gram as signature, which will affect the number of candidates as well as the query time for a Winnowing based method. This motives us to propose following Winnowing-family algorithm to fight against these problems.

4.3 Collection Frequency Biased Winnowing

We propose *Frequency Biased Winnowing*, which achieves a better stability by using small q's, yet it stilll achieves good efficiency for query processing.

Collection frequency, defined as the number of times that a term appears in the collection, is a statistical measurement to evaluate the importance of a term (or q-gram) in a collection. This leads us to use frequency of q-grams when selecting signatures. Rare q-grams are more preferable because they are more representative and able to make the length of posting lists shorter.

Our proposed signature selection algorithm, **frequency biased Winnowing**, is a Winnowing-family algorithm which takes collection frequency for each q-gram as their hash values. Since the frequency of two different q-grams might be the same, the *alphabet order* of the q-grams will be used to break such tie.

Example 2. Consider the same setting and document as in Example 1, where q = 3 and $w = 4$, q-grams of the document are: {`abc`, `bcd`, `cde`, `ded`, `edc`, `dcb`, `cba`}. *Assume their corresponding collection frequencies are* $\{18, 62, 50, 43, 30, 79, 30\}$.

Then underlined numbers (also in bold face) can be selected from 4 *windows: (**18**, 62, 50, 43), (62, 50, 43, **30**), (50, 43, **30**, 79), and (43, 30, 79, **30**). In the last window, the last* 30 *is selected because of the* alphabet order *of* `cba` *and* `edc`.

Obviously our proposed frequency biased Winnowing is a Winnowing-family algorithm, therefore it holds the *locality* property. And its *k-stability* is $\frac{w-k}{w+k}$.

Comparison with Winnowing. Frequency biased Winnowing prefers small q. Because when q becomes larger (e.g., $q > 10$), the number of possible distinct q-grams tends to be extremly large (i.e., $|\Sigma|^q$, where Σ is the alphabet), and most of them will have the frequency of 0 or 1. Then the algorithm selects signatures almost only based on their alphabet orders and has no benifit from collection frequencies. According to the experiments, q between 3 and 5 is the best setting for our method.

According to Corollary 1, a smaller q is more preferable for Winnowing-family algorithms with respect to its stability. On the other hand, since we always choose the q-gram with smallest collection frequency in the window, the posting list in our algorithm will not be very long. Therefore, our method will improve the effectiveness compared with Winnowing with large q's and also improve the efficiency compared with Winnowing with small q's. Our experiments have verified our analysis.

5 Generate Candidate Texts

In this section, we introduce the methodology of generating candidate texts in our near duplicate text detection method (i.e., the GenCandTexts function, Line 8 of Algorithm 1).

For each candidate document, we now have a sorted list that contains positions of matching signatures in the document. We do not use order among matching signatures, as it is quite common to have near duplicate text with reordered subparts. Instead, candidate texts are generated by applying the following heuristic rules of *merge signatures* and *determine boundaries*.

- **Merge Signatures.** We first combine *continuous* signatures together. Two signatures are continuous if they *may* be derived from two overlapping or adjacent windows. We can either store the window positions where the signature are generated. Otherwise, given the positions of two signatures s_i and s_j (assuming $pos_{s_i} < pos_{s_j}$), they are considered to be continuous if $pos_{s_j} - pos_{s_i} \leq 2w + q - 2$, as this is the worst case where the two signatures are the first and last signature in two adjacent windows, respectively.
- **Determine Boundaries.** Given an ordered merged list of sigantures $\{s_1, s_2, \ldots, s_m\}$ of document d, we generate the candidate text that has the *longest possible length*: we take the substring between positions pos_{start} and pos_{end}, where $pos_{start} = \max(1, pos_{s_1} - w + 1)$ and $pos_{end} = \min(pos_{s_m} + w + q - 1, len(d))$.

6 Heap Based Optimization

Lines 9-11 of Algorithm 1 compute the similarity for every candidate text and returns the largest one. This is not efficient if there is a large number of candidate texts which are long, or not similar to Q. In this section, we propose a heap-based optimization to reduce the number of similarity computations.

We observe that an upper bound of the similarity between two strings can be easily computed based on their lengths. In Equation (1), $|grams_Q|$ is fixed for a given query, thus the similarity between c and Q is only affected by $|grams_c \cap grams_Q|$. Since $|grams_c \cap grams_Q| \leq \min(|grams_c|, |grams_Q|)$, we can easily work out an upperbound of $sim(c, Q)$ as follows:

$$sim(c, Q) \leq sim_{ub}(c, Q) = \frac{\min(|grams_c|, |grams_Q|)}{|grams_Q|} = \min(\frac{|grams_c|}{|grams_Q|}, 1)$$

In addition, $sim_{ub}(c, Q)$ increases monotonically with $|grams_c| = len(c) - q + 1$.

The optimized query algorithm is shown in Algorithm 2, which should replace Lines 7–11 of Algorithm 1. The major modifications are:

- We use a max-heap H to organize candidate texts, based on their upper bound similarity *ub_score*.
- We maintain the current maximum score in *max_score*, and we terminate the loop only when the head of the heap H's upper bound score is no more than *max_score*.
- We use a similarity computation function CalcSim2 which can stop earlier during the similarity computation (See Algorithm 3). Note that we convert a multiset of q-grams to a set of q-grams by annotating a q-gram g as g_i if it is the i-th occurrence of q-gram. We perform the same transformation for Q and index it so that the set membership query (Line 4) can be performed efficiently.

Our experiments show that this optimization can save up to 99% number of similarity computations.

Algorithm 2. OptimizedQuery

```
   /* generate candidate texts for each candidate document          */
1  for each d_i ∈ G do
2      for each candidate text c_i ∈ GenCandTexts(G[d_i]) do
3          ub_score ← SimUB(c_i);
4          H.enqueue(c_i, ub_score);
   /* find the best candidate text                                   */
5  max_sim ← 0;
6  while H.head.ub_score > max_sim do
7      c ← H.dequeue();
8      max_sim ← max(max_sim, CalcSim2(Q, c, max_sim);
9  return c;
```

Algorithm 3. CalcSim2(Q, c, max_sim)

```
1 max_err ← |grams_c| · (1 − max_sim);
2 err ← 0;
3 for each q-gram g ∈ grams_c do
4     if g ∉ Q then
5         err ← err + 1;
6         if err ≥ max_err then
7             return 0;
8 return (|grams_c − err|)/|grams_Q|;
```

7 Experimental Results

In this section, we report our experiment results with two different implementations of our near duplicate text detection method, based on Winnowing [17] and frequency biased Winnowing respectively. We compare the performance of our proposed algorithm against Winnowing. We also show the improvments of heap based optimization introduced in Section 6.

7.1 Experiments Setup

Our near duplicate text detection system is implemented in Java and compiled using JDK 1.6.0. We use the Lucene library (Version 3.3.0)[1] to help build and retrieve the indexes. All experiments are carried out on a PC with a Quad-Core AMD Opteron 8378@2.4GHz Processor and 96GB RAM, and running Ubuntu 4.4.3.

Dataset. We use **PAN-PC-10**[2], a publicly available real dataset, to test our method. The **PAN-PC-10** dataset is published and used in Plagiarism Detection Task of PAN Workshop and Competition of Year 2010, which contains 11,148 source documents and 68,558 plagiarism cases. For each plagiarism case, the corresponding source sections are provided in the annotation of the dataset.

We remove those non-English documents from dataset, as our method is not designed for cross-lingual plagiarism. For both source documents and plagiarism cases, we converte all the non alphanumeric characters to '_' and all the uppercase characters to lowercase. Thus we finally have 10,482 documents with average length of 149,354 in the dataset. The total size of the dataset is 1.57 GB and the alphabet size is 37 (i.e., [a-z0-9_]).

Parameter Setting. We implemented our method on both Winnowing and frequency biased Winnowing under variant settings of q and w. We keep $q+w = 150$, such that the same locality guarantee will hold.

[1] `http://lucene.apache.org/`
[2] `http://www.webis.de/research/events/pan-10`

As suggested in [17], We set $q = 50$ and $w = 100$ for Winnowing. We also try other possible q's from 3 to 60 and Winnowing achieves its best performance considering both efficiency and effectiveness on PAN-PC-10 dataset when $q = 10$ and $w = 140$. Therefore, we report the experiment results for Winnowing on these two settings.

For frequency biased Winnowing, we also try different q's from 3 to 5, and results for q equals to 4 and 5 are reported.

Queries and Measurements. There are two main different types of near-duplications in the **PAN-PC-10** dataset [16], which are *artificial* (automatic) plagiarism cases and *simulated* (manual) plagiarism cases. In artificial plagiarism cases, there are three different obfuscation levels (i.e., *none*, *low* and *high*). For each of the four types above, we randomly select 100 plagiarism cases as queries and use the facts in the annotation as the ground truth for evaluation.

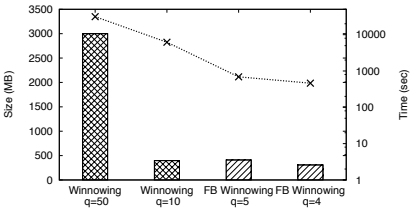
We focus on the following 5 measurements (all measurements are averaged over all queries):

- **Index Size**, which is the space needed to store the index.
- **Index Time**, which is the total time needed to index the whole collection.
- **Accuracy**. We use recall, precision and F_1 score to measure the accuracy of our method. Given query Q, its recall and precision are defined as $|\Omega| \,/\, |S|$ and $|\Omega| \,/\, |Q|$ respectively. Where Ω represents the detected plagiarized paragraph, and S indicates the real plagiarized paragraph.
- **Query Time**, which is the total time to process a query.
- **Calculated Candidate Texts**, which is the number of candidate texts whose similarity to Q is calculated. We report the number before and after applying optimization. We also report the total length of calculated candidate texts, as they are approxmately proportional to the query time, and the query time before applying optimization is extremly long thus we do not report it.

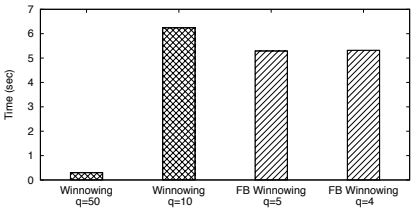
7.2 Indexing Time and Size

We plot the index size and indexing time for both algorithms with different parameters in Figure 1(a). The spots and line show indexing time. It is clearly that Winnowing takes much more time on indexing than frequency biased Winnowing. This is mainly due to the following two reasons. Firstly, the number of distinct signatures in Winnowing is much more than those in frequency biased Winnowing, (E.g., $30,982,703$ vs. $486,248$). Secondly, the time cost for calculating hash values in Winnowing is much longer than looking up the frequency table in frequency biased Winnowing.

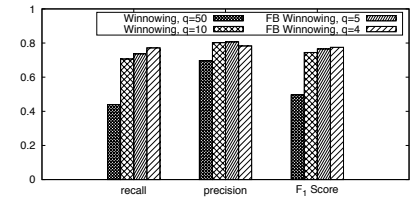
The bars show index size of two different algorithms. Apparently Winnowing also has a larger index size, especially when q is large. This is because of the different number of distinct signatures two algorithms, also the length of signatures in Winnowing is longer. It usually takes more space to store a String (e.g., signatures) than integers (e.g., positions), thus Winnowing requires more space.



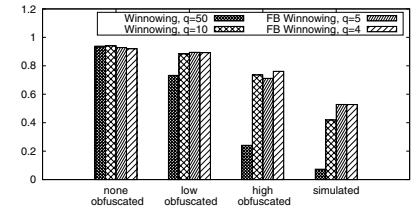
(a) Index Size and Time for Different Algorithms



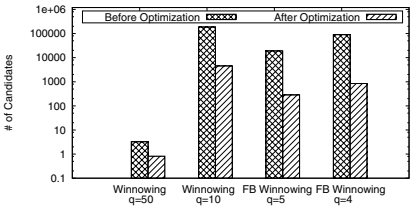
(b) Query Time for Different Algorithms



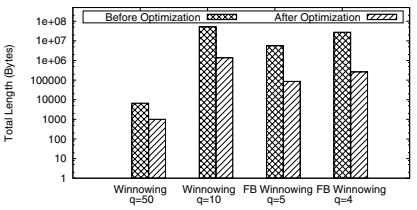
(c) Accuracy for Different Algorithms



(d) F_1 Score on Different Type of Queries



(e) Number of Calculated Candidate Texts



(f) Total Length of Calculated Candidate Texts

Fig. 1. Experimental Results

7.3 Accuracy

We plot the average recall, precision and F_1 score over 400 queries on both algorithms with different parameters in Figure 1(c). Clearly, frequency biased Winnowing has a much better accuracy than Winnowing.

More specifically, frequency biased Winnowing with $q = 4$ achieves F_1 score of 0.775 while Winnowing achieves 0.496 with $q = 50$ and 0.745 with $q = 10$. Considering that both algorithms achieve similar precisions, this gap is mainly due to the low recall of Winnowing algorithm. As we stated in Section 4.2, the stability affects recall for Winnowing-family algorithm. Winnowing with larger q has lower stability thus lower recall than frequency biased Winnowing (i.e., $44.00\% - 70.66\%$ vs. $76.56\% - 77.15\%$).

We also plot F_1 score for both algorithms on *different types* of queries in Figure 1(d). Both algorithms perform well on none obfuscated plagiarism cases. Winnowing starts to fail on low obfuscated artificial plagiarism cases, especially with $q = 50$. And frequency biased Winnowing completely beats Winnowing

on hard queries (i.e., high obfuscated artificial and simulated plagiarism cases). This is also due to Winnowing's low recall, especially for hard queries, where the loss of stability will bring non-negligible impact on its accuracy.

It is worth mentioning the results of PAN-10 competition. The first place achieves average recall of 69.17% and F_1 score of 0.797, while the second place achieves 62.99% and 0.709. Although our current method cannot support cross-lingual plagiarism cases, and our queries are generated based on the ground truth, it is still justified to say that our near duplicate text detection method is competitive against the top works in the area.

7.4 Query Time

We plot the average query time for both algorithms with different q's in Figure 1(b). We observe that the query time of frequency biased Winnowing is smaller than Winnowing with $q = 10$, but much larger than Winnowing with $q = 50$. Winnowing generates very few or even no candidate for hard queries when $q = 50$. But when $q = 10$, it generates more candidates than frequency biased Winnowing, which leads to more similarity computations thus more time cost.

7.5 Calculated Candidate Texts

In order to verify our analysis of query time as well as show the improvement of heap based optimization, we plot the number of calculated candidate texts and the total length of them, before and after using heap based optimization, in Figure 1(e) and Figure 1(f) respectively. Our optimization brings significant improvements. Up to 99% of candidate texts are skipped, so does the total length of them. Our optimization also saves approximately 99% of query time, as most time is spent on similarity computations. This is due to that most queries have a high similarity answer, once we find it, we can almost ignore the rest candidates.

8 Related Work

Winnowing is considered very important in various areas and used in a number of works. For example, [22] uses it to partition the files and further detect the redundancy in the file. [10] uses it to generate variable sized blocks in order to perform accelerating multi-pattern matching. It also used to quickly find the possible plagiarism parts [5], but only "copy and paste" plagiarism cases are explored. [11] uses it to shorten the size of input data on its secure file scanning system on enterprise networks. However, seems no one focuses on improving Winnowing.

There are many works focusing on near duplicate text detection by using different signature selecting methods. [3] selects every l-th q-grams, which is susceptible to positional changes such as insertion or deletion. The $\pmod p = 0$ scheme [13] selects q-grams whose hash values can be divided by p, but it is possible to select nothing from a document. Very similar to Winnowing, [2] selects the q-gram whose hash value is smaller than its previous and next h q-grams. It

holds a weaker locality which only guarantees to return same or no signature for identical substrings. Spotsigs [19] takes a chain of words that follows a stopword as signatures to find near duplicate Web documents. [1] takes the idea of Spotsigs but also considers the standard tf-idf weighting. It uses sampling to detect duplicate news stories and achieves a good performance. All metioned methods, including other methods like [9,15,18,21], either offer no locality guarantees or suffer from large number of false positives.

There are previous works exploiting collection statistics (mainly inverse document frequency). [8] uses words with the first 30 to 60 highest idf, [7] selects terms with high idf, and its extention [12] uses external collection statistics.

Another category of approaches to detect near duplicate document is to find documents that are highly similar to the query document as a whole. Representative approaches include those emplying Jaccard similarities based on tokens or word n-grams, and those employing Hamming distance based on a binary feature vectors constructed from the documents. There are efficient exact computation algorithms [20,14,24] as well as approximate algorithms based on locality sensitive hashing [4,1,6].

9 Conclusion

In this paper, we propose a new near duplicate text detection framework using signatures selected by Winnowing-family algorithms. We raise a new concept named k-stability with theoretical analysis to measure the stability of Winnowing-family algorithms when small errors happening, and propose a new frequency biased Winnowing algorithm.We also propose candidate text generation methods and optimization to improve the performance of our framework. Our experimental result shows the significant improvement of our proposed algorithm and the good performance on a plagarism detection benchmark.

References

1. Alonso, O., Fetterly, D., Manasse, M.: Duplicate news story detection revisited. Tech. Rep. 60, Microsoft Research (2013)
2. Bjørner, N., Blass, A., Gurevich, Y.: Content-dependent chunking for differential compression, the local maximum approach. J. Comput. Syst. Sci. 76(3-4), 154–203 (2010)
3. Brin, S., Davis, J., Garcia-Molina, H.: Copy detection mechanisms for digital documents. In: SIGMOD Conference, pp. 398–409 (1995)
4. Broder, A.Z., Charikar, M., Frieze, A.M., Mitzenmacher, M.: Min-wise independent permutations (extended abstract). In: STOC, pp. 327–336 (1998)
5. Butakov, S., Scherbinin, V.: On the number of search queries required for internet plagiarism detection. In: ICALT, pp. 482–483 (2009)
6. Charikar, M.: Similarity estimation techniques from rounding algorithms. In: STOC, pp. 380–388 (2002)
7. Chowdhury, A., Frieder, O., Grossman, D.A., McCabe, M.C.: Collection statistics for fast duplicate document detection. ACM Trans. Inf. Syst. 20(2), 171–191 (2002)

8. Conrad, J.G., Guo, X.S., Schriber, C.P.: Online duplicate document detection: signature reliability in a dynamic retrieval environment. In: CIKM, pp. 443–452 (2003)
9. Hamid, O.A., Behzadi, B., Christoph, S., Henzinger, M.R.: Detecting the origin of text segments efficiently. In: WWW, pp. 61–70 (2009)
10. Hua, N., Song, H., Lakshman, T.V.: Variable-stride multi-pattern matching for scalable deep packet inspection. In: INFOCOM, pp. 415–423 (2009)
11. Jiang, J., Tang, Y., Liu, B., Xu, Y., Wang, X.: Skip finite automaton: A content scanning engine to secure enterprise networks. In: GLOBECOM, pp. 1–5 (2010)
12. Kolcz, A., Chowdhury, A., Alspector, J.: Improved robustness of signature-based near-replica detection via lexicon randomization. In: KDD, pp. 605–610 (2004)
13. Manber, U.: Finding similar files in a large file system. In: USENIX Winter, pp. 1–10 (1994)
14. Manku, G.S., Jain, A., Sarma, A.D.: Detecting near-duplicates for web crawling. In: WWW, pp. 141–150 (2007)
15. Mittelbach, A., Lehmann, L., Rensing, C., Steinmetz, R.: Automatic detection of local reuse. In: Wolpers, M., Kirschner, P.A., Scheffel, M., Lindstaedt, S., Dimitrova, V. (eds.) EC-TEL 2010. LNCS, vol. 6383, pp. 229–244. Springer, Heidelberg (2010)
16. Potthast, M., Barrón-Cedeño, A., Eiselt, A., Stein, B., Rosso, P.: Overview of the 2nd international competition on plagiarism detection. In: CLEF (Notebook Papers/LABs/Workshops) (2010)
17. Schleimer, S., Wilkerson, D.S., Aiken, A.: Winnowing: Local algorithms for document fingerprinting. In: SIGMOD Conference, pp. 76–85 (2003)
18. Seo, J., Croft, W.B.: Local text reuse detection. In: SIGIR, pp. 571–578 (2008)
19. Theobald, M., Siddharth, J., Paepcke, A.: Spotsigs: robust and efficient near duplicate detection in large web collections. In: SIGIR, pp. 563–570 (2008)
20. Xiao, C., Wang, W., Lin, X., Yu, J.X., Wang, G.: Efficient similarity joins for near-duplicate detection. ACM Trans. Database Syst. 36(3), 15 (2011)
21. Yang, H., Callan, J.P.: Near-duplicate detection by instance-level constrained clustering. In: SIGIR, pp. 421–428 (2006)
22. Zhang, J., Suel, T.: Efficient search in large textual collections with redundancy. In: WWW, pp. 411–420 (2007)
23. Zhang, Q., Wu, Y., Ding, Z., Huang, X.: Learning hash codes for efficient content reuse detection. In: SIGIR, pp. 405–414 (2012)
24. Zhang, X., Qin, J., Wang, W., Sun, Y., Lu, J.: Hmsearch: An efficient hamming distance query processing algorithm. In: SSDBM (2013)