Local Text Reuse Detection

Jangwon Seo
jangwon@cs.umass.edu

W. Bruce Croft
croft@cs.umass.edu

Center for Intelligent Information Retrieval
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003

ABSTRACT

Text reuse occurs in many different types of documents and for many different reasons. One form of reuse, duplicate or near-duplicate documents, has been a focus of researchers because of its importance in Web search. Local text reuse occurs when sentences, facts or passages, rather than whole documents, are reused and modified. Detecting this type of reuse can be the basis of new tools for text analysis. In this paper, we introduce a new approach to detecting local text reuse and compare it to other approaches. This comparison involves a study of the amount and type of reuse that occurs in real documents, including TREC newswire and blog collections.

Categories and Subject Descriptors

H.3.1 [**Content Analysis and Indexing**]: Indexing methods

General Terms

Algorithms, Measurement, Experimentation

Keywords

Text reuse, fingerprinting, information flow

1. INTRODUCTION

Text reuse and duplication can occur for many reasons. Web collections, for example, contain many duplicate or near-duplicate versions of documents because the same information is stored in many different locations. *Local* text reuse, on the other hand, occurs when people borrow or plagiarize sentences, facts, or passages from various sources. The text that is reused may be modified and may be only a small part of the document that is being created.

Near-duplicate document detection has been a major focus of researchers because of the need for these techniques in Web search engines. These search engines handle enormous collections with a great number of duplicate documents. The duplicate documents make the system less efficient in that they consume considerable system resources. Further, users typically do not want to see redundant documents in search results. Many efficient and effective algorithms for near-duplicate document detection have been described in the literature [1, 4, 5, 6].

The obvious application involving local text reuse is plagiarism detection, but being able to detect local reuse would be a powerful new tool for other possible applications involving text analysis. For example, Metzler et al. [15] discussed tracking information flow, which is the history of statements and "facts" that are found in a text database such as news. This application was motivated by intelligence analysis, but could potentially be used by anyone who is interested in verifying the sources and "provenance" of information they are reading on the Web or in blogs.

Local text reuse detection requires different algorithms than have been developed for near-duplicate document detection. The reason for this is that, in the case of local text reuse, only a small part (or parts) of a document may have been taken from other sources. For example, state-of-art near-duplicate detection algorithms like the locality sensitive hash [5] assume a transitive relation between documents. That is, if a document A is a near-duplicate of document B, which is a near-duplicate of document C, then document A should be a near-duplicate of document C. A text reuse relationship based on parts of documents, however, violates this assumption, as shown in Figure 1.

In this paper, we focus on algorithms for detecting local text reuse based on parts of documents. In Section 2, we discuss the related literature. In Section 3, we expand on the idea of local text reuse by introducing categories of reuse. These categories are the basis of our experimental evaluation. In Section 4, we introduce a novel algorithm for local text reuse detection called *DCT fingerprinting*. This algorithm is evaluated for efficiency and effectiveness in Section 5. In Section 6, the local reuse detection algorithm is used to measure the amount and type of text reuse that occurs in TREC news and blog collections.

2. RELATED WORK

There have been broadly two approaches to text reuse detection. One approach is using document fingerprints through hashing subsequences of words in documents. This approach is known to work well for copy detection. Shivakumar and Garcia-Molina [19, 20] and Broder [3] introduced efficient frameworks. Since handling many fingerprints is too expensive, various selection algorithms for fin-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGIR'08, July 20–24, 2008, Singapore.
Copyright 2008 ACM 978-1-60558-164-4/08/07 ...$5.00.

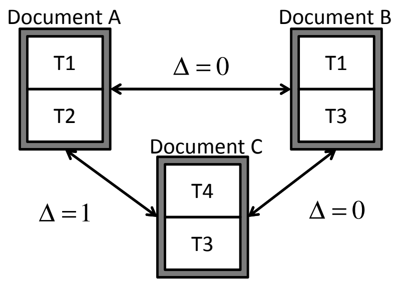

Figure 1: Text reuse by partial text of documents (local text reuse), where Δ is an indicator function that is 1 if two documents have no text reuse relationship. A document A and a document B have a text reuse relationship by text T1. Document B and a document C have a relationship by text T3. But, Document A and C have no such relationship.

gerprints were proposed by Manber [14], Heintze [9], Brin et al. [2] and Schleimer [18]. Broder et al. [4] suggested an efficient near-duplicate algorithm generating new fingerprints (super-shingles) by hashing sequences of fingerprints again. Charikar [5] introduced a hashing algorithm based on random projections of words in documents. Henzinger [10] empirically compared a variant of Broder et al's algorithm and Charikar's algorithm on a large scale Web collection. Chowdhury et al. [6] and Bernstein and Zobel [1] proposed filtration algorithms for fast near duplicate detection.

Another approach is computing similarities between documents in the Information Retrieval sense. Shivakumar and Garcia-Molina [19] and Hoad and Zobel [11] suggested similarity measures based on relative frequency of words between documents. Metzler et al. [15] compared similarity measures using an evaluation corpus that was developed for studies of local text reuse.

3. TEXT REUSE DETECTION

3.1 Fingerprints

Most algorithms for near-duplicate detection use a *chunk* or a *shingle* as a text unit [2, 4, 9, 14, 18]. A chunk or a shingle generally represents a substring or a subsequence of words. Since these chunks are based on small pieces of text, they may be expected to also work well for our local text reuse detection task. Therefore, the techniques mentioned later use this approach. The drawback with this approach is that some forms of text modification will not be captured.

In order to efficiently perform text reuse detection, the string form of a chunk must be converted into a numeric form. Generally, the conversion is performed by hashing algorithms such as MD5 [17] or Rabin fingerprinting [16]. For convenience, we refer to the hash value as the chunk instead of the string. We use a subset of a set of chunks generated in a document to represent the document for text reuse detection. Chunk values in the subset are referred to as *fingerprints* and the process for obtaining them is referred to as *fingerprinting*.

3.2 Describing the Amount of Shared Text

A text reuse relationship is a pairwise relationship. Given a pair of documents, we need to estimate the amount of text shared between the two documents. The amount of

Table 1: Definitions of text containment terms

Term	*Most*	*Considerable*	*Partial*
Range	$C(A,B) \geq 0.8$	$C(A,B) \geq 0.5$	$C(A,B) \geq 0.1$

Table 2: Text Reuse Categories

Term	**Relationship**
$C1$	*Most-Most*
$C2$	*Most-Considerable*
$C3$	*Most-Partial*
$C4$	*Considerable-Considerable*
$C5$	*Considerable-Partial*
$C6$	*Partial-Partial*

text of document A that is shared with document B can be represented as a ratio of the number of shared fingerprints to the number of fingerprints of document A. The ratio, *containment* of A in B [3] is estimated as follows:

$$C(A,B) = \frac{|F_A \cap F_B|}{|F_A|} \quad (1)$$

where F_A and F_B are sets of fingerprints of document A and B, respectively.

Note that the shared fingerprint ratio is a non-symmetric metric, i.e. $C(A,B) \neq C(B,A)$. Generally, symmetric metrics like *resemblance* [3] have been used for near-duplicate detection because it has to be determined whether the estimated value is greater than a threshold in order to easily check if the document pair has a near-duplicate relationship. Since our goal is to understand more general forms of text reuse rather than simply judging near-duplicate documents, we use the non-symmetric metric that contains more information.

We divide containment values into three ranges as shown in Table 1. That is, if greater than 80%, 50% or 10% of the total fingerprints of document A are shared with a document B, then we say that *most*, *considerable* or *partial* text of document A is reused by document B. These thresholds are not fixed but may be changed based on the properties of collections or goals of the text reuse application. Here, we set the values based on reviewing results for various collections.

3.3 Text Reuse Category

General text reuse occurs in various levels. Most of the text of a document might be shared with other documents, or only several words of a document might be shared with other documents. As a basis for evaluating local detection techniques and investigating the frequency of text reuse, we classify text reuse relationships into six categories as shown in Table 2. For example, if *partial* text of document A is shared with document B and the shared text is *most* text of document B, then document A and document B have a $C3$ type relationship.

Note that in a broad sense, $C1$, $C2$ and $C4$ correspond to near-duplicate cases, whereas $C3$, $C5$ and $C6$ correspond to local text reuse. We now briefly describe each category or type. An analysis of real collections based on these categories is presented in Section 6.

- *C1 (Most-Most)*: This is a typical near-duplicate case, where two documents are almost identical.
- *C2 (Most-Considerable)*: Generally, in this case, a short passage is added to text of another document. A typical example can be observed in blogs, i.e. copying

the entire text of a news article and appending a short comment about the article.

- *C3 (Most-Partial)*: In this case, a whole document is used as a partial text of a new document. $C3$ types are typically shown in cases where a news article is composed of several small news articles or where a document quotes interviews from other short news articles.
- *C4 (Considerable-Considerable)*: This is a case where a new document is composed of large parts of other documents.
- *C5 (Considerable-Partial)*: This is generally similar to $C4$ except for the amount of the shared text.
- *C6 (Partial-Partial)*: This generally happens with boilerplate text or common phrases.

4. FINGERPRINTING TECHNIQUES FOR TEXT REUSE DETECTION

The fingerprints of a document are numerical representations for text reuse detection and, for local reuse detection, should represent as much as possible of the content of the document. For example, if the fingerprints of documents are hash values of the first three sentences of each document, then they do not capture enough of the content. Documents can be very different even if the fingerprints of the documents are identical.

For efficient text reuse detection, an inverted index is generally built with fingerprints extracted from documents. To find all documents which have text reuse relationships with a document A, we first read all inverted lists of the fingerprints of document A, then merge the lists, and finally, find text reuse relationships according to the rules in Section 3. The first step is the most critical in time complexity because it requires significant I/O access, whereas the other steps can be performed in main memory. Since the maximum length of the inverted list is the number of documents in the collection, this can be naively thought as an $O(Mn)$ algorithm, where M and n are the number of the fingerprints of document A and the number of documents in the collection, respectively.

On real collections, however, the length of the inverted list is at most the occurrence count of the most frequent fingerprints in the collection. Moreover, we can restrict the upper bound of the length by setting very common fingerprints to stop-fingerprints in the same way as stop-words in Information Retrieval. Therefore, the practical time complexity is $O(Ml)$, where l is the restricted length of the inverted list such that $l \ll n$.

When we try to discover all text reuse relationships in the collection, the above process is repeated n times, where n is the number of documents in the collection. This is an $O(nml)$ algorithm, where m is the average number of the fingerprints of a document.

Since the length of the inverted list is not only invariant regardless of the fingerprinting methods but also generally small, the average number of the fingerprints m is more critical than the length of the list l for efficiency.

In sum, good fingerprinting techniques for text reuse detection should satisfy the following properties.

- For accuracy, a fingerprinting technique should generate fingerprints that accurately represent documents.
- For efficiency, a fingerprinting technique should generate the smallest number of fingerprints possible.

Considering accuracy and efficiency, we introduce several fingerprinting methods. There are broadly two kinds of fingerprinting techniques for text reuse detection: overlap methods and non-overlap methods. We first review the overlap methods and introduce the non-overlap methods later.[1]

4.1 Overlap Methods

Overlap methods use a sliding window. Basically, the window is shifted by a word, and a word sequence in the window or its hash value is handled as a chunk. If the size of the window is k, i.e. the i^{th} window contains the i^{th} word to the $i+k-1^{th}$ word in the document, then the i^{th} chunk in document D is computed as follows:

$$C(D,i) = h(t(D,i), t(D,i+1), \cdots, t(D,i+k-1)) \qquad (2)$$

where h and $t(D,i)$ are the hash function and the i^{th} term in document D.

As you see, $k-1$ of the words that appeared in the previous window appear in the current window again. That is, methods based on the sliding window are referred to as *overlap methods* in that the adjacent windows overlap each other.

Generally, overlap methods generate many chunks, but show good performances. However, since processing a large number of chunks as fingerprints can be too expensive, chunk selection techniques have been introduced [9, 14, 18].

4.1.1 k-gram

k-gram is the simplest technique of the overlap methods. It uses all the chunks generated from each sliding window as fingerprints. Thus, the number of the fingerprints of document D is computed as follows:

$$M_{k\text{-}gram}(D) = L(D) - k + 1 \qquad (3)$$

where $L(D)$ is the term count of document D.

As k-gram uses all chunks, it generally shows good performance. However, it might be infeasible in big collections because of too many fingerprints.

4.1.2 0 mod p

Instead of using all the chunks generated by the sliding window, *0 mod p* tries to select some of them as fingerprints [14]. A random selection of chunks would reduce the number but we cannot predict which chunks would be selected. If different chunks are selected each time, then two documents may be determined to be different even when they are identical. Therefore, all chunk selection methods have to satisfy the property that the same chunks should be selected for identical documents.

0 mod p selects only chunks such that $C(D,i)\ mod\ p \equiv 0$. When two documents are identical, chunks in the documents are the same. Assuming that the chunk values are uniformly distributed, the expected number of selected chunks, i.e. the number of fingerprints of document D, is given by:

$$M_{0\,mod\,p} = M_{k\text{-}gram}(D)/p \qquad (4)$$

That is, *0 mod p* can reduce the number of the fingerprints by a factor p.

This method may however not represent the whole document accurately. For example, although the upper halves

[1]Although we only describe cases of using subsequences of words for fingerprinting in this paper, all methods introduced here can be used for fingerprinting based on substrings.

of two documents are identical, the lower halves might be different. In the worst case, if chunks in either the upper halves or the lower halves are selected, then the detection algorithms would falsely determine the reuse relationship.

4.1.3 Winnowing

Winnowing is another selection method based on k-gram [18]. Winnowing adopts another fixed size window, i.e. a winnowing window over the sequence of chunks generated by the original window, and it selects a chunk whose value is the minimum in each winnowing window. If there is more than one minimum value in the winnowing window, then the rightmost minimum value in the window is selected.

Schleimer et al. [18] showed that winnowing performs better than *0 mod p* in practice. Further, they showed that the expected number of fingerprints has a lower bound as follows:

$$M_{winnowing}(D) = \frac{2}{w+1} M_{k\text{-}gram}(D) \qquad (5)$$

where w is the size of winnowing window.

4.2 Non-overlap Methods

A main idea of non-overlap methods is splitting text into a few meaningful text segments such as phrases or sentences instead of generating many subsequences of words. Thus, sliding windows are not used, and accordingly, there is no overlap between chunks. We refer to a process of splitting text segments as *breaking*. A word position where a break occurs is referred to as a *breakpoint*.

A chunk value of non-overlap method is computed as follows:

$$C(D, i+1) = h(t(D, b(i)+1), \cdots, t(D, b(i+1))) \qquad (6)$$

where $b(i)$ is a breakpoint for the i^{th} text segment.

A simple breaking method is using punctuation characters like '.,!?'. In case of well formatted text, punctuation characters play an significant role in forming sentence boundaries. However, in general Web text including blogs, there is substantial noise and text segments are often defined by HTML tags rather than by punctuation. In such environments, where text reuse detection algorithms are often used, breaking by punctuation does not work well. Therefore, the non-overlap methods to be introduced here are based on mathematical properties of text instead of punctuation.

4.2.1 Hash-breaking

Hash-breaking [2] is a non-overlap version of *0 mod p*. A hash value $h(w)$ for each word w is computed, and hash values such that $h(w)\ mod\ p \equiv 0$ are selected as breakpoints for text segments. That is, a sequence of words from the next word of the previous breakpoint to the current breakpoint is considered as a meaningful text segment. We can get a chunk by applying Equation (6) to the text segment. Since the number of chunks generated in a document is relatively small, any selection method does not need to be used and all the chunks are used as fingerprints of the document.

The expected number of fingerprints is given by $L(D)/p$. That is, the average length of text segments is p. In bad cases, the length of text segments might be much shorter than we expected. The worst case is that the text segment includes only a very common word such as 'a' or 'the'. Then, the fingerprint is noise and hurts the text reuse performance. To address this problem, we suggest a simple revision of the original hash-breaking algorithm. When we set a p value, we expect the length of text segment to be p. Thus, we can ignore text segments whose lengths are shorter than p. We can reduce noisy fingerprints through this approach.

A weak point that still remains is that hash-breaking is too sensitive to small modifications of text segments. In case of overlap methods, even if there is a small change in a sentence, then adjacent chunks without the change as well as chunks with the change are generated by window overlapping. Accordingly, the effect of the change can be minimized. On the other hand, since there is no overlap between chunks split by hash-breaking and the number of chunks is small, the change might be overestimated.

4.2.2 DCT fingerprinting

We propose a robust method called *DCT fingerprinting* to address the sensitivity problem of hash-breaking. The Discrete Cosine Transform (DCT) is a real valued version of Fast Fourier Transform (FFT) and transforms time domain signals into coefficients of frequency component. By exploiting a characteristic that high frequency components are generally less important than low frequency components, DCT is widely used for data compression like JPEG or MPEG. DCT is formulated as follows:

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \qquad (7)$$

$$k = 0, 1, \cdots, N-1 \qquad (8)$$

where x_n and X_k are the n^{th} value in the time domain signal sequence and a coefficient of the k^{th} frequency component, respectively. Note that the length of the time domain signal sequence N is the same as the number of the frequency domain components.

A main idea of DCT fingerprinting is that a sequence of hash values of words can be considered as a discrete time domain signal sequence. That is, we can transform the hash value sequence into the coefficients of frequency components by using DCT.

The process of DCT fingerprinting is composed of seven steps.

i. Get a text segment by using revised hash-breaking with a parameter p.
ii. Compute hash values for words in the text segment, $x_0, x_1, \cdots, x_{N-1}$, where N is a length of the text segment.
iii. Perform a vertical translation of the hash values so that the median of the hash values is located at 0.
iv. Normalize the hash values by the maximum value.
v. Perform DCT with the normalized hash values.
vi. Quantize each coefficient to be fitted in a small number of bits, e.g., 2, 3 or 4 bits.
vii. Form a fingerprint with the quantized coefficients Q_k's as shown in Figure 2. If N is so big that all Q_k's cannot fit the format, use only lower frequency coefficients. One approach is to use only the p lower frequency coefficients if the length of the text segment N is greater than the hash-breaking parameter p.

DCT fingerprinting is expected to be more robust against small changes than hash-breaking. As you see in Equation (7), when there is a small change of an input value, i.e. a hash value of a word, the change is propagated over all coefficients by a reduced effect. Since we quantize the coefficients, the final fingerprint value can be kept unchanged. That is,

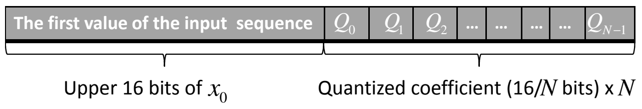
The first value of the input sequence
Q_0
Q_1
Q_2
…
Q_{N-1}
Upper 16 bits of x_0
Quantized coefficient (16/N bits) x N

Figure 2: A format of 32bit DCT fingerprint

this robustness can be interpreted as an advantage of data reduction. The following examples show the robustness of DCT fingerprinting. The numbers in [] are the fingerprints for the right string sequences.

`[0x295D0A52]`	one woman **comedy** by person Willy
`[0x295D0A52]`	one woman **show** by person Willy
`[0xF1315F87]`	company **scheduled** another money
`[0xF1315F87]`	company **slated** another money

It is difficult to show theoretically how many changes DCT fingerprinting can be tolerant of because input signal values are almost randomly mapped to by hashing. That is, while a minor change, e.g., a change from 'product' to 'products' might cause a big change of the hash value of the word, a word replacement might be coincidentally mapped to the same value. Nevertheless, a single word change tends to change a few high frequency components, and we can ignore the high frequency components by the formatting scheme. Thus, we can expect that DCT fingerprinting sometimes handles a single word change. When more than one word is changed, the input signal shape is likely to be distorted and the DCT coefficients are changed. Moreover, if words are added to or removed from the text segment, then even the number of the coefficients is changed. Therefore, we conclude that DCT fingerprinting can be tolerant of at most a single word replacement.

Note that DCT fingerprinting and hash-breaking generate the same number of fingerprints. Although computation of DCT fingerprinting might seem somewhat expensive, it requires only p^2 more multiplications for each fingerprint compared to hash-breaking. In practical, since running time of text reuse detection mostly depends on the amount of I/O access, computation complexity of DCT fingerprinting has little impact on the time.

We should mention that there have been some efforts to use DCT as a robust hash function for images [12] or videos [7]. However, they have totally different schemes and contexts from ours and there is no similarity except for using the DCT approach.

5. EVALUATION

5.1 Comparison of Fingerprinting Techniques

To compare fingerprinting techniques, we designed experiments on a test dataset. To build the dataset, we ran each technique on TREC newswire collection and collected detected document pairs. Then, by manually judging them, we obtained 100 labeled document pairs for each text reuse type defined in Section 3.3.

We first optimized parameters for each fingerprinting technique with 50 document pairs of the dataset. Then, with the remaining 50 document pairs, we ran our text use detection engine using each technique with the tuned parameters to evaluate the performance. Small values were chosen for the parameters, e.g., $k = 3$ for k-gram, $p = 6$ for *0 mod p*, $w = 10$ for winnowing and $p = 3$ for hash-breaking and DCT fingerprinting. For near-duplicate detection, big values for the parameters are generally used to avoid false detections and to reduce the number of fingerprints. However, in order to catch local text reuse, small values are more desirable.

For our experiments, we used 32 bit integers for fingerprints. Since we used MD5 [17], i.e. a 128 bit algorithm for hashing, we had to choose the upper 32 bits of the MD5 value. We used the harmonic mean of recall and precision, $F1$ as an evaluation metric for accuracy in that this task could be considered as a classification problem with six categories. The number of fingerprints was used as another metric to evaluate efficiency.

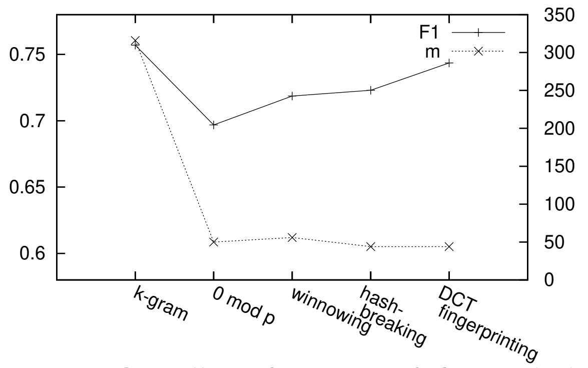

Figure 3: Overall performance of fingerprinting techniques. $F1$ and m represent the average of $F1$ of the six categories and the average number of fingerprints of a document, respectively.

Table 3 and Figure 3, 4 and 5 show the experimental results. Overall, in accuracy, k-gram outperformed the others as shown in Figure 3. However, it generated too many fingerprints as we predicted. DCT fingerprinting showed the second best accuracy with a small number of fingerprints. For near-duplicate cases ($C1$, $C2$ and $C4$), both k-gram and winnowing showed good performance as shown in Figure 4. For the local text reuse cases ($C3$, $C5$ and $C6$), DCT fingerprinting worked as well as k-gram as shown in Figure 5. It is also noticeable that DCT fingerprinting generated many fewer fingerprints than k-gram while they showed similar performance.

Based on the results, if there is enough resources and the target collection is small, then k-gram is the best. Otherwise, DCT fingerprinting would be the most practical choice. When the main purpose is near-duplicate detection, winnowing might be a good choice. For local text reuse, DCT fingerprinting is most desirable considering both accuracy and efficiency.

5.2 Empirical Validation for DCT fingerprinting

DCT fingerprinting needs to be validated in practice in that its robustness might be misunderstood as a result of hash collisions and false detection. The best way to validate would be to directly examine all detected pairs, but that is impossible. Thus, we chose an indirect method: comparison with k-gram. We know that k-gram is the most accurate method. Moreover, it has the lowest probability of false detection because it uses all chunks. Although testing on a larger collection where the false detection probability tends to be higher is preferable, running k-gram on big collections is too expensive. Therefore, we used a TREC

Table 3: Performance of fingerprinting techniques on TREC newswire collection. Numbers are values of $F1$. m represents the average number of fingerprints.

	k-gram	*0 mod p*	winnowing	hash-breaking	DCT fingerprinting
$C1$	0.8958	0.8125	0.8791	0.7529	0.7816
$C2$	0.6783	0.5968	0.7156	0.7475	0.7475
$C3$	0.6234	0.5714	0.6234	0.7711	0.7857
$C4$	0.7480	0.7193	0.7778	0.6726	0.7179
$C5$	0.7327	0.6304	0.5941	0.6195	0.6415
$C6$	0.8636	0.8511	0.7111	0.7742	0.7872
Average $F1$	0.7570	0.6969	0.7186	0.7230	0.7436
m	316	50	56	44	44

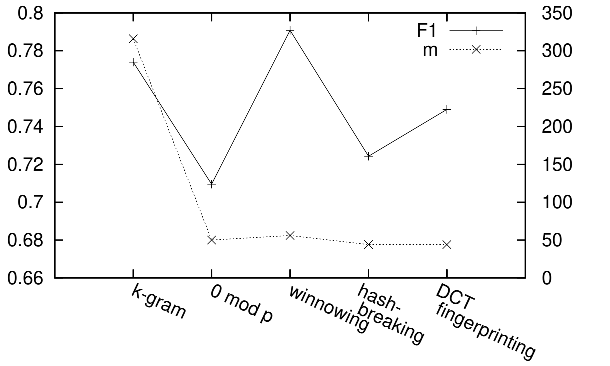

Figure 4: Near-duplicate detection performance of fingerprinting techniques. $F1$ and m represent the average of $F1$ of $C1$, $C2$ and $C4$ and the average number of fingerprints of a document, respectively.

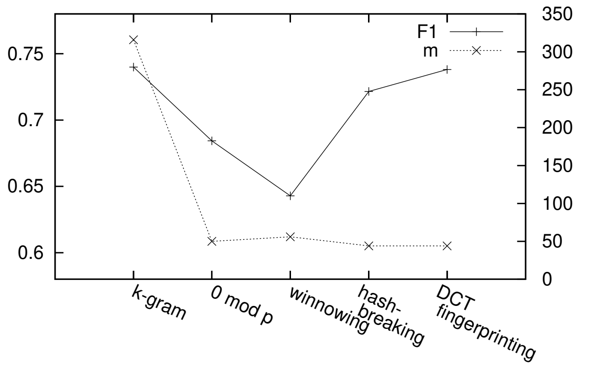

Figure 5: Local text reuse detection performance of fingerprinting techniques. $F1$ and m represents the average of $F1$ of $C3$, $C5$ and $C6$ and the average number of fingerprints of a document, respectively.

newswire collection containing about 758,224 documents as the test collection. The collection is comprised of *Associated Press* (1988-1990), the *Wall Street Journal* (1987-1992), the *Financial Times* (1991-1994) and the *Los Angeles Times* (1989-1990).

Table 4 presents the results of the two techniques. They show almost same results for each type. The fact that DCT fingerprinting detected slightly more documents means that there might be false detections in some degree, but that would not be significant. Further, while the size of the index for k-gram was 1.4 gigabytes, that for DCT fingerprinting was only 239 megabytes. The running time for DCT fingerprinting (1 hour) was also shorter than that of k-gram (3 hours). Thus, we conclude that DCT fingerprinting is comparable in effectiveness and more efficient than k-gram.

Table 4: Text reuse detection results of k-gram and DCT fingerprinting in TREC newswire collection. '#Sibling' represents the average number of documents which are related to the detected document through a category

	k-gram			DCT fingerprinting		
Type	#Doc	%	#Sibling	#Doc	%	#Sibling
$C1$	21087	2.78%	1.68	20173	2.66%	1.48
$C2$	21579	2.85%	1.93	21080	2.78%	1.60
$C3$	9338	1.23%	1.59	8533	1.13%	1.33
$C4$	33448	4.41%	21.96	31477	4.15%	20.67
$C5$	41693	5.50%	4.90	41406	5.46%	3.74
$C6$	99219	13.09%	59.72	101934	13.44%	62.38
Total	171320	22.59%	40.60	170874	22.54%	42.36

6. TEXT REUSE ON REAL COLLECTIONS

In this section, we analyze the amount and type of text reuse in two collections using DCT fingerprinting. We use two metrics to analyze the collections. One metric, the number of documents in each text reuse type, shows how many documents involve text reuse. Another metric is the average number of *siblings*. The siblings of a document represent documents which have text reuse relationships with the document.

6.1 Newswire Collection

News articles are public and official documents written by professional reporters or editors. Text reuse happens most frequently when stories are revised or when journalists "borrow" from earlier stories. The results in Section 5.2 and in Table 4 were based on news collections.

We sampled 50 document pairs for each type from the detection results and manually classified the text reuse into three more classes based on the style of text reuse (rather than the amount of text). These classes are *'Text reuse'*, *'Common Phrase'* and *'Template'*. The results are shown in Table 5.

'Text reuse' patterns correspond to actual text reuse cases. That is, a document pair with these patterns is derived from the same source or has a direct relation. For example, while an article A written on April 4, 1988 describes an event with four sentences, another article B written on April 5, 1988 has six sentences in addition to the four sentences. We can infer an information flow from article A to article B.

'Common phrase' patterns are caused by common phrases. Thus, we might not infer any actual relation. For example, 'Dow Jones average of 30 industrials' and 'the New York Stock Exchange' are commonly used in many articles about the stock market. If two documents share these phrases, we

Table 5: Text reuse in the TREC newswire collection.

Pattern	$C1$	$C2$	$C3$	$C4$	$C5$	$C6$	Total
Text Reuse	64%	68%	100%	0%	48%	6%	48%
Common Phrase	0%	0%	0%	0%	12%	84%	16%
Template	36%	32%	0%	100%	40%	10%	36%

cannot say that they have a text reuse relationship.

The most interesting patterns are *'Template'* patterns, which make up one third of the total detected relationships. In these patterns, there are reusable templates where only a few words are replaced. The following is an example where only highlighted numbers are changed with the remaining words unchanged.

> U.S. automakers were scheduled to make **167,791** cars this week, compared with **170,279** a week ago and **152,751** in the same week in **1987**. [AP880616-0266]

Such template patterns are easily found in the economic news about stock market indexes, foreign exchange rates, or bank interest rates. These patterns are observed in most text reuse types and the class C4 is particularly dominated by these patterns.

Consequently, the news collection has a small amount of text reuse if we exclude $C4$ and $C6$ which are dominated by noisy patterns like *'Common Phrase'* and *'Template'*. That is, to build an accurate text reuse detection system, local text reuse detection which can identify this type of noise is necessary. Further, each document typically has few siblings. That is, the reused text is not likely to move through many documents and the propagation paths are limited.

6.2 Blog Collection

Blogs are generally operated by individuals, in contrast to news organizations. We use the TREC Blogs06 collection [13], which contains about 3 million postings. Since the collection size is about 100 gigabytes, it is difficult to process using the k-gram technique.

When we find text reuse in blog collections, there is a problem to be considered. In most blogs, navigation bars are located on the top or the side of each page and advertisement links like Google AdSense[2] or links to the previous postings occupy the corners of each page. Text in such frames is repeated in most of the postings of a blog. As a result, blog postings could be falsely detected as text reuse relationships even though their actual contents are not related to each other at all. We refer to this as *frame noise*. To remove such noise, we employed a Document Slope Curve (DSC) content selection algorithm [8]. The algorithm plots a curve as follows:

$$DSC[k] = \begin{cases} 0 & \text{if } k = 0 \\ DSC[k-1] + 1 & \text{else if } T[k] \text{ is a tag} \\ DSC[k-1] & \text{otherwise} \end{cases} \quad (9)$$

where $T[k]$ is the k^{th} token in an HTML page. By exploiting the observation that there are fewer HTML tags in content bodies than in the other areas, we regard the lowest slope area of the curve as the content body.

We ran our detection engine using DCT fingerprinting. Despite the collection size, the index size was only 1.3 gigabytes, which can be easily handled in main memory. The run took 10 hours with eight 3.2 MHz CPUs. We also tried to use k-gram for time comparison, for which the index size was 10 gigabytes. In addition, we employed a speed-up algorithm for k-gram, 'SPEX' [1], which filters out unnecessary unique fingerprints while building the index. Nevertheless, processing had not finished after several days of elapsed time because of heavy I/O load.

[2]http://www.google.com/adsense

Table 6: Text reuse detection result in TREC Blogs06 collection. '#Sibling' represents the average number of documents which are related to the detected document through a category.

Type	#Doc	%	#Sibling
$C1$	125241	3.90%	562.16
$C2$	171619	5.34%	612.54
$C3$	166527	5.18%	731.68
$C4$	269064	8.38%	528.34
$C5$	439655	13.69%	688.60
$C6$	450539	14.03%	973.53
Total	675015	21.02%	1749.43

Table 6 shows the text reuse detection result. Many more documents are involved in text reuse relationships compared to the newswire collection. In fact, the numbers were overestimated as we see later.

We sampled 50 document pairs for each type from the results and identified five styles of reuse, i.e., *'Text Reuse'*, *'Common Phrase'*, *'Spam'*, *'Frame'* and *'URL Aliasing'*. The result is shown in Table 7.

In *'Text Reuse'* patterns, text reuse originated from authoritative sources such as news articles or academic papers. This appears more frequently than text reuse based on other blog postings. That is, many bloggers seem to still trust authoritative sources more than blog documents.

Most *'Common phrase'* patterns are composed of boilerplate text, in contrast to the newswire collection. For example, the following paragraph is a representative example of boilerplate text which is located below content text with the highlighted date changed.

> This entry was posted on **Friday, January 13th, 2006** at **12:00 pm** and is filed under XXX. You can follow any responses to this entry through the RSS 2.0 feed.

The boiler plate text is different from *'Template'* patterns in that the text forms a small part of the document, e.g., a header or footer rather than the content of the document and is observed in most postings.

'Frame' patterns correspond to frame noise. Although we preprocessed the collection by using the DSC content selection algorithm, a considerable amount of frame noise still remains. Since this noise is almost evenly distributed over all types, we cannot distinguish it easily by classification.

Another new pattern is *'Spam'*. Spam phrases such as 'free gift' and 'poker casino' tend to be repeated in or between spam postings, and accordingly, they could be detected as text reuse.

Another special pattern is *'URL Aliasing'* which has been reported in near-duplicate studies on Web [20]. While two postings have different URLs, they correspond to the same document. Since their contents are identical, these patterns are observed in only the C1 type.

As you see in Table 7, noisy patterns like *'Frame'* and *'Spam'* account for 50∼70% of each class, which causes most

Table 7: Text reuse in the TREC Blogs06 collection.

Pattern	$C1$	$C2$	$C3$	$C4$	$C5$	$C6$	Total
Text Reuse	16%	20%	20%	6%	12%	18%	15%
Common Phrase	2%	12%	12%	24%	28%	28%	18%
Spam	30%	22%	20%	8%	12%	20%	19%
Frame	36%	46%	48%	62%	48%	34%	46%
URL Aliasing	16%	0%	0%	0%	0%	0%	3%

of the overestimation of text reuse. Therefore, to more accurately investigate text reuse in blog or Web collections, better content selection techniques and spam filtering algorithms are required.

Taking the overestimation into account, the blog collection does not contain more text reuse than the newswire collection. However, the number of the siblings in the blog collection is still greater than that in the newswire collection. This shows that the reused text is easily spread over the blogs.

In addition, *'Text Reuse'* patterns are almost equally distributed over all text reuse types. That is, we need to consider all text reuse types in order to accurately infer relationships between documents. Therefore, for text reuse detection applications like information flow tracking, local text reuse detection is likely to more effective than near-duplicate detection which can detect only a few text reuse types.

7. CONCLUSION

We defined a general framework for text reuse detection. The six categories for text reuse detection can be flexibly applied to various tasks including near-duplicate detection and local text reuse detection.

We reviewed several fingerprinting techniques for the framework and introduced a robust technique, DCT fingerprinting. Through performance comparison and empirical validation, we showed that DCT fingerprinting is one of the best candidates for general or local text reuse detection with high accuracy and efficiency.

Finally, using this algorithm, we investigated the text reuse aspects of a newswire collection and a blog collection. Through the analysis, we showed that the text reuse patterns of the two collections are different from each other and local text reuse detection will be more effective than near-duplicate detection for applications like information flow tracking on such collections.

8. ACKNOWLEDGMENTS

This work was supported in part by the Center for Intelligent Information Retrieval, in part by NHN Corp. and in part by NSF grant #IIS-0534383. Any opinions, findings and conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect those of the sponsor.

9. REFERENCES

[1] Y. Bernstein and J. Zobel. Accurate discovery of co-derivative documents via duplicate text detection. *Information Systems*, 31:595–609, 2006.

[2] S. Brin, J. Davis, and H. García-Molina. Copy detection mechanisms for digital documents. In *Proc. of the 1995 ACM SIGMOD Intl. Conf. on Management of data*, pages 398–409, 1995.

[3] A. Z. Broder. On the resemblance and containment of documents. In *Proc. of the Compression and Complexity of Sequences*, pages 21–29, 1997.

[4] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Comput. Netw. ISDN Syst.*, 29(8-13):1157–1166, 1997.

[5] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. of the 34th Ann. ACM Symp. on Theory of computing*, pages 380–388, 2002.

[6] A. Chowdhury, O. Frieder, D. Grossman, and M. C. McCabe. Collection statistics for fast duplicate document detection. *ACM Trans. Inf. Syst.*, 20(2):171–191, 2002.

[7] B. Coskun, B. Sankur, and N. Memon. Spatio-temporal transform based video hashing. *IEEE Transactions on Multimedia*, 8(6):1190–1208, 2006.

[8] A. Finn, N. Kushmerick, and B. Smyth. Fact or fiction: Content classification for digital libraries. In *Joint DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries*, 2001.

[9] N. Heintze. Scalable document fingerprinting. In *1996 USENIX Workshop on Electronic Commerce*, 1996.

[10] M. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proc. of the 29th Ann. Intl. ACM SIGIR Conf. on Research and development in information retrieval*, pages 284–291, 2006.

[11] T. C. Hoad and J. Zobel. Methods for identifying versioned and plagiarized documents. *J. Am. Soc. Inf. Sci. Technol.*, 54(3):203–215, 2003.

[12] C.-Y. Lin and S.-F. Chang. A robust image authentication method distinguishing JPEG compression from malicious manipulation. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(2):153–168, 2001.

[13] C. Maconald and I. Ounis. The TREC blogs06 collection: Creating and analysing a blog test collection. Technical Report TR-2006-224, University of Glasgow, Department of Computing Science, 2006.

[14] U. Manber. Finding similar files in a large file system. In *Proc. of the USENIX Winter 1994 Tech. Conf.*, pages 1–10, 1994.

[15] D. Metzler, Y. Bernstein, W. B. Croft, A. Moffat, and J. Zobel. Similarity measures for tracking information flow. In *Proc. of the 14th ACM Intl. Conf. on Information and knowledge management*, pages 517–524, 2005.

[16] M. O. Rabin. Fingerprinting by random polynomials. Technical report, Harvard University, 1981. TR-15-81.

[17] R. Rivest. The MD5 Message-Digest Algorithm, RFC 1321, 1992.

[18] S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: local algorithms for document fingerprinting. In *Proc. of the 2003 ACM SIGMOD Intl. Conf. on Management of data*, pages 76–85, 2003.

[19] N. Shivakumar and H. García-Molina. SCAM: A copy detection mechanism for digital documents. In *Proc. of the 2nd Ann. Conf. on the Theory and Practice of Digital Libraries*, 1995.

[20] N. Shivakumar and H. García-Molina. Finding near-replicas of documents on the web. In *Intl. Workshop on the World Wide Web and Databases*, 1999.