DEIM Forum 2009 D8-6

Local Text Reuse Detection の LSH による高速化

能祖　正樹† 　黄瀬　浩一† 　岩村　雅一 †

† 大阪府立大学大学院工学研究科　〒 599–8531 大阪府堺市学園町 1–1
E-mail: †nouso@m.cs.osakafu-u.ac.jp, ††{kise,masa}@cs.osakafu-u.ac.jp

あらまし　Local text resuse detection とは，文書の部分的なコピーを発見する技術であり，文書の盗用検出に有効である．ただし，この処理には，データベースの規模を大きくすると，処理時間が大幅にかかってしまうという問題がある．その原因は，大量のデータに対して最近傍探索を行うためである．最近傍探索にかかる時間を短縮する手法の一つに，近似を導入するものがある．そこで，本稿では近似最近傍探索の一手法である Locality-Sensitive Hashing(LSH) と Spherical LSH を用いて，local text reuse detection の処理を高速化する手法を提案する．

キーワード　local text reuse detection，near-duplicate detection，LSH, Spherical LSH

Speeding up Local Text Reuse Detection by Locality-Sensitive Hashing

Masaki NOUSO†, Koichi KISE†, and Masakazu IWAMURA†

† Graduate School of Engineering, Osaka Prefecture University Gakuen-cho 1-1, Sakai, Osaka, 599-8531 Japan
E-mail: †nouso@m.cs.osakafu-u.ac.jp, ††{kise,masa}@cs.osakafu-u.ac.jp

Abstract　Local text reuse detection is a technology to detect partial copies of documents and useful for anti–plagiarism of documents. A problem of methods for local text reuse detection is how to make the calculation of similarity of text parts efficiently, since it is generally based on nearest neighbor search whose cost is related to the size of the database. In order to solve this problem, we propose a method of local text reuse detection with the help of approximate nearest neighbor search. Methods called Locality-Sensitive Hashing (LSH) and its variant called Spherical LSH are both employed to speed up the processing.

Key words　local text reuse detection，near-duplicate detection，LSH, Spherical LSH

1. はじめに

近年，インターネットの普及により，電子文書を扱う機会が増加している．それに伴い，電子文書の不正コピーも増加している．そのため，電子文書の不正コピーを検出する技術が必要不可欠なものとなっている．

電子文書の不正コピーに適用可能な技術として，near-duplicate detection [1] が挙げられる．これは，文書間の類似度を計算する技術であり，類似文書として不正コピー文書を検出することができる．しかし，この技術で検出できるものは文書全体のコピーであり，文書の部分的なコピーに対応できない．この問題に対処するために，local text reuse detection [2] が提案されている．これは，Near-duplicate detection の対象を文書全体から文書の部分文字列にしたものと位置づけることができ，これにより，文書の部分的なコピーを検出できる．

local text reuse detection では，chunk と呼ばれる文書の部分文字列を一つの単位として照合を行うことが多い．最適な chunk の長さは chunk の照合方式により違ってくる．chunk の完全一致による照合を行う場合，改変部分の影響を受けないように短い chunk を多数作成し，改変部分は周辺の chunk の照合により補完して検出する．一方，chunk の類似度計算によって照合を行う場合，短い chunk を作成してしまうとすべて類似して見えるため，特定性がなくなってしまう．そのため，特定性のある長い chunk を作成する必要がある．また，長い chunk を作成すると文書から作成される chunk の数が減少する．そのため，chunk の完全一致の手法よりもデータベースの規模が小さいという利点がある．一方で，類似度計算には処理時間がかかるという問題点もある．

本研究では，この問題点を解決するために，類似度計算の高速化手法，具体的には，ベクトルの近似最近傍探索を用いる手法を提案する．近似最近傍探索の手法として，Locality-Sensitive Hashing(LSH) [3] や Spherical LSH [4] が挙げられる．LSH では，類似したベクトルは同じ値になるようなハッシュを用いることで，類似したベクトルのみに計算対象を絞りこみ，計算コ

ストを削減する．Spherical LSH では，ベクトル間の角度を利用してハッシュ値を求める．

以下，**2.** では local text reuse detection の高速化に用いる LSH と Spherical LSH について述べる．**3.** では関連研究である Near-duplicate detection や local text reuse detection について述べる．**4.** では local text reuse detection の処理を高速化する手法について提案する．

2. Locality-Sensitive Hashing(LSH)

まず，local text reuse detection の高速化のために用いる LSH と Spherical LSH について述べる．

2.1 LSH

LSH は,「類似したベクトルはハッシュ値が一致する」という特徴を持つハッシュ関数を用いる手法である．d 次元特徴ベクトルを$\boldsymbol{x}$とすると，ハッシュ関数は

$$h_j(\boldsymbol{x}) = \lfloor \frac{y(\boldsymbol{x})}{c} \rfloor \quad (1)$$

$$y(\boldsymbol{x}) = \boldsymbol{a} \cdot \boldsymbol{x} + b \quad (2)$$

で表される．ここで，$\boldsymbol{a}$は正規分布に従ったランダムな d 次元ベクトル，c は量子化パラメータ，b は $b \in [0, c]$ となる乱数である．このハッシュ関数を複数用いた関数

$$g_i(\boldsymbol{x}) = (h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), \cdots h_k(\boldsymbol{x})) \quad (3)$$

を用いて，d 次元特徴ベクトル$\boldsymbol{x}$を k 次元へと射影する．この関数 $g_i(\boldsymbol{x})$ を L 個用意し，各々個別のハッシュ表に対応付ける．検索時には，各ハッシュ表においてハッシュ値が一致した特徴ベクトルの集合の和集合を用意することで，類似した特徴ベクトルのみに対象を絞ることができる．

2.2 Spherical LSH

Spherical LSH では，ランダムな回転行列を用いたハッシュ関数を定義している．回転行列によりランダムに回転したベクトルとの内積を計算し，最大となったベクトルの番号をハッシュ値としている．

Spherical LSH の手順について詳しく述べる．まず下式で定義される M 個のベクトル $\tilde{\boldsymbol{v}}_i$ を用意する．

$$[\tilde{\boldsymbol{v}}_i]_j = \delta_{ij} - \frac{d+1-\sqrt{d+1}}{d(d+1)} \quad (i = 1, 2, \cdots, M) \quad (4)$$

$$[\tilde{\boldsymbol{v}}_{M+1}]_j = \frac{1-\sqrt{d+1}}{d} - \frac{d+1-\sqrt{d+1}}{d(d+1)} \quad (5)$$

ここで，$[\tilde{\boldsymbol{v}}_i]_j$ は i 番目のベクトル $\tilde{\boldsymbol{v}}_i$ の j 番目の座標であり，δ_{ij} は $i = j$ の時のみ 1 となり，それ以外は 0 となる関数である．このベクトル $\tilde{\boldsymbol{v}}_i$ とランダムな d 次元回転行列 $\boldsymbol{A}$ をかけ，ベクトル $\boldsymbol{v}_i$ とする．

$$\boldsymbol{v}_i = \boldsymbol{A}\tilde{\boldsymbol{v}}_i \quad (6)$$

このベクトル $\boldsymbol{v}_i$ と，d 次元特徴ベクトル $\boldsymbol{x}$ の内積を計算し，最大値を得た i をハッシュ値とする．ランダムな d 次元回転行列 $\boldsymbol{A}$ のハッシュ関数 $h_{\boldsymbol{A}}(\boldsymbol{x})$ は以下の式で表される．

$$h_{\boldsymbol{A}}(\boldsymbol{x}) = \arg\max_i (\boldsymbol{v}_i \cdot \boldsymbol{x}) \quad (7)$$

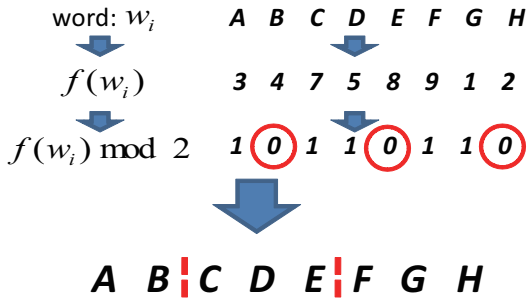

図 1　hashed breakpoint の処理の流れ ($p = 2$)

このランダムな d 次元回転行列 $\boldsymbol{A}$ を複数用いることで，d 次元特徴ベクトル $\boldsymbol{x}$ を k 次元ベクトルへと射影し，その後は LSH と同様の処理を行う．

3. 関 連 研 究

まず，chunk を用いた local text reuse detection の概要を **2.1** で述べる．そして本研究の関連研究として，提案手法で用いる hashed breakpoint について **2.2** で述べ，DCT fingerprinting について **2.3** で述べる．

3.1 chunk を用いた local text reuse detection

chunk を用いた local text reuse detection は，chunk の照合方式によって大きく 2 つの手法に分けることができる．一つ目は，chunk の完全一致による照合を行う手法で，二つ目は chunk の類似度計算によって照合を行う手法である．

chunk の完全一致による照合を行う場合，改変部分の影響を受けないように短い chunk を多数作成し，改変部分は周辺の chunk の照合により補完して検出する．この手法では，短い chunk を多数作成するため，データベースが大規模になるという問題点がある．そのため，様々な手法で chunk を取捨選択し，fingerprint を作成する．作成された chunk 全体の部分集合である fingerprint を用いることで，データベースの規模を現実的な範囲で構築する手法 [5] が提案されている．

chunk の類似度計算によって照合を行う手法では，ある程度の長さをもった chunk を作成する必要がある．これは，短い chunk を用いると様々な箇所を検出してしまい，正確な検出が出来なくなるためである．また，長い chunk を作成することにより，文書から作成される chunk の数を少なくすることができる．そのため，chunk の完全一致による照合手法よりもデータベースの規模が小さいという利点がある．しかし，類似度計算を行うために処理時間がかかるという問題点がある．

3.2 hashed breakpoint

hashed breakpoint の処理の流れを図 1 に示す．まず文書中の単語 w_i をハッシュ関数 f を用いて数値に変換する．次に，

$$(f(w_i) \bmod p) \equiv 0 \quad (8)$$

となる単語 w_i で文書を区切ることで，重複しない chunk を作成し，盗用の検出を行う．chunk の重複がないため，他の chunk 作成手法に比べてデータベースが小規模で済むという利点がある．

しかし，chunk が重複しないため，文書が変化した場合，変

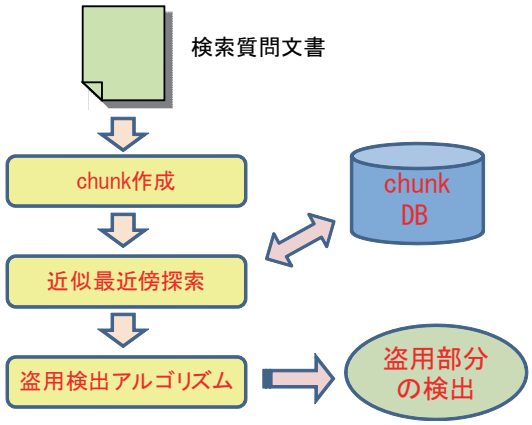

図 2　提案手法の処理の流れ

化した chunk が上手く検出できない可能性がある．この問題を解決するため，パラメータ p を複数用いて違う単語で区切り，chunk を重複させることで変化に頑強にする改良手法 [6] が提案されている．

3.3　DCT fingerprinting

DCT fingerprinting [2] では，まず hashed breakpoint を用いて chunk を作成する．そして，chunk 中の単語を数値へ変換することで，chunk を数値列で表現する．次に，この数値列を平均が 0 になるように各数値から数値列の平均値を減算する．その後，数値列の最大値を用いて正規化を行う．これらの数値列に対し，DCT を適用し，その係数を量子化して fingerprint を作成する．chunk の類似性は fingerprint の一致により判断する．

DCT fingerprinting では，文字列に対し DCT を行い量子化することで改変の影響を受けないようにしている．しかし，DCT fingerprinting が有効なのは，chunk 中の単語が一つ改変された場合であり，単語が二つ以上改変された場合，単語の挿入，削除があった場合には効果が低い手法となっている．そのため，hashed breakpoint のパラメータ p の値を低く設定し，短い chunk を作成することで対応できない種類の chunk の数を出来るだけ少なくするようにしている．

4.　提 案 手 法

4.1　提案手法の流れ

提案手法の処理の流れを図 2 に示す．提案手法は大きく分けて chunk 作成，近似最近傍探索を用いた chunk の検出，盗用検出アルゴリズムの 3 段階で構成される．

まず，あらかじめ多数の文書から chunk を作成して，chunk ベクトルとして表現する．そして，chunk ベクトルに基づいて chunk をインデキシングし，chunk データベースを作成する．検出時には，検索質問文書から同様に chunk ベクトルを作成する．そして，データベースの chunk に対して近似最近傍探索を行い，chunk を検出する．最後に，盗用検出アルゴリズムを用いて，検出されなかった chunk を拾い上げ，盗用部分を抽出する．

以下，chunk 作成，近似最近傍探索を用いた chunk の検出，盗用検出アルゴリズムについて説明する．

4.2　chunk 作成

本研究では，chunk 間の類似度を用いて盗用の検出を行う．そのため chunk の作成手法は，以下の 3 点の性質を満たす手法である必要がある．

- 作成する chunk の数が出来るだけ少ない
- chunk がある程度の長さを持つ
- 文書の改変に対して安定した chunk を作成する

作成する chunk の数を少なくすることで，データベースの規模を小さくすることができ，処理時間を削減することができる．次に，chunk が短ければ特定性がなくなってしまうため，類似度による検出ができなくなる．そのため，chunk にはある程度の長さを持つことが必要となる．最後に，文書の改変に対して安定した chunk とは，単語の入れ替え，追加，削除によって周辺の chunk に影響を与えない chunk のことである．特に，単語の追加や削除によって chunk を区切る単語が変化せず，改変がその chunk 以外に影響しないことが必要となる．

この要求を満たす一手法は，常に一定の単語で文書を chunk に分割し，chunk が重複しない hashed breakpoint である．そのため，本研究では chunk 作成の手法に hashed breakpoint を用いる．

hashed breakpoint により作成された chunk を，ベクトル空間法を用いて chunk ベクトルにする．chunk x_1，x_2 が単語 w_i により，

$$x_1 : w_1\ w_3\ w_4\ w_1\ w_2$$
$$x_2 : w_2\ w_3\ w_5\ w_6$$

と表現されていたとする．全単語が w_1〜w_6 であるとき，各単語の出現回数を各次元の要素としたベクトルで表すと，chunk ベクトル $\boldsymbol{x}_1$，$\boldsymbol{x}_2$ は

$$\boldsymbol{x}_1 = (2,\ 1,\ 1,\ 1,\ 0,\ 0)$$
$$\boldsymbol{x}_2 = (0,\ 1,\ 1,\ 0,\ 1,\ 1)$$

となる．このようにして作成した chunk ベクトルに対し，tf-idf 重みを用いることで重みづけを行う．単語 w_i が chunk $\boldsymbol{x}_j$ に出現した回数を o_{ij}，chunk の総数を N，単語 w_i を含む chunk の数を N_i，インデキシングされた単語数を M とすると，単語 w_i の tf-idf 重みは

$$\boldsymbol{x}_j = (m_1, \ldots, m_M) \tag{9}$$
$$m_i = o_{ij} \cdot \log \frac{N}{N_i} \tag{10}$$

と表わされる．本研究では，この tf-idf で重み付けした chunk ベクトルを用いる．

4.3　近似最近傍探索を用いた chunk の検出

提案手法では，chunk ベクトルに対して近似最近傍探索を行う．その後，定めた閾値以上の類似度を持つ chunk を盗用部分として検出し，盗用検出アルゴリズムへの入力とする．

LSH は，データ点の分布が均一なときに最も性能が良くなるように作成されている．しかし，今回用いる単語の chunk 内で

の分布は偏りの強い分布である．そのため，LSH が本来の性能を発揮できない可能性がある．そのため，通常の LSH に加えて，ハッシュ値の出現頻度が均一になるように式 (1) の量子化パラメータを可変にした手法も用いる．具体的な処理は以下のとおりである．式 (1) の $h_j(\boldsymbol{x})$ を定義する際に，a と b を定めた上で，データベース中の chunk を用いて $y(\boldsymbol{x})$ の分布を計測する．そして，閾値 $t_s(s = 1, \ldots, B-1)$ を用いて値を B 個に量子化し，

$$h_j(\boldsymbol{x}) = \begin{cases} 1 & \text{if} \quad y(\boldsymbol{x}) < t_1 \\ s & \text{if} \quad t_s \leqq y(\boldsymbol{x}) < t_{s+1} \\ B & \text{if} \quad t_{(B-1)} < y(\boldsymbol{x}) \end{cases} \tag{11}$$

のようにハッシュ値を定める．ここで，閾値 t_s は，ハッシュ値 $1, \ldots, B$ の頻度が均一になるように定める．以下では，この手法を LSH' と呼ぶ．

本研究では，近似最近傍探索の手法として上記の手法に加えて Spherical LSH も用いる．これは，特徴ベクトル間の類似度は内積計算によって求められているため，ベクトル間の角度を利用してハッシュ値を求める Spherical LSH を用いることでより効果的に計算対象が削減できる可能性があるためである．

しかし Spherical LSH では，単語数で定まる巨大な回転行列が必要となり，メモリ使用量の観点から現実的でない．そのため，次元削減を行うことで，Spherical LSH を使用可能とする．本研究では，次元削減のために，Latent Semantic Indexing(LSI) [7] を用いる．LSI の具体的な方法について説明する．データベース中の各 chunk ベクトルをまとめた行列を，

$$\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots \boldsymbol{x}_N] \tag{12}$$

とする．ここで，N はデータベースに登録されている chunk の数である．次に，この $\boldsymbol{X}$ を下式のように特異値分解する．

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{13}$$

この処理で得られた $\boldsymbol{U}$ の最初の l 次元の左特異値ベクトルのみから構成される行列を $\boldsymbol{U}_l$ とする．このとき，$\boldsymbol{x}_i$ の l 次元表現 $\boldsymbol{x}^{(l)}$ は以下の式で与えられる．

$$\boldsymbol{x}^{(l)} = \boldsymbol{U}_l^T \boldsymbol{x}^{(l)} \tag{14}$$

この chunk ベクトルの l 次元表現 $\boldsymbol{x}^{(l)}$ に対して，Spherical LSH を用いた．

4.4 盗用検出アルゴリズム

盗用検出アルゴリズムの処理例を図 3 に示す．図 3 の検出された chunk は，類似度計算において検索質問の chunk のうち，閾値以上の類似度を持ったものである．盗用検出アルゴリズムは，chunk 列中の検出された chunk の部分列を基に，盗用部分を抽出する処理を担う．

盗用部分は以下のように抽出される．図 3 に示すように，一般に，検索質問では検出された chunk が点在する．本手法では，このうち連続する検出された chunk 列を盗用部分として抽出する．ただし，ギャップの閾値 e を設け，検出された chunk 列に挟まれた，検出されていない chunk が e 個以下である場合には，それらを連結する．このようにして検出された部分を盗用部分とする．

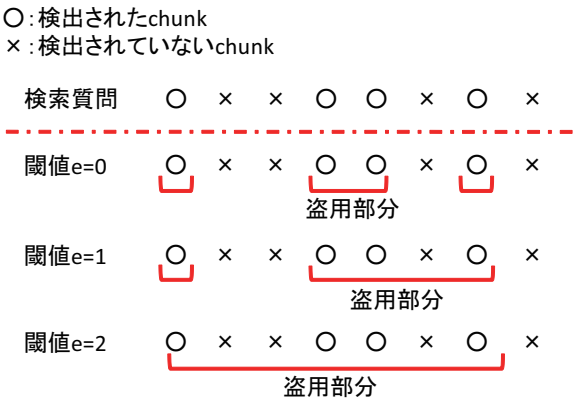

図 3　盗用検出アルゴリズムの例

5. ま と め

本研究では，local text reuse detection の高速化について提案した．本手法は，chunk の作成数を少なくする hashed break point と類似度計算を高速化する近似最近傍探索を用いることで，local text reuse detection の高速化を図るものである．

今後の課題としては，実験を行い，提案手法の有効性を示すことが挙げられる．

文　献

[1] C. Xiao, W. Wang, X. Lin and J. X. Yu, "Efficient similarity joins for near duplicate detection ", 17th International World Wide Web Conference (WWW2008), 2008.

[2] J. Seo and W. B. Croft, "Local text reuse detection ", In Proceedings of SIGIR'2008, pp.571–578, 2008.

[3] M. Datar, N. Immorlica, P. Indyk and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions ", SCG '04: Proceedings of the twentieth annual symposium on Computational geometry, ACM, pp.253–262, 2004.

[4] K. Terasawa and Y. Tanaka, "Spherical lsh for approximate nearest neighbor search on unit hypersphere ", 10th Workshop on Algorithms and Data Structures, WADS2007, LNCS 4619, pp.27–38, 2007.

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

[6] M. Pataki, "Plagiarism detection and document chunking methods ", World Wide Web Conference Series, 2003.

[7] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer and R. Harshman, "Indexing by latent semantic analysis ", Journal of the American Society of Information Science, pp.391–407, 1990.