2C3-02

汎用検索手法の高速化と実適用

Efficient Algorithm of Generic Search Method and its Application

足立史宜[*1] Fuminori Adachi　鷲尾 隆[*1] Takashi Washio　藤本 敦[*1] Atsushi Fujimoto　元田 浩[*1] Hiroshi Motoda　丹羽 雄二[*2] Yuuji Niwa　花房 英光[*2] Hidemitsu Hanafusa

[*1]大阪大学 産業科学研究所 I.S.I.R., Osaka University　[*2]原子力安全システム研究所 INSS Inc.

The needs of efficient and flexible search engines for the multimedia contents stored in computer networks are growing significantly. However, each existing approach is dedicated to each content and file format. In this work, we propose an efficient and generic search method which is directly applicable to various types of contents. The power of this approach comes from the use of the generic and invariant feature information obtained from byte patterns in the files through a mathematical transformation in efficient manner. The experimental evaluation of the proposed approach for both artificial and real data indicates its high feasibility.

1. はじめに

現状の検索技術は，大部分がテキスト文書などの特定データ形式に特化している [1],[2],[3]．そのため，文書や写真，音声などが交じるマルメディアコンテンツを直接検索するには，それぞれに対応した検索エンジンを組み合わせる必要がある [4],[5]．しかし，これでは各検索エンジンの開発，運用コストや手間が大きくなる．また，新種のフォーマットや，フォーマット仕様が未公開のコンテンツの検索も困難である．

一方，如何なるファイルも，ビットやバイトを単位とする二進数で表現されている．従って，このような低レベルの表現形式上で類似したビットやバイトのパターンを検索できれば，コンテンツ形式に依存しない検索が実現される．この際，コンテンツレベルでの類似性に対応した低表現レベルでの類似性を考える必要がある．両レベルにおける類似性を完全に一致させることは一般に困難だが，本研究では以下の点を考慮する．

(1) 特定のバイトデータがある順序，ある長さで連なっている共通部分を見出す．
(2) バイト系列がほぼ共通した順序波形部分を見出す．
(3) バイト系列中の位置に影響を受けずにほぼ共通した系列部分を見出す．
(4) バイト系列中に混入するノイズや一部の値の違いによる影響を軽減しつつ共通部分を見出す．
(5) ファイル全体に亘りバイト系列のほぼ共通部分を見出し，その個所の多寡で類似性を判定する．
(6) 大多数のファイルに頻出する共通部分は，類似性を特徴付ける共通部分から除外する．

昨年提案した手法では上記を考慮しつつ，テキストデータやワープロデータ，音声データなど，ファイル内でのコンテンツ情報の関連配置を１次元系列として扱うことが可能なデータに関し，低表現レベルでの類似性に基づく汎用検索手法を提案する．そのような類似性を高精度, 高効率に判定するため，１次元バイト系列にある種の数学的変換を適用して系列の有する不変的特徴を抽出し，その特徴の効率的類似性判定からファイル情報を検索した．しかし，昨年のシステムでは後述する逆引き情報の作成に時間がかかりすぎるため実用的なシステムであるとは言い難かった．そこで逆引き情報作成に関する処理を高速化し，システムの改良を図った．

連絡先: 足立史宜 大阪大学 産業科学研究所 567-0047 茨木市美穂ヶ丘 8-1 Tel (06)879-8542 Fax (06)879-8544
e-mail: adachi@ar.sanken.osaka-u.ac.jp

2. 汎用検索手法の概要

以下にこれまでに提案した汎用検索手法の概略について説明する．上記のバイト系列類似性判定の考慮点の中で (1) と (2) を達成するためだけであれば，各ファイルをバイト系列と見なし，各バイトデータ毎に完全に一致するか，ないしは十分に近い値で一致するか否かを判定すれば十分である．しかしながら，(3) の点に対応するためには，バイト系列上の位置に依存しない照合が必要となる．与えられた２つのバイト系列間におけるこのような条件を満たす最も単純な照合は，あらゆる長さの部分系列を２つのバイト系列のあらゆる場所から取り出して比較する方法である．しかし，この方法では各バイト系列の長さを n_1, n_2 としたとき，おおよそ $O(n_1^2 n_2^2)$ のオーダーの比較時間がかかり実用的でない．そこで，本研究ではバイト系列を数値系列と見なし，それにある種の数学的変換を施すことでバイト系列の位置に依存しない共通部分の特徴を抽出，照合する．系列内の位置に依存せずに特徴を抽出するためには，系列順序方向への”シフト不変性”を有する数学的変換を用いることが必要である．また (4) のようにノイズの影響を軽減するためには，バイト系列内のデータの一部置換や交換などによって生起する局所的な系列データや順序波形の変化が，変換結果の一部にしか影響を与えない数学的変換を用いることが望ましい．更に十分実用的な検索速度を保証するためには，高速な変換アルゴリズムが確立された数学的変換を用いる必要がある．このような一連の条件を満たすものとして，本研究では高速フーリエ変換 (FFT) を用いる [8]．FFT は変換すべきバイト系列長を n とした時 $O(n \log n)$ の計算時間しか必要とせず，十分実用的な計算処理を行うことが可能である．また，係数の絶対値の対称性から係数の個数を約半分に圧縮できるメリットもある．ただし，テキストデータやワープロデータにおける語句や音声データにおける発音のような類似性判定に意味を持つデータ長に対して，あまりに長すぎるバイト系列に FFT を適用すると，個別の類似部分の特徴が他の部分の特徴と重畳して判定できなくなる恐れがある．そこで，各データファイルから細切れに一定長のバイト系列を切り出し，それぞれに FFT を適用してファイル

各所の特徴を表す係数絶対値ベクトルを得ることとする．

また，置換などのノイズに関しては数学的変換以外にも変化を吸収する工夫が必要である．そこで，ノイズの影響の吸収や音声データの順序波形のように完全一致しない共通部分の判定，更に照合高速化のために，FFT 係数の量子化を行う．本研究では汎用な検索手法を目指すため，どのようなデータファイルに対しても有効な各係数絶対値の量子化を行う必要がある．しかしながら，検索対象となる未知のデータファイルのバイト系列から導かれるフーリエ変換係数絶対値の分布を予め知ることはできないため，n バイト系列の全ての 2^{8n} パターンのデータに関して各係数絶対値分布を計算し，その分布の下で m 個に量子化された値が等確率で出現するように各係数絶対値の量子化閾値を決めることとした．このような全分布に関する量子化が粗すぎると，実際のデータファイルから得られるバイト系列のパターンには一般に偏りがあるため，大半の係数絶対値が特定少数の量子区間に集中してしまい，各バイト系列の特徴が明確にならない．そこで本研究では，各係数絶対値毎に詳細な $m = 16$ 段階の量子化を施した．これにより８バイト系列は f_0〜f_4 までの各係数絶対値が量子化された，５次元の１６進数からなる特徴ベクトルに圧縮される．

更に，データファイルの先頭から最後まで固定長バイト系列（この例では８バイト長）を１バイトずつずらして切り出す固定長移動窓方式でデータを区分し，それぞれについて上記の圧縮特徴ベクトルを得ることで，一層ノイズに対する耐性や類似性判定能力を向上させる．このようにして各データファイルから多数の特徴ベクトルが導かれるが，それらの中には多くのデータファイルに共通して現れる特徴ベクトルも存在する．そのような特徴ベクトルは前節の最後 (6) に述べたように，データファイルを特徴付けるものではなく検索絞り込みにはあまり役に立たないため，無効ベクトルとしてリストアップし，検索段階において必要に応じて検索対象データから除外する．無効ベクトルを決めるためには，全データファイルに対して現れる割合を無効ベクトル閾値として指定する必要がある．

さらに，各データファイルについて得た大量の特徴ベクトルから高速に類似したファイルを検索するためには，特徴ベクトル情報の格納データ構造を工夫しなければならない．ここではテキストキーワード検索で大量情報の高速検索に用いられる逆引きファイル手法を用いる [3]．前述の処理によって各データファイルについて，”データファイル名 → 特徴ベクトル集合 ”が得られる．この情報から逆に各特徴ベクトルについてそれが導かれるデータファイル名をリストアップし，”特徴ベクトル名 → データファイル名集合 ”の情報をまとめる．この情報を格納したファイルを逆引きファイルという．予め逆引きファイルを作成しておくことによって，検索時には実データファイルを参照することなく，特徴ベクトルから即座にそれを共通部分として含むデータファイル名を知ることができる．

図 1 に検索システムの概要を示す．はじめに被検索対象データファイルに関する黒矢印で示される処理について説明する．データ抽出部では，被検索対象データファイルから１バイトずつずらした移動窓によるデータ系列の切り出しを行う．そして，数学的変換部において切り出されたデータ系列に対して FFT を施す．更に変換ベクトル量子化部において，係数絶対値ベクトルを量子化し特徴ベクトルを作成する．ベクトル集計部では，１つのデータファイルから得られる特徴ベクトルの中に同じものが多数含まれるため，重複ベクトルを１つにまとめて ”データファイル名 → 特徴ベクトル集合 ”の対応中間ファイルを得る．最後に逆引き情報作成部で中間ファイルから ”特徴ベクトル名 → データファイル名集合 ”の逆引きファイルを得る．検索時には検索キーとなる事例ファイルやキーワードファイルに対して，点線矢印で示すようにベクトル集計部までは被検索対象ファイルに対するものと全く同じ処理を行い，キーファイルに関する特徴ベクトル集合を得る．そしてこの中から無効ベクトル除去部において検索絞り込みに有効ではない無効ベクトルを除去し，残った特徴ベクトルの各々についてベクトルマッチング部において逆引きファイルの内容から対応するデータファイル名をリストアップする．この際，１つでも特徴ベクトルがキーファイルと共通するデータファイルが全てリストアップされてしまうが，検索結果出力部では特徴ベクトルのうち一定閾値割合以上が共通しないと類似したファイルと見なさないとして判定足切りを行う．また，一致する割合が多いものを検索結果の上位に出力する．

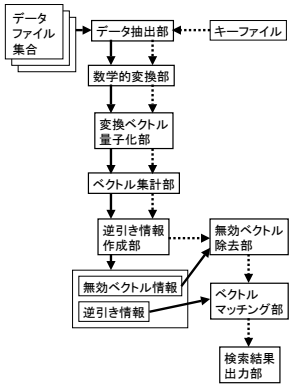

図 1: 検索システムの概要

3. 手法の改良

前節では提案手法の簡単な説明を行ったが，これを計算機に実装する場合においてはできるだけ高効率かつ高速に実行できるようにする必要がある．しかし，初期に作成したシステムでは図 1 の逆引き情報作成部の処理に非常に多くの時間がかかるため，非実用的なシステムと言わざるを得なかった．そこで，逆引き情報作成の高速化のために，プログラムの実装面で以下の点について大幅な改良を行った．

(1) ファイル入出力の効率化

(2) FFT における冗長演算の削除

(3) 逆引き情報作成アルゴリズムの改良

まず，(1) は検索対象ファイルから特徴ベクトル集合を作成する際のファイルの読み込み回数と中間ファイルの出力や逆引き情報が書かれたファイルの書き出し回数を最小限にすることで入出力待ちによるロスを減少させ，効率的に処理を実行できるようにした．例えば，データ抽出部において検索対象ファイルや事例ファイルのデータ系列から特徴ベクトルを作成する際に，重ね移動窓を用いて固定長のデータ系列を切り出すが，このときファイル全体をメモリ上に読み出し，そこから固定長のデータを切り出すことでファイル読み出し回数を最低限に抑えている．次に (2) であるが，8 点 FFT の演算は図で表すと図 2 のようになる．図から分かるとおり，8 点 FFT

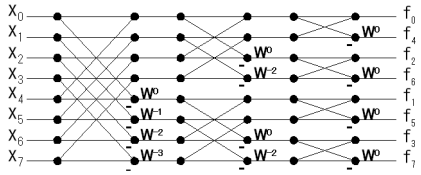

図 2: 8 点 FFT のバタフライ演算

においては位相項 $W^{-n}(n = 0, 1, 2, 3)$ が頻繁に現れる．ここで，

$$\exp\left(j\frac{2\pi n}{N}\right) = \cos\left(\frac{2\pi n}{N}\right) + j\sin\left(\frac{2\pi n}{N}\right) \quad (1)$$

であるが，計算機で cos 及び sin の演算を行うのは加算や乗算に比べて時間がかかる．そこで，FFT ルーチンの初回呼び出し時にのみ $\cos\left(\frac{2\pi n}{N}\right)$ 及び $\sin\left(\frac{2\pi n}{N}\right)(n = 0, 1, 2, 3)$ の計算を行ってメモリ上に記録しておき，次回以降の計算時にはそれを参照するようにして計算がより高速に行えるようにしている．また (3) については厳密に特徴ベクトル間の大小関係を定義し，ベクトル集計部で中間ファイルに書かれる特徴ベクトルがその大小関係の下で昇順にソートされているように出力する．そうすることで，ある特徴ベクトル V_i についての逆引き情報ファイルを作成する場合に，中間ファイルが V_i を含むかを調べる為に中間ファイルの先頭の特徴ベクトルから初めて V_i より大きくなるベクトルまでを走査するだけで良く，無駄な比較を減らすことができる．また,「小さい」特徴ベクトルから順番に逆引き情報を作成すると，以前に逆引き情報ファイルを作成した特徴ベクトルで走査した部分を調べる必要がなくなるので，さらに無駄な比較を減らすことができ，結果的に全体の処理時間の短縮につながる．具体的には以下の処理を実行している．

(0) 前準備として，それぞれの中間ファイルに対して中間ファイルに書かれている特徴ベクトルの数だけの配列と，1 つの状態フラグ，1 つの変数を作成し，その値としてすべて 0 を格納する．さらに，結果を格納する配列 R を作成する．ここでファイルの識別番号が i である検索対象ファイルの中間ファイルに対する配列の j 番目の値を N_{ij}，j 番目に書かれている特徴ベクトルを V_{ij}，状態フラグの値を f_i，変数は検索開始位置を格納するものであり，その値を p_i とする．また，検索対象ファイルに付けられた識別番号の値の最大値を M とする．

(1) $r = 0, R \leftarrow \{0\}$ とする．

(2) $s = \min\{j|N_{rj} = 0\}$ となる s を求る．ただし，そのような s が存在しない場合は (5) へ進む．そして，$M \geq t > s$ となるすべての t のそれぞれに対して $u = \min\{z|z \geq p_t \wedge V_{rs} \geq V_{tz}\}$ となる u を求め，(3) の処理を行う．

(3) V_{rs} と V_{tu} の値によって次の 3 つの処理に分岐する．

1. $V_{rs} \geq V_{tu}$ **となるものが存在しない場合**　特徴ベクトルは昇順にソートされて格納されているので，識別番号 r の中間ファイル中の Vrs より後に書かれている特徴ベクトルは識別番号 t の中間ファイルには存在しないことがわかるので，$f_t = 1$ とし，以降，r の値が変わるまで特徴ベクトルの検索を行わない．
2. $V_{rs} = V_{tu}$ **の場合**　この場合は，V_{rs} が識別番号が t のファイルに存在する．よって，R に $\{t\}$ を加えて $R \leftarrow R \cup \{t\}$ とし，チェックが完了した印である N_{tu} を 1 に設定する．また，特徴ベクトルは各中間ファイルで昇順に並んでいるので $V_{tu} = V_{rs} < V_{rs+1}$ であるから，次回以降の検索では V_{tu} 以前のベクトルを検索する必要がないので検索開始位置を $p_t = u + 1$ に更新する．
3. $V_{rs} > V_{tu}$ **の場合**　この場合は，V_{rs} が識別番号 t のファイルに存在しない．また，特徴ベクトルは各中間ファイルで昇順に並んでいるので $V_{tu-1} < V_{rs} < V_{rs+1}$ であるから，次回以降の検索では V_{tu-1} 以前のベクトルを検索する必要がないので検索開始位置を $p_t = u$ に更新する．

(4) $M \geq t > s$ となるすべての t に対して (3) の処理が終了した後，V_{rs} に対する逆引きファイルとして，R の要素を列挙したファイルを作成する．このとき，R の要素数が無効ベクトル情報作成の閾値以上の場合は無効ベクトル情報に V_{rs} を追加する．さらに，$N_{rs} = 1, R \leftarrow \{r\}$ とし，(2) に戻る．

(5) 識別番号 s の中間ファイルに対する状態フラグを $f_s = 1$ とし，s より大きい識別番号を持つ中間ファイルの状態フラグを 0，検索開始位置を 0，$r = r+1$，$R \leftarrow \{r\}$ として (2) に戻る．ただし，$R = M$ のときは終了する．

これら改良により，逆引き情報の作成にかかる時間が大幅に短縮された．次の表 1 は検索対象ファイルの数を変えた時の逆引き情報作成に要する時間を表したものである．また，この評価に使用したファイルの平均サイズは約 30KB である．

表 1: ファイル数を逆引き情報作成時間の関係

ファイル数	10	100	1000	2000	5000	500(旧システム)
処理時間 (分)	0.5	7	109	366	836	120 時間以上

表 1 より改良前のバージョンとの比較で相当の実行速度の向上があったことがわかる．また，同様に表より逆引き情報の作成にかかる時間はファイル数に対しておおよそ $O(n \log n)$ となっておりある程度実用に耐えうる速度になっていると言える．

4. MS ワードファイルに対する実適用評価

使用したデータセットの元となるのは，マイクロソフトワードの DOC 形式のファイル 2253 個で，平均サイズが約 20KB であり，各ファイルにはおおよそ 600 文字の文章が書かれている．ただし，使用したデータセットはファイル間の類似性検索能力を評価するために次の方法で元のファイルのバイト列に変更を加えたものである．まず，2253 個の中から，無作為に種となる 1 つのファイルを選び，このファイルの番号を 1 とする．次に別のファイル X を無作為に選び，16 文字からなる部分文字列を無作為に切り出す．そして，それをファイル番号 1 の文字列から無作為に選んだ 16 文字からなる部分文字列に上書きし，そのファイルをファイル番号 2 とする．同様に，ファイル番号 n の 16 文字を無作為に選んだファイルから無作為に取り出した 16 文字の部分文字列で上書きし，ファイル番号 $n+1$ として保存する操作を 2253 回繰り返す．
このように種となるファイルを徐々に変更していく．こうすることで，ファイル番号の近いファイルがある程度の類似性を持つような 2253 個のファイルセットが作成されるので，これを用いて類似性検索能力の評価を行っ

た．このデータセットに対して逆引き情報を作成し，任意に選んだファイルをキーファイルとして与え，提案手法を用いて検索を行った結果が次の表 2 である．

表 2: ファイル数を逆引き情報作成時間の関係

No.100	No.500	No.1000	No.1500	No.2000
100	500	1000	1500	2000
102	676	789	1499	2001
99	664	979	1494	1999
104	508	648	1498	1995
96	554	999	150	2158
97	503	967	1497	2258
105	579	997	1496	1868
106	561	856	1503	2019
103	543	852	1504	1989
98	485	543	1506	1877
Std. 17.0	Std. 142.4	Std. 176.4	Std. 190.0	Std. 108.2
$\chi^2 =$ 1352	$\chi^2 =$ 316	$\chi^2 =$ 256	$\chi^2 =$ 1765	$\chi^2 =$ 385
0.642 sec	0.466 sec	0.422 sec	0.844 sec	0.370 sec

表 2 は特徴ベクトルマッチングの観点から，類似度の高い順に上位の 10 ファイルを列挙したものである．この結果から，キーとなるファイルの番号と近い番号をもつファイルが検索結果として現れていることは明らかである．また表の下から 3 行目の欄は検索結果の上位 50 ファイルのファイル番号の標準偏差を表しており，下から 2 行目の欄は 50 ファイルをランダムに取り出したときのファイル番号の分布の期待値と提案手法で得られた上位 50 ファイルの分布から χ^2 値を求めた結果を示している．1 から 2253 を等間隔に 50 個の区間に分けると，50 ファイルをランダムサンプリングしたときに各区間に含まれるファイル数の期待値は 1 であり，このとき χ^2 値は自由度 49 の χ^2 分布に従う．ここで χ^2 値が 94.6 以上であるとき，検索結果がランダムサンプリングである確率は 0.0001 以下であるから提案手法で検索した結果は有意にキーファイルの周りに分布していると言え，ファイル番号の近いものが共通文章を多く含むことを考慮すると，提案手法は十分な類似ファイル検索能力を有していると言える．さらに，表の一番下の欄は検索に要した時間を表している．各々のキーファイルに対して検索時間は 1 秒以内になっており検索時間に関しても十分な実用性を持っていると言える．

5. 関連研究と考察

完全マッチング検索手法の中で，検索対象データファイルを固定バイト系列長の移動窓に区切って変換を行う方法としては，これまでに特徴ファイル法が提案されている [2]．これは切り出した各バイト系列をハッシュ関数を用いて不完全不可逆圧縮し，より少量のバイナリデータ上でキーパターンマッチング絞り込みを行う手法である．ただし，不完全マッチングとなるため，圧縮データ上でマッチングしたデータファイルが本当にキーパターンを含むかどうかを直接データマッチングで確かめる必要がある．一方，Namazu に代表されるように，何れのキーパターンが何れのファイルに含まれているかを示す逆引きファイルを予め作成して高速検索を実現する逆引きファイル法は非常によく用いられている [3], [9]．逆引きファイルは一般に巨大になるが，最近の補助記憶装置の容量増大に伴い問題視されなくなりつつある．これらは直接キーパターンの逆引きファイルを用いるものであるため，テキストファイルなどの完全マッチング検索を対象とするものである．

本提案手法は特徴ベクトルの作成において，特徴ファイル法の固定バイト系列長の移動窓切り出しやハッシュ変換関数を適用する点と類似した処理を含むが，最大の違いは単なるハッシュ圧縮関数ではなく，バイト系列の有する特徴をよく保存しつつ圧縮を行う変換不変性を有する数学的関数を用いる点である．これにより特徴ベクトルの一致割合の閾値を高く取れば完全マッチング検索を実現でき，従来の特徴ファイル法と直接データマッチング法を組み合わせた場合と等価な機能を１つの手法で実現できる．更に閾値を低く取ることで，バイト系列の順序波形が類似しているような不完全類似性マッチング検索も同等のアルゴリズムと処理速度で実現可能である．また，同時に逆引きファイルによる処理も併用するため，逆引きファイル法に近い高速な完全マッチング検索に加え，同様に高速な不完全類似性マッチング検索を実現可能である．

6. おわりに

本研究では，テキストファイルやワープロファイル，音声データファイルなどのような１次元的な順序情報が意味を持つデータファイルを対象として，汎用な類似性検索を行う手法を提案した．この手法は，従来手法が有する機能的特徴の多くを網羅しかつ高速動作が可能である．また２次元データである画像に対する適用も現在進めている．今後はさらに複雑なデータ構造がまとまった意味を有するデータファイルに対しても，適応可能なように拡張を図る予定である．

参考文献

[1] Baeza-Yates, R.A.: String Searching Algorithms, Information Retrieval, Data Structures & Algorithms, Chapter 10, ed. Baeza-Yates, R.A., New Jersey: Prentice Hall, pp.219–240 (1992)

[2] Faloutsos, C.: Signature Files, Data Structures & Algorithms, Chapter 4, ed. Baeza-Yates, R.A., New Jersey: Prentice Hall, pp.44–65 (1992)

[3] Harman, D., Fox, E. and Baeza-Yates, R.A.: Inverted Files, Information Retrieval, Data Structures & Algorithms, Chapter 3, ed. Baeza-Yates, R.A., New Jersey: Prentice Hall, pp.28–43 (1992)

[4] Ogle, V.E., Stonebraker, M.: Chabot: Retrieval from a Relational Database of Images, IEEE Computer, Vol. 28, No. 9, pp.1–18 (1995)

[5] Faloutsos, C., Equitz, W., Flickner, M., Niblack, W., Petkovic, D., Barber, R.: Efficient and Effective Querying by Image Content, Journal of Intelligent Information Systems, 3, 3/4, pp.231–262 (1994)

[6] Fox, C.: Lexical Analysis and Stoplists, Data Structures & Algorithms, Chapter 7, ed. Baeza-Yates, R.A., New Jersey: Prentice Hall, pp.102–130 (1992)

[7] Salton, G. and McGill, M.J.: Introduction to Modern Information Retrieval, McGraw-Hill Book Company (1983)

[8] 電子通信学会編. ディジタル信号処理 第１０版. 技報堂, pp.49-61 (1983)

[9] http://www.namazu.org/