シグナチャ法と曖昧検索を用いた文書検索システム

増井俊之
シャープ株式会社 技術本部
情報技術開発センタ
masui@shpcsl.sharp.co.jp

1997 年 5 月 23 日

概要

テキスト検索におけるシグナチャ (signature) 法とは、テキスト及び検索パタンの特徴のビット列表現 (シグナチャ) のマッチングにより高速に必要テキストを抽出する方法であり、全文探索よりも高速でかつ巨大インデクスを必要としないという特長を持っている。シグナチャ法と曖昧検索コマンド**agrep** を併用した小規模テキスト検索システムを**Nemacs** 上に構築した。本システムは (1) インデクス作成が高速、(2) インデクスサイズが小さい、(3) 検索が高速、という特長を持っており、また若干綴りの違う単語を含むテキストも正しく検索を行なうことができる。多量のメール及びニュースの検索に適用した結果、**egrep** を使用した場合の数倍の検索速度を得ることができた。

1 はじめに

ハードディスクや磁気ディスクの普及にともない大量の文章が個人用またはオフィスにおける UNIX ワークステーション上で扱われるようになってきている。標準 UNIX においては文章内容を検索するには**grep** 類のコマンドを使用するのが一般的であるが、文章の数が多くなると**grep** よる全文探索方式は非常に時間がかかるため、検索用のなんらかのインデクスを作成し使用することが望ましい。例えば NeXT の Librarian においては検索される全文書の全単語に対してあらかじめインデクスを作成し、それを参照しながら検索が行なわれるようになっている。インデクスの使用により検索は高速に行なわれるようになるが、文章中の全単語に関してインデクスを作成するとインデクスの大きさが文章の大きさと同程度になってしまい、ディスクを豊富に持たない環境では大きな負担となる。適当な圧縮を行なうことによりインデクスを数分の 1 のサイズにすることもできるが [3]、これは静的なデータベースに対しては有効なものの、多量のファイルが動的に作成 / 消去される環境ではインデクス作成のオーバヘッドが大きいため有効ではない。このため、個人やオフィスにおける小規模な環境では全文探索方式と全単語インデクス方式の間に位置する検索方式を採用することが望ましいと考えられる。

以上のような要求にこたえるためシグナチャ法 [4] という検索方式が提案されている。シグナチャ法に曖昧検索を併用した文書検索システムを**Nemacs** 上に試作し良好な結果を得ることができた。第 2章においてまずシグナチャ法の簡単な解説を行なう。第 3章で本システムで用いた曖昧検索方式に関して解説する。第 4章では今回作成したシステムの構成及び使用例を示す。第 5, 6章において評価及び今後の課題について述べ、第 7章で結論を述べる。

2 シグナチャ法

2.1 シグナチャ法概要

シグナチャ法とは、検索の対象となる文書 (以下 “テキスト” と表記) 及び検索文字列 (以下 “パタン” と表記) の「特徴」をビット列表現したもの (これをシグナチャと呼ぶ) のマッチングにより高速に必要テキストを抽出する手法である。シグナチャは簡単にはテキストまたはパタンに含まれる単語から計算される。テキストまたはパタン中の各単語に対しハッシュを計算し、その値で示されるビット位置だけを “1” としたビット列をその単語のシグナチャとし、全単語のシグナチャの論理和をテキストまたはパタンのシグナチャとする。あらかじめ検索される全テキストに対してシグナチャを計算しておき、検索パタンのシグナチャが完全にテキストのシグナチャに包含されるときテキストがパタンにマッチしたと判断する。

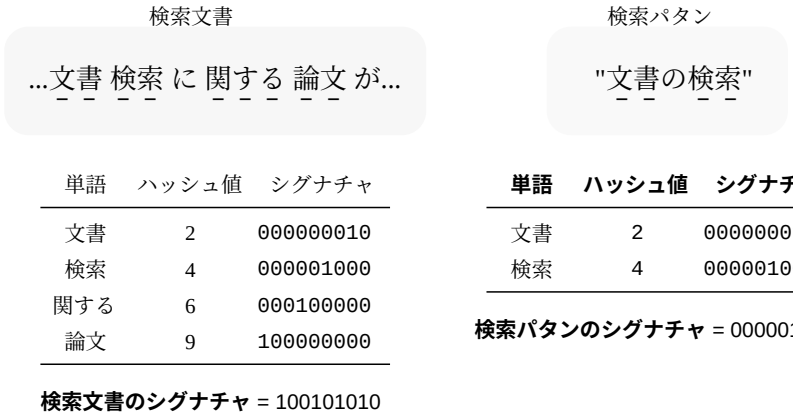

図 1: シグナチャの例 – この例のように検索パタンのシグナチャがテキストのシグナチャに包含されているときテキストはパタンにマッチしていると判定する。

シグナチャにより非マッチと判定されたテキストはパタン中の単語を含むことはないが、同じシグナチャを持つ単語は複数個存在する可能性があるため、シグナチャによりマッチと判定されたテキストが必ずしもパタンとマッチしているとは限らない。このためシグナチャテキストが本当にパタンにマッチしているかどうかを後処理としてもう一度判定する必要がある。

2.2 シグナチャ法の特長

シグナチャ法は以下のような特長を持っている。

- 検索が高速

  テキストのシグナチャとパタンのシグナチャのマッチングの計算は単純な論理計算であるため高速である。またシグナチャの計算法をうまく選ぶことにより、不要なテキストの大部分をシグナチャによる判定によりふるい落とすことができるため、後処理を含めても全体的に高速な検索をすることが可能である。

- シグナチャファイルの作成 / 更新が高速

  各テキストファイルのシグナチャは連結してひとつのファイル (シグナチャファイルと呼ぶ) に格納しておくことができる。テキストファイルの内容が変化したり追加されたりした場合シグナチャファイルも更新する必要があるが、シグナチャファイルは構造が単純であるため簡単にそのファイルの部分だけを入れかえて更新することができる。全単語にインデクスをつけるシステムにおいては、検索を高速にするためトライ (trie) や B-tree 等の特殊な木構造を使用するのが普通であるが、これらの構造を使用すると追加や削除に一般に時間がかかる。

- 余分に必要なファイルが小さい

  シグナチャ法による検索において余分に必要となるのはシグナチャファイルのみである。各テキストファイルのシグナチャはテキストファイル本体に比べてかなり小さくてすむため、全単語のインデクスを使用する方法に比べると余分なファイルが小さくてすむ。

以上のようにシグナチャ法は全文探索法とインデクス法の中間の特徴を持っており、個人またはオフィスにおける中規模の情報検索に適している。

2.3 シグナチャの計算法

各単語のシグナチャは 2.1節で述べたようにハッシュ関数を使用してランダムに計算するのが簡単であるが、次章で述べるように、類似した単語から同じシグナチャが計算されようになっているとさらに都合がよい。単語の類似性はその読みや意味から判定できる。

2.4 単語の切り出し

シグナチャの計算のためにはテキストまたはパタンから単語を正しく切り出すことが必要である。日本語のべた書きの文章からの単語の切り出しは完璧に行なうことは難しいが、辞書をトライ (trie [1]) で表現することにより簡単に充分な精度で単語を切り出すことができる。トライとは辞書を効率良く検索するための木構造で、例えば {“tail”, “talk”, “tex”, “vi”, “view”} で構成される辞書は以下のように表現される。

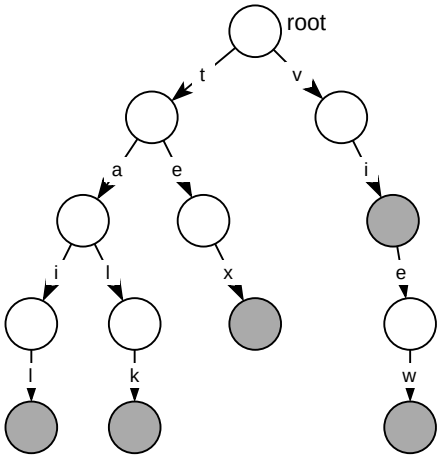

図 2: “tail”, “talk”, “tex”, “vi”, “view” を表現する辞書トライ

このように、あらゆる辞書エントリが木構造中のノードで表現され、木の根からそのノードまでの枝に対応する文字を並べた文字列がそのエントリの表現する文字列となるようなデータ構造をトライと呼ぶ。図 2では灰色のノードがそれぞれの辞書エントリを示している。トライを根から順にたどることにより、文字列が辞書に含まれるかどうかを線型時間で判定することができる。

単語切り出しのアルゴリズムは以下のものを使用する。これはテキストと辞書を比較し、テキストの先頭からの文字列で辞書中に含まれる最長のものを捜すようになっている。例えば “viewpoint” というテキストを図 2の辞書で解析した場合 “vi” でなく “view” が単語として認識され、 “vice” というテキストを解析した場合は “vi” が単語として認識される。ここではひらがなはキーワードとして扱わないことにしている。

```
p ← root;
while((c ← getch()) ≠ EOF){
        if (文字 c がひらがな) continue;
loop:
        if (p が c に対応する子を持つ){
                p ← c に対応する p の子;
        }
        else {
                if (文字列が辞書中に存在){
                        解析中の文字列を単語して認識し、
                        シグナチャの計算に使用する;
                }
                if (p ≠ root){
                        p ← root;
                        goto loop;
                }
        }
}
```

図 3: 文字切り出しアルゴリズム

[1] “Trie” とは “retrieval” から作られた造語である。

図 3のアルゴリズムはバックトラックを行なわないのでテキストの大きさに比例した実行時間しか必要としない。

トライは辞書の表現として優れていることに加え、データ構造を工夫することによりそのサイズを圧縮することが可能である [1][5]。辞書の表現形式としては一般にハッシュがよく使用されているが、今回のような用途に対してはアルゴリズムの点においてもサイズの点においてもトライの方が適している。

単語切り出し用の辞書として Wnn の pubdic と SKK 辞書をマージしたものを使用し、あるメールメッセージに対し上記アルゴリズムを適用した結果を図 4に示す。枠で囲まれているものは固有名詞、アンダーラインのものは普通名詞と認識されたものである。ほとんどの単語が正しく認識されていることがわかる。

To: postmaster%cabbage.math.keio.ac.jp@uunet.uu.net
Subject: NNTP
Date: Thu, 15 Mar 90 15:41:56 EST
From: Toshiyuki.Masui@A.NL.CS.CMU.EDU

カーネギーメロン 大学 出向 中 の 増井 と 申し ます。
貴 サイトの NNTP サーバを 使って fj の ニュース を 読 んでいたのですが、
この 2週間 ほど 接続 ができない 状態 が 続 いています。 慶 應 大学 では
現在 も NNTPサービス を 行なっておられるのでしょうか。 従来 と
変わっていないということであれば 当方 の 異常 と 思わ れますので
どうか 現状 をお 教え いただきたくお 願 い 申し あげます。
また、 他 にも NNTPサービス を 行なっているサイトを 御存知 でしたら
教 えていただけると 幸い です。

増井 俊之 カーネギーメロン 大学 機械 翻訳 センタ

図 4: 文字切り出しアルゴリズムの適用結果

切り出しアルゴリズムにおいて単語が正しく認識されなかった場合でも検索はほとんどの場合問題なく行なわれる。例えば “■□” という単語が辞書中に無い場合、この単語のシグナチャを計算するかわりに “■” 及び “□” のシグナチャが計算されてその論理和が使用されることになるが、パタン及びテキスト中の両者で同じシグナチャが計算されるため問題はおこらない。

辞書の不備及びアルゴリズムの単純さのため単語の切り出しがうまくいかない場合も存在する。例えば “大地震” は辞書に無いが “大地” は辞書に登録されている場合、 “大地震” という文字列を含むテキストに対して本アルゴリズムを適用すると “大地” 及び “震” に対してシグナチャが計算されてしまうため “地震” による検索が成功しないことになる。このような不都合は “大地震” のような単語も辞書にもれなく入れておくことによりある程度避けることができる。

各単語のシグナチャを計算する代わりに文字のシグナチャを使用することにすると単語の切り出しは必要なくなり上記のような問題は発生しないが、検索の効率は減少する。

3 曖昧検索

情報検索プログラムは、スペルミスや綴りの違い ({ 斉藤, 斎藤 }, {centre, center}, etc.) に対応するため、パタンマッチングにおいてある程度の誤りを許すようになっていることが望ましい。しかし UNIX の標準の**grep** などのコマンドはテキストが正しく入力正規表現にマッチしたときのみ成功と判定するため、誤りを許すようにするためには複雑な表現が必要になる。例えば “abac” という文字列の検索において 1 文字の欠落を許すようにするだけでも、**egrep** を使用すると **'(bac|aac|abc|aba)'** という複雑な指定が必要になる。これを避けるため曖昧検索を行なうコマンド**agrep** を開発した。

agrep は 1 文字欠落 (例: “aac”)、 1 文字化け (例: “abdc”)、余分な 1 文字追加 (例: “abdac”) を許すパタンマッチプログラムである。上記の 3 種類の誤りを許すパタンマッチング機械は図 5のような非決定性状態遷移機械で実現することができる。

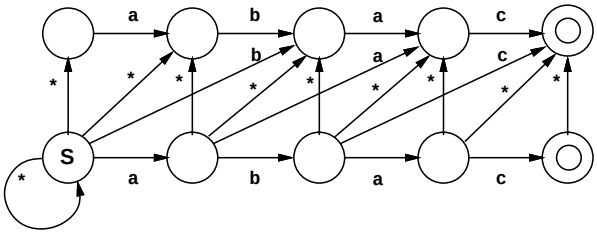

図 5: “abac” を認識する遷移グラフ

この機械を文字列**"abracadabra"**に対し適用すると以下のような状態遷移がおこる。灰色のノードがアクティブなノードを示す。

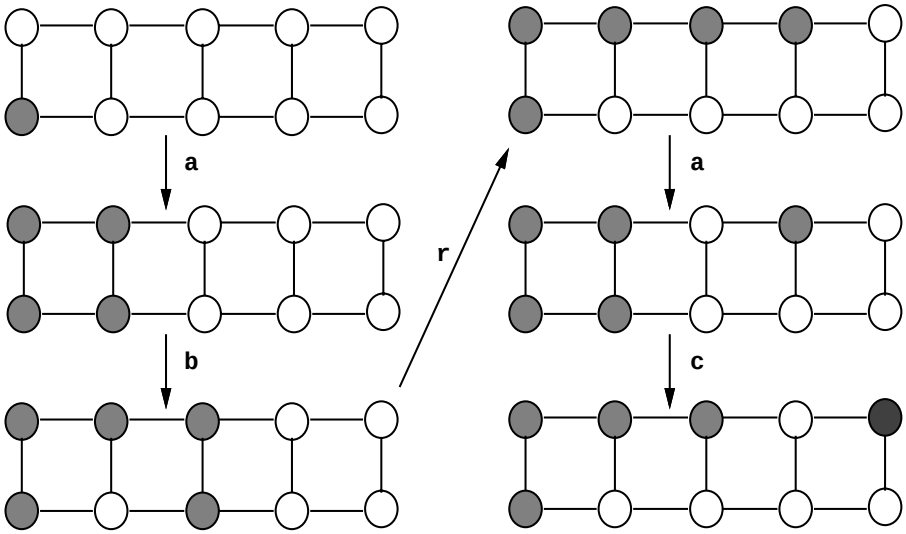

図 6: 入力 “abracadabra” による遷移

最後の状態のように最右のノードがアクティブになると文字列が認識されたことを示す。ここで、**"abrac"**は**"abac"**に余分の文字**"r"**が付加されたとものとみなされている。

このような状態遷移機械はハードウェアで簡単に構成することができるが [**?**]、ソフトウェアでも以下のように効率良い実現が可能である。各ノードのアクティブ状態を**1**, **0** で表現し、最左のノードを LSB、最も右のノードを MSB として上下の各ノード列を整数とみなす。例えばノード列●─○─●─○─○は**00101** (= 5) と表現する。上段のノード列を**u**, 下段のノード列を**l** という変数で表現し、時刻**t** の**u** の値を**u[t]** と表現することにすると、**u[t+1]**, **l[t+1]** は以下のように計算される。

```
l[t+1] = ((l[t] << 1) & mask[c] | 1);
u[t+1] = ((u[t] << 1 | l[t] << 2) & mask[c] | l[t] | (l[t] << 1));
```

ここで、**mask[c]** は文字**c** とパタンからあらかじめ計算しておく値で、パタン中の文字に対応する位置を**1** にしたものである。例えばパタンが**"abac"**のとき**mask['a']** は**01010** となる。

本アルゴリズムはビット列のシフト演算及び論理演算のみを使用しているため高速であり、高々テキスト長に比例した時間で実行できる。

本アルゴリズムは Baeza-Yates[2] による “shifter algorithm” を拡張したものとみなすこともできる。 Baeza-Yates の方法では誤りを許していないため、上述の**l[]** のみが使用されている。

agrep はシグナチャ法の後処理マッチングプログラムとして有効に使用することができる。例えばシグナチャ法において単語の読みからハッシュによりシグナチャが計算されるようになっていると、 “斎藤” と “斉藤” のような単語から同じシグナチャが計算されるため、テキスト文字列 “斎藤” はシグナチャ法及び**agrep** の両者においてパタン文字列 “斉藤” にマッチし、最終検索結果として得られることになる。偶然同じシグナチャを持った単語を含むテキストは、シグナチャ法による検索は通過するが、 “斉” も “藤” も含んでいなければ**agrep** を通過しないため最終結果としては残らない。このように本当に必要となる文書だけを検索することができる。

4 試作システム

上述のツールを使用し、**Nemacs** 上に文書検索システムを試作した。シグナチャの計算や曖昧検索自体は C で記述されたコマンドにより実行される。シグナチャは単語の読みからハッシュにより計算される。以下に本システムの使用例を示す。

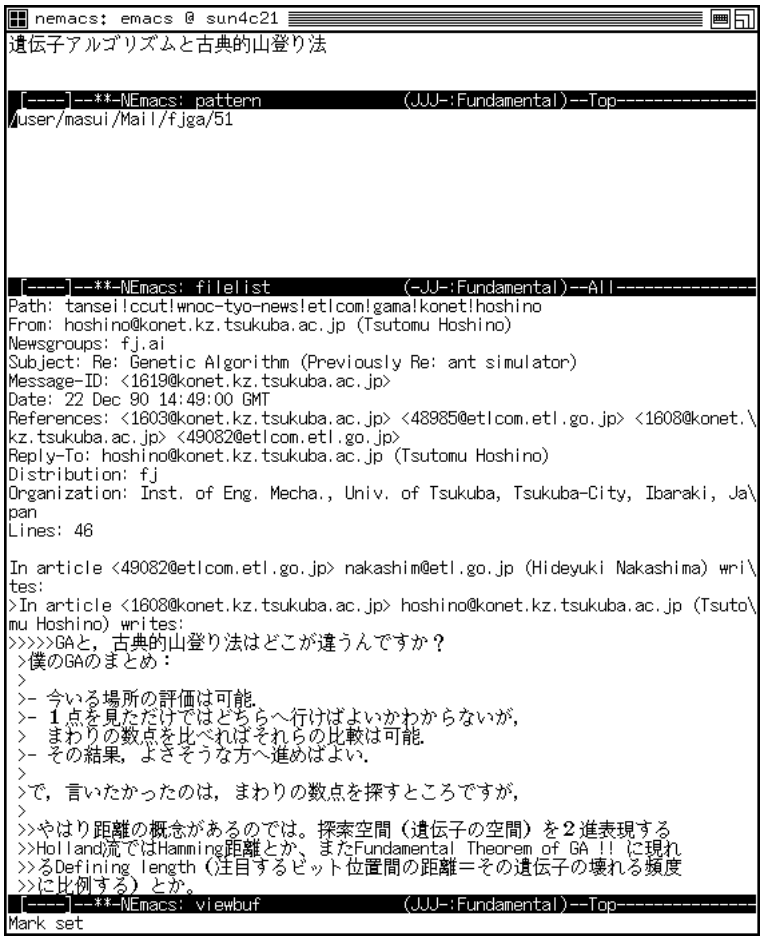

図 7: Nemacs 上に実現した文書検索システム

図 7の Nemacs 画面において、最上部のウィンドウは検索パタン文字列を示し、中間のウィンドウはマッチしたファイルのリストを示し、最下部のウィンドウはファイルの内容を示している。シグナチャファイルは検索の前にあらかじめ作成しておく。本システムでは 1 ファイルあたりファイル名 128 バイトとシグナチャ 128 バイト (1024 ビット) の合計 256 バイトを使用している。

5 評価

ニュース及びメール記事約 13,000 個 (38MByte) についてシグナチャファイルを作成して本システムによる検索の実験を行なった。実験に使用したシステムは NeXTstation(CPU は 68040) OS2.0 である。

- シグナチャファイルサイズ

  シグナチャファイルの大きさは 3.3MByte ($= 13,000 \times 256$) になった。これは文書本体のサイズの 1/10 以下であり、文書サイズに対するシグナチャのオーバーヘッドは全単語インデクス方式に比べかなり小さいということができる。

- 検索時間

  13,000 個全てのファイルに対し**egrep** で検索を行なった場合[2]、平均すると user time, system time とも 5 秒強となり、実際にかかる時間は 30 秒程度であった。これに対しシグナチャファイルを使用する場合はシグナチャ

[2]egrep 起動のオーバーヘッドを最小にするため、 1 回あたり 1,000 個引数を与えて**egrep** を 13 回起動した。

による一次検索は平均すると user time が 0.7 秒、 system time が 1.8 秒であり実際にかかる時間は数秒であった。このように、シグナチャの使用により検索を数倍から 10 倍程度高速化することができた。この差は検索ファイルの量が多くなるとさらに重大になると考えられる。 Nemacs を通して検索する場合は**elisp** を使うため実行が遅くなるが、平均約 30 秒で検索を行なうことができた。

- 検索精度

  (評価中)

6 今後の課題

通常の文書検索システムにおいては、テキスト中に存在しない単語を指定して検索を行なうことはできない。現システムでも単語の読みからシグナチャを計算しているためこの点は同様であるが、シグナチャの作成方法を工夫して似た単語から同じシグナチャが得られるようにすれば文字列がマッチしなくても関連したテキストを抽出することができるようになる。文書検索システムにおいてはパタンに指定した単語とテキスト中で使われている単語の違いのために検索率が予想外に悪いという実験結果 [7] を考えると、字面よりも意味を重視する検索手法が重要と思われるが[3]、シグナチャ法はそのような検索にも有効である。例えばシソーラスを使用して単語のクラスタリングを行ない、クラスタ番号をシグナチャとして使用することにより、ある程度意味を考慮した検索を行なうことができると予想される。

現システムは Nemacs 上で**elisp** を多用しているためシグナチャ法の本来の速度が得られていない。プログラム全体を日本語 NeXTstep 等に書きかえることによりさらに実用的なシステムを構築することができると思われる。

7 結論

シグナチャ法及び曖昧検索コマンドを使用して、 UNIX 上の実用的なテキスト検索システムが構築できた。本システムを使用すると**grep** のような標準の検索コマンドを使用する場合に比べ数倍の速度で検索を行なうことができた。またこの際必要となる余分のインデクスファイルは原文書サイズの 1/10 以下であった。シグナチャの作成方法を工夫して文書の意味まで考慮した検索を行なうシステムを作成予定である。

参考文献

[1] Jun'ichi Aoe. An efficient digital search algorithm by using a double-array structure. *IEEE Transactions on Software Engineering*, Vol. 15, No. 9, pp. 1066–1077, September 1989.

[2] Ricardo A. Baeza-Yates and Gaston H. Gonnet. A new approach to text searching. In *Proceedings of the Twelfth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 168–175, June 1989.

[3] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression*. Prentice Hall, Englewood Cliffs, NJ, 1990.

[4] C. Faloutsos and S. Christodoulakis. Description and performance analysis of signature file methods for office filing. *ACM Transactions on Office Information Systems*, Vol. 5, No. 3, pp. 237–257, July 1987.

[5] Toshiyuki Masui. Keyword dictionary compression using efficient trie implementation. In *Proceedings of Data Compression Conference*, Salt Lake City, Utah, April 1991.

[6] 日経エレクトロニクスの昔の記事 (未調査)

[7] CACM の 1985 ごろの記事 (未調査)

[3]例えば 4 つのキーワードを使用して検索を行なう場合、抽出したいテキスト中でそれぞれのキーワードが使用されている確率が 0.8 であるとするとそのテキストが見つかる確率は $0.8^4 = 0.4$ となってしまう。 [7] の判例データベースによる実験によると、実際には 1 割程度しか検索できていない場合でも被験者 (弁護士) は 7 割程度は検索できているものと信じていたということである。