複数文字列の探索に関する研究

清谷則好

複数文字列の探索に関する研究

平成８年９月

浦谷　則好

目次

第1章　序論

1．1　本研究の背景と目的

情報化社会の進展に伴って，個人個人に供給される情報はどんどん増大し，２４時間を費やしても，もはやその全てにアクセスすることができなくなっていると言われている．必要な情報を，必要な時に利用できるようにデータベースが様々な機関で作成され，供給されている．画像や音声情報のデータベース化も進んできているが，未だに文字情報が主流である．データの検索の手掛かりとしては，キーワードが最も利用されている．

キーワード検索手法としては，キーワードに対する索引表をあらかじめ構築しておくインデックス検索手法[1]が一般的であるが，この手法には次のような問題点がある．

（1）キーワードの抽出に多大な労力と時間を必要とする．

（2）検索対象はキーワードにより制約され，任意の語彙の検索は不可能である．

そこで，あらかじめデータに何もキーワードを付けておかないで，文書中にある語を全てキーワードとして使えるようにする全文検索が考えられている．この場合，ユーザは何の制約も受けずに自由にキーワードを指定できることになる．データベースシステムはあらかじめユーザの要求を知ることができないので，検索要求が出る度にユーザが指定したキーワードの出現有無をデータに直接当たって調べなければならない．したがって，データベースが大きくなるに連れて，検索時間もどんどん大きくなっていく．そこで，できるだけ高速な文字列検索手法が求められている．

また，ワードプロセッサやプログラミング言語のコンパイラ（やインタープリタ）においても文字列の検索は基本的な機能として必須のものである．文字列の検索は，指定した文字列（以下パターンと呼ぶ）が文書（以下テキストと呼ぶ）中のどこに現われているかを調べるという単純な作業であるが，前述のように電子化された文書の処理にとっては全体の効率に大きく関与する操作なので，これを効率良く実現するためのアルゴリズムは各所で研究されている[2]．

１個のパターンを探す手法としてはBoyer-Mooreの方法（ＢＭ法）[3]が最も効率の良い方法として知られている．しかし，この方法で複数パターンの検索を行うためにはパターンの数だけの繰り返し使用が必要となる．複数のパターンを同時に効率良

く検索する方法としては，Aho-Corasickの方法（ＡＣ法）[4]が有名である．このprobe rate（テキスト１文字当たりの照合回数；つまり効率の逆数）は常に１になる．

本論文ではＢＭ法にＡＣ法の状態遷移という考え方を取り入れて，ＢＭ法を複数パターンに拡張する方法を研究する．ＢＭ法を複数パターンに拡張しようとする試みとしてはKowalski とMeltzerによる方法[5]やCommentz-Walter による方法[6]があるがここで紹介する方法はそれよりprobe rateが優っている．

この方法の効率を推定し，実験によりそれが正しいことを示す．また，アルゴリズムの正当性を証明する．さらに，この方法を２次元パターンの検索に拡張する方法についても明らかにする．

１．２　文字列検索の手法

１．２．１　インデックス検索手法

キーワード検索手法としては，キーワードに対する索引表をあらかじめ構築しておくインデックス検索手法が一般的である．この手法の概念図を図１．１に示す．つまり，あらかじめキーワードを人手あるいは自動的に抽出しておいて，それが出現する位置をインデックスファイルに記憶しておく方法である．キーワードの検索時には，索引表からキーワードを検索し，キーワードの位置情報を得ればよい．したがって，非常に高速な検索が実現できるが，

（１）索引表をとっておくためにかなり大きな記憶容量を必要とする．

（２）新たなキーワードを追加するには索引表を作り直す必要があり，この時多大な時間を必要とする．

（３）キーワードの自動抽出法も研究されている[7,8]が，キーワードの抽出に多大な労力と時間を必要とする．

という問題点があり，何よりも

（４）検索対象はキーワードにより制約され，任意の語彙の検索は不可能である．

といった致命的な欠点を有している．

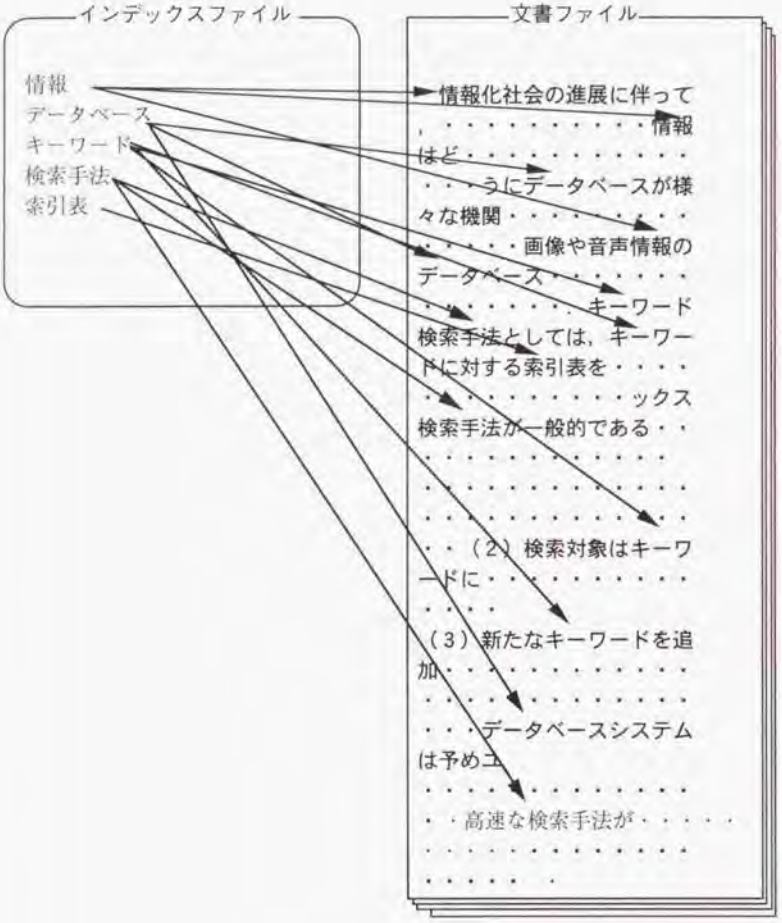


図１．１　インデックス検索の概念図

１．２．２　長尾・森の方法

インデックス検索手法には「検索対象はキーワードにより制約され，任意の語彙の検索は不可能」という致命的な欠点があった．それなら，「テキストに現われる全ての文字列に対してインデックスを作成しておけばよい」という考えもありうる．しかし、例えばｎ文字列の全てのインデックスを作成するには，文字列を記憶するのに約２ｎｋバイト（ｋはテキストの長さ；日本語は２バイトで表されるから２が掛かる），文字の出現位置を記憶するのに４ｋバイト（ポインタを記憶するのに４バイト必要と仮定して），合わせて２ｋ（ｎ＋２）バイトの記憶容量が必要である．したがって，長さｎまでの文字列全てのインデックスを作成するには（n^2+5n）ｋの記憶容量が必要だということになって，ｎが３でも元のテキストの１２倍，ｎが１０なら元のテキストの７５倍もの記憶容量が必要となる．したがって，元のテキストが大きければインデックス検索手法によって任意の語彙を検索するということは事実上不可能となる．

長尾と森はテキストからｎグラム統計を取る際に，記憶容量をそれほど必要としない方法を考案している[9]．この手法は文字列検索のために考案されたものではないが文字列検索にも利用することができる．

この手法では，テキストのｉ文字目から最後の文字（ｋ文字目）まで（実際には最大２５５文字分としている）を１つの文字列と見なし，ｉ＝１からｉ＝ｋまでのｋ個の文字列の存在を仮定する（実際には図１、２の原ストリングのように１本の文字列である）．このｋ個の文字列を辞書順にソートする．ソートされた結果は文字列の配列順序によって示される．その簡単な例を図１，２に示す．各文字の位置を示すために一般に４バイトを必要とするので，このソートされたデータの記憶容量は４ｋバイトである．

ポインター	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
原ストリング	ある日ある場所で会合がもたれた。
ソート結果	16 1 4 11 8 15 13 12 2 5 14 3 9 10 7 6

図１．２　文字列のソート例

このようにソートされた文字列に対して，隣接している２つの文字列が先頭から何文字目まで一致しているかを調べ，これを前の文字列の情報として図１．３のように付加する．一致文字数を最大２５５までとすればこの情報を保持しておくのに１文字列当たり１バイトが必要なので全体ではｋバイトが必要である．以上の処理に必要な記憶容量は，（１）長さｋの原テキストの記憶に２ｋバイト，（２）ソートされた結果の記憶に４ｋバイト，（３）一致文字数の保持のためにｋバイト，となり全体で７ｋバイト必要であることがわかる．言い換えれば，付加すべき記憶量は元のテキストの２．５倍で良いということになる．これは，インデックス検索手法よりずっと少ない量である．

今，ある文字列をテキストから検索する場合を考える．この文字列がテキストに存在するか否かはソート結果があるからバイナリーサーチでポインターの示す文字列と比較することで簡単に調べることができる．これに要する時間は$O(\log_2 k)$で済むことは良く知られている．また，出現位置が１つでない場合も，バイナリーサーチで最初に見つかった位置から前後に一致文字数を調べていくだけで文字列そのものを比較することなく，どれだけの検索出力があるのかを知ることができる．この検索時間はもちろん直接検索結果の得られるインデックス検索手法と比べれば大きいが，$O(\log_2 k)$であることを考えればかなり小さいと言える．

部分文字列	一致文字数
ある日ある場所で・・	5
ある日ある人が・・・	2
ある場所で会合が・・	6
ある場所で会う。	2
あるのは・・・・・・	1
あればあったで・・・	0
いい加減な・・・・・	

図１．３　一致文字数の情報

1．2．3　全文検索手法

インデックス検索手法では任意の語をキーワードとして検索することができないことが分かった．長尾・森の方法では元のテキストに対して２．５倍もの余分な記憶容量が必要な上に，文字列のソートのためにかなりの前処理時間が必要である．

これらの欠点を克服するためには，あらかじめデータに何もキーワードを付けておかないで，文書中にある語を全てキーワードとして使えるようにする全文検索以外に手段がない．ユーザは何の制約も受けずに自由にキーワードを指定できることになるが，データベースシステムは予めユーザの要求を知ることができないので，検索要求が出る度にユーザが指定したキーワードの出現有無をデータに直接当たって調べなければならない．

全文検索法は，全文書を対象にそのまま語を検索する方法（全走査法）と文書をブロックに分けそのブロックの特徴ベクトル（重ね合わせ符号）を事前に照合することで，一部の文書ブロックのみを検索対象とする方法（部分走査法）[10,11]に分けることができる．

部分走査法の概念図を図１．２に示す．この手法では文書をブロックに分け，それに含まれる語彙を特徴ベクトル（文書ベクトル）で表しておく．検索したいキー

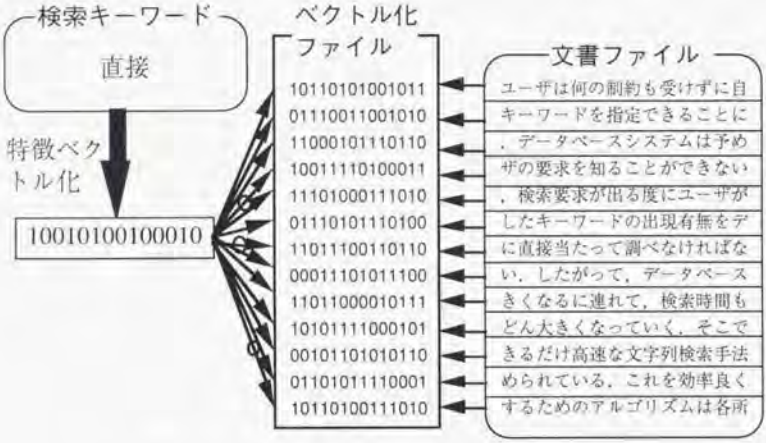


図１．４　部分走査法による検索の概念図

ワードも特徴ベクトルで表し，文書ベクトルとの論理積をとり，それがキーワードの特徴ベクトルと一致すれば，その文書ブロックを対象にキーワードの検索を行なう方法である．この方法の利点はうまく特徴ベクトルを作れば，検索対象とするブロックを絞ることができて高速な検索ができることである．しかし，ベクトル照合に成功しても，文書ブロックにはキーワードが存在しない場合もあり（これをFalse Dropという），無駄な検索を行なう場合もある．False Dropを少なくするには特徴ベクトルを大きくすればよいが，そうすると文書ベクトルをとっておく記憶容量が大きくなるだけでなく，ベクトル照合に要する時間も長くなるという問題が生じる．そんな訳で，期待するほど全走査法に比べて検索時間を短縮できていない．そこで，以降は全走査法に絞って議論を進める．また，全走査法で使われる手法は部分走査法で文書ブロックを絞った後，キーワードの検索を行なう場合に使われるものと，本質的に何ら変わりはないので，より一般的でもある．

全走査法ではハードウェアによる高速化も試みられている[12),13)]が，柔軟性を考えればやはりソフトウェアによる高速化は現在でも重要な研究課題である．全走査法のアルゴリズムについては第２章で詳述する．

１．３　本論文の構成

まず第１章で，研究の背景と目的，特にキーワード検索手法について概観し，合わせて，本論文の構成についても述べる．

第２章では全文検索手法のアルゴリズムについて検討する．従来のアルゴリズムについて説明した後，ＡＣ法の状態遷移という考え方を取り入れて，ＢＭ法を複数パターンの検索に拡張する方法（ＦＡＳＴ法）を提案する．

第３章ではＦＡＳＴ法の効率について検討し，アルゴリズムの正当性を示す．さらに，実験結果を示し，効率の推定式がよく一致すること，多パターン時にはパターン数の増加に伴って効率が単調に減少しないという現象について説明する．メモリ節減手段についてもふれる．

第４章では，２次元パターン探索の従来のアルゴリズムについて説明した後，ＦＡＳＴ法を２次元パターン探索に拡張する方法について述べる．実験により，従来の手法より効率が良いことを示し，さらに一般のｎ次元パターンの検索に拡張する方法についても述べる．

最後に第５章で結論を述べる．

第２章　複数文字列の探索手法

２．１　まえがき

近年，文書の作成や編集は，ワードプロセッサを用いて行なうことが多い．その際，しばしば使用する機能に，文字列の検索や置換がある．文字列の検索は，与えられた文字列が文書のどこに現われているかを調べ，見つかったらその位置にカーソルを移動し，見つからない場合にはそれをユーザに通知するものである．文字列の置換は，文書中のある文字列を別の文字列で置き換えるものであるが，当然，このためには文字列の検索を必要とする．コンパイラやインタープリタにおいても予約語の処理に文字列の探索が必要である．このように，文書を処理するためには文字列の検索は最も基本的な操作ということができる．このため，これを効率良く実現するためのアルゴリズムは昔から各所で研究されている．

文書データベースをあらかじめ決められたキーワードで検索する場合には第１章で説明したインデックス検索手法を使うのが一般的であるが，全く自由なキーワードをユーザに許すためには，この章で説明する全文検索手法を使わざるを得ない．この場合，第１章で述べた部分走査法による高速化も考慮することができるが，全走査法の方が根源的であることは前述した通りである．そのうえ，文書データベース検索以外のワープロやコンパイラなどの文書処理においても，全走査法の全文検索手法がオーバーヘッドの計算時間や記憶容量の面から考えて，最適であるのは明らかである．

２．２　全文検索手法

提案する文字列検索の新しいアルゴリズムの理解を容易にするために，まず従来の全文検索手法（このうちの全走査法）を概説する．最初に素朴な方法について説明した後，KnuthとMorrisとPrattが考案した照合回数がテキストの長さに比例するアルゴリズム（ＫＭＰ法）について説明する．BoyerとMooreは「パターンの右端から逆向きにテキストとの照合する」ことで，ＫＭＰ法より効率の良い方法を考案した．AhoとCorasickは有限オートマトンの考え方を取り入れることによってＫＭＰ法を複

数のパターンの同時照合に拡張する方法を考案した．２．２．３と２．２．４でこれらの方法について説明する．

２．２．１　素朴な方法

文字列の照合は，あるパターンがテキスト中に含まれるか否かを判定し，含まれているならば，その位置を見つけることである．パターンが１つの場合には，この問題を解く次のような素朴なアルゴリズムを簡単に思いつくことができる．

まず，パターンの左端の文字をテキストの左端の文字に合わせるようにパターンをテキストの上に重ねておいて，パターンとその真下にあるテキストの部分文字列が一致しているかどうかの照合を行う．つまり，パターンとテキストの対応する文字が一致しているかどうかを左端から右方向へ１文字ずつ調べていく方法である．もし，途中で照合に失敗（文字が不一致）すると，パターンをテキスト上で１文字分右へずらし，再び照合を開始する．もちろん，パターンの全ての文字について照合に成功すれば，そこでテキスト中にパターンが見つかったことになる．

全てのパターン出現位置を探す場合，このアルゴリズムによる全照合回数は大体テキストの文字数に比例すると考えられるが，最悪の場合（例えば，全てaからなるテキスト"aaa.....a"を対象にパターン"aaaab"を探すような場合）には，パターンの長さ（文字数）をm，テキストの長さをｎとすれば，m回の照合で照合位置を１文字分ずらせるだけであるから，ｎがmより十分大きければ全照合回数は約ｍｎ回となってしまう．

２．２．２　Knuth-Morris-Prattの方法[1]

最悪の場合でも照合回数がテキストの長さに比例するようなアルゴリズム（ＫＭＰ法）をKnuthとMorrisとPrattが考案している．これはどこまで照合に成功したかの情報を保持しておいて失敗したときにパターンの検出漏れの心配のない最大の長さ分パターンを右にずらす方法である．

図２．１にこの方法による照合の様子を示す．この図で３度目の照合（↑の所）に失敗した後，３文字分パターンをずらしているのは，以下の理由による．

パターンのｉ番目の文字をπ[i]とし，テキストのｊ番目の文字をτ[j]とすれば，

$\pi[1]=\pi[3]=\pi[4]=a \neq \pi[2]$,

$\pi[2]=\pi[5]=b$

で，

$\pi[1]=\tau[1]$，

$\pi[2]=\tau[2]$（$\neq\pi[1]$），

$\pi[3]\neq\tau[3]$（$\because\tau[3]\neq\pi[1]$）

であるから，3文字以下のパターン移動では必ず照合に失敗してしまうからである．つまり，照合失敗時のパターンの移動量およびパターン中で次に照合すべき文字位置はパターンそのものから（テキストとは無関係に）あらかじめ決めておくことができる．紙面の都合でこの量を計算するアルゴリズムは割愛するが，同様な考え方は，後述するBoyer-Mooreのアルゴリズムでも使われている．

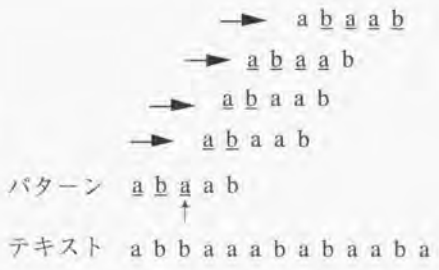


図2．1　KMP法による照合手順

KMP法それ自身はあまり使われていないようであるが，テキストの長さに比例する時間で照合が可能であることを証明したこと，後に出現するAho-Corasick法やBoyer-Moore法に示唆を与えたことで高く評価されている．

2．2．3　Boyer-Mooreの方法[1),2)]

BoyerとMooreは，「パターンの右端から逆向きにテキストとの照合を行ったほうが左端から行うより多くの情報が得られる」ということを利用した方法を考案した．例えば図2．2のような場合，パターン（“state”）の右端から照合を始めると，どちらも‘e’なので照合は成功する．この場合次に1つ手前の文字の照合を行う．今度は照合に失敗する．このときパターンを右に4以下分動かしても必ず失敗する

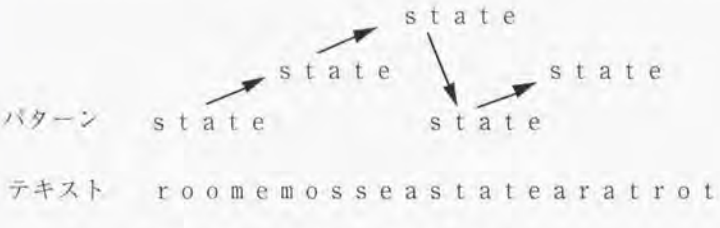


図2．2　BM法による照合手順

ことは，

・‘m’がパターン中に存在しない

・‘e’はパターンの先頭文字ではない

ことから明らかであるので，5だけパターンを右に動かしてから照合を再開してもよい．つまりパターンの右端から照合していくことによって無駄な比較を減らすことが可能である．このときパターンを動かす量はパターン自身と照合失敗時のテキスト上の文字とから決まるので，照合を始める前にあらかじめ計算しておける．これがBM法の基本的な発想である．

照合失敗時のテキスト（τ）の文字cから得られるパターン（π）をずらす文字数d[c]は，次のように定式化できる．ただし，パターンの長さをmとする．

d[c]＝min{i|(i=m)または(0≦i<m)かつπ[m-i]=c)}

パターン自身から得られる移動量dd[p]は，

dd[j]＝min{i+m-j|(i≧1)かつ(i≧jまたはπ[j-i]≠π[j])
かつ(任意のk(j<k≦m)に対して(i≧kまたはπ[k-i]=π[k]))}

となる．このdとddを用いればBM法の照合アルゴリズムは図2．3のようになる．（図2．3以下，アルゴリズムは全てPascal風に記述する．）dを求めるアルゴリズムは簡単なので省略するが（定義通りコーディングすればよい），ddを求めるアルゴリズムは多少複雑であり，図2．4のようになる．図2．4でfは，

f[j]＝min{i|(i=m)または[(j<i<m)かつ
(π[i+1]…π[m]=π[j+1]…π[m+j-i])]}

で定義される関数であり，ddの計算のための補助として用いられている．

```
begin j:=m;
  repeat i:=m;
    while (i>0) and (π[i]=τ[j]) do
      begin i:=i-1;  j:=j-1 end;
    if i≠0 then j:=j+max{d[τ[j]],dd[i]}
  until (i=0) or (j>n)
  {i=0ならばτ[j+1]から始まる部分文字列がπ
 と一致している．i<0ならばτ中にπはない．}
end
```

図2．3　BM法の照合アルゴリズム

ＢＭ法のアルゴリズムの効率については，最良の場合probe rateが１／ｍとなることは明らかである．テキスト中にパターンが全く現れない場合には，最悪の場合でも４以下になることが証明されている[3]．（実際にはもっと小さいであろうと予想されている．）この方法のprobe rateはランダムテキストを対象にした場合，１以下になることが証明されている[1]．

```
begin
   for k:=1 to m do dd[k]:=2*m-k;
   j:=m;  i:=m+1;
   while j>0 do
     begin
       f[j]:=i;
       while (i<m) and (π[j]≠π[i])
         begin
           dd[i]:=min{dd[i],m-j};
           i:=f[i]
         end;
       i:=i-1;  j:=j-1
     end;
   t:=i;  (f[0]に相当)
   for i:=1 to m do
     begin
       dd[i]:=min{dd[i],t+m-i};
       if (i<t) then t:=f[t]
     end
end
```

図２．４　関数ddを求めるアルゴリズム

２．２．４　Aho-Corasickの方法[4]

２．２．１から２．２．３までは全て照合したいパターンが１個の場合の方法であった．これらの方法で複数のパターンを照合しようとするとパターン数だけの繰り返しが必要となってしまう．AhoとCorasickは有限オートマトンの考え方を取り入れることによって複数のパターンを同時に照合する方法を考案している．

ＡＣ法はパターンの集合からパターン照合機械（以下ｐｍｍと略記）と呼ばれる一種の有限オートマトンを作り，それをテキスト上で左から右へ１回走査させて各パターンの出現位置をことごとく検出するアルゴリズムである．ｐｍｍはgoto，failure，出力という３つの関数から構成される．例えば，パターンが“state”，“east”，“smart”の場合のｐｍｍは図２．５のようになる．図で実線はgoto関数を，破線はfailure関数を，下線付きの文字列はその状態における出力を示している．破線で示した遷移は通常のオートマトンのε動作，すなわちヘッドを固定したまま状態だけを変えることを表している．ただし図２．５では状態５，８，９以外から状態０への破線は省略してある．

ｐｍｍ作成のアルゴリズムの記述は省略するが，後述するＦＡＳＴ法でも同様なアルゴリズムが用いられる．このｐｍｍをテキスト上で走査させれば，その状態遷移によってパターンの存否がわかることになる．テキストが“osseastatear”の場合を例に状態遷移の様子を示すと図２．６のようになる．この例からも分かるようにＡＣ法ではパターンの数に関わりなくテキストの長さに比例する時間で照合が

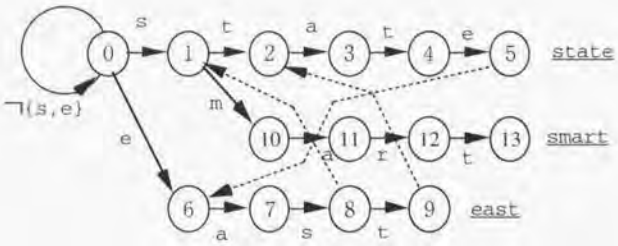


図２．５　ＡＣ法におけるパターン照合機械

可能である．failure関数をなくし，goto関数だけのｐｍｍを作ることも可能で，この場合にはprobe rate を完全に１にすることができる．

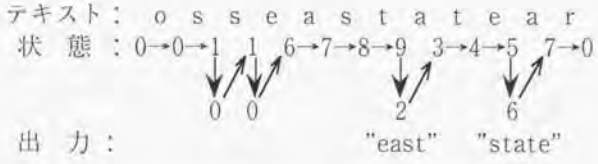


図２．６　ｐｍｍの状態遷移

２．３　ＦＡＳＴ法のアルゴリズム

前述したように，パターンが１つの場合の照合はBoyer-Moore法が最も効率の良い方法として知られている．ところが，複数のパターンの照合にはパターンの数だけの繰り返し使用が必要となって，急激に効率が低下してしまう．一方，Aho-Corasick法は複数パターンを同時に扱うことが可能である．ＡＣ法は，ＫＭＰ法に有限オートマトンの考えを取り入れて複数パターンを同時に照合できるようにしたものと見なすことができる．ＢＭ法に有限オートマトンの考えを取り入れれば，複数パターンの場合でも平均的に亜線形（sublinear, prove rate が１以下）の照合が可能となるはずである．筆者はＢＭ法とＡＣ法の両者の考え方を取入れることによって，複数パターンの場合でも効率的にパターンを照合できるアルゴリズムを考案した．この方法をＦＡＳＴ法（a Flying Algorithm for Searching Terms）と呼ぶことにする．以下でＦＡＳＴ法の考え方とアルゴリズムについて述べる．

２．３．１　ＦＡＳＴ法の考え方

ＢＭ法の「右端からパターンを照合する」というアイデアとＡＣ法の「ｐｍｍによって複数パターンの同時照合を可能にする」というアイデアを両方とも組み込むことができれば，複数パターンにおいてもＢＭ法と同様な低いprobe rateを実現できるはずである．

パターンの集合は図２．５の場合と同じで，テキストは図２．２と同じであると仮定する．このとき複数のパターンの照合を同時に行うにはパターンの右端を揃え

て図２．７のようにテキストの上に並べてから照合を開始すればよい．図２．７の場合，テキスト上の文字‘m’は３つのパターンの右端のどれとも一致しないからパターン群を後ろに動かすことができる．‘m’はパターン“smart”の中にしか現れないから３文字分動かしても探索もれが起きる心配はない．したがってパターン群を３文字分右へ動かしてから照合を再開すればよいことになる．問題は，探索もれが生じない範囲でパターン群を動かせる最大の量をどのようにすれば求められるか，ということにある．

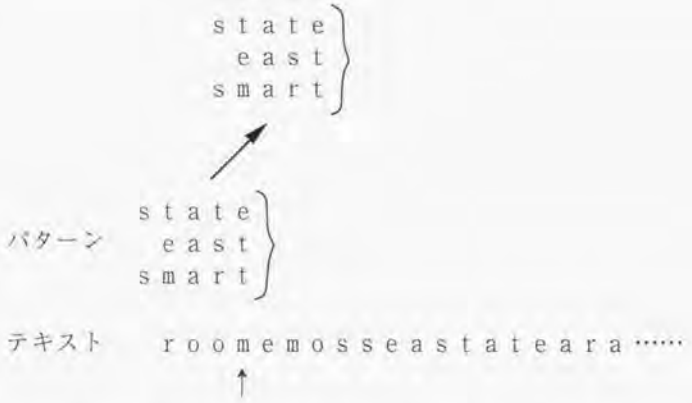


図２．７　複数パターンの右端からの照合

これには図２．８に示す２つの場合を考慮すればよい．つまりテキスト上の文字a_jから始まった照合があるパターンαと途中まで成功して，a_iで失敗した場合，

・テキストの部分列$a_i a_{i+1} \cdots a_j$があるパターンβの部分列になっている場合（ａ）と

・テキストの部分列$a_k a_{k+1} \cdots a_j$（$i < k \leq j$）があるパターンβの先頭の部分列になっている場合（ｂ）

である．この２つの場合，パターンβの一致する部分列をテキストの当該部分列に重ねるような位置にパターン群を動かして（図２．８のシフト量$S_{\alpha\beta}$）照合操作を再開してもパターンβの探索もれが起こる可能性はない．すなわち，$S_{\alpha\beta}$を厳密に

定義すれば，（ただし，パターンαの長さをmとし，パターンβの長さをm'とする．）

$S_{\alpha\beta}$= min {d|[(d≧1)かつ((m'-d)≦(j-i)または$\beta_{m'-d-j+i}=a_i$)

かつ(任意のn(m-j+i＜n≦m) に対して

((m'-d)≦(m-n) または $\beta_{m'-d-m+n}=\alpha_n$))] または [d=m']}

となる．

もちろん，場合（a），（b）に相当する部分列がない場合にはパターンβの長さだけ動かすことが可能である．このシフト量$S_{\alpha\beta}$はパターンの集合が与えられた時点で決まってしまう．

BM法でシフト量を求める方法も基本的にはこの考え方と同じである．ただBM法ではパターンαとパターンβは同一であるのに対し，複数パターンを考慮するためには，照合中のパターンαに対し，すべてのパターン（同一パターンを含む）に

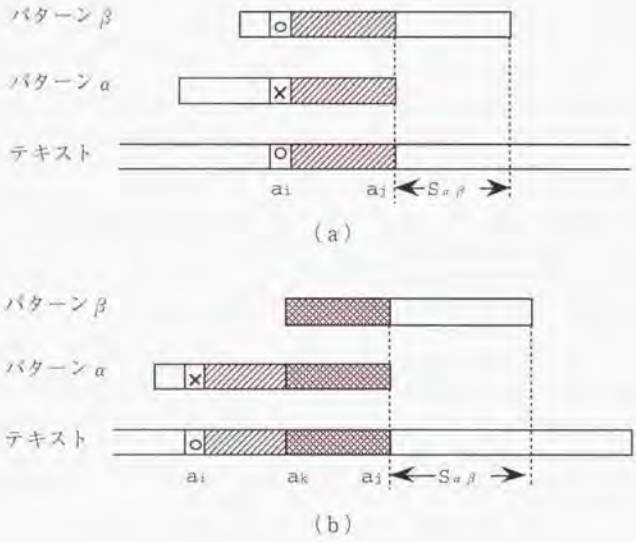


図2．8　パターン群シフト量の計算

対する$S_{\alpha\beta}$を求め，その最小値S_{α}つまり，

S_{α} =min |$S_{\alpha\beta}$：$\beta \in$ パターン集合|

をパターン群のシフト量としなければならない．また，ＢＭ法では照合に失敗した時，テキスト上の文字（a_j）に対するシフト量をdという関数で，それまでの照合情報（$a_{i+1}a_{i+2}\cdots a_j$）に基づくシフト量をddという関数で別々に取り扱っていたが，ＦＡＳＴ法では$a_ia_{i+1}\cdots a_j$の出現をまとめて取り扱っている点でも異なる．このため，パターンが１つの場合でもＦＡＳＴ法の方がＢＭ法よりシフト量が多くなる場合が生じ，一般に効率が良い．

一方，照合に成功している間は，どのパターンのどの部分を照合しているかの情報はＡＣ法と同様にｐｍｍの状態として保持しておけばよい．ただＡＣ法と異なる点は状態遷移を引き起こす文字入力がテキスト上で右から左への方向で取られるだけである．この状態遷移を求めるアルゴリズムは２．３．２で述べるが，状態遷移のためのｐｍｍは前述のシフト量の計算にも使われる．

ここで述べるアルゴリズムは，簡単に言えば，照合に成功している間はＡＣ法と同様の状態遷移を行い，照合失敗時の動作には複数パターンに拡張したＢＭ法のシフト関数を利用するというものである．

２．３．２　ＦＡＳＴ法のアルゴリズム

図２．９にgoto関数と出力関数を求めるためのアルゴリズムを示す．このアルゴリズムはｐｍｍをパターンの後ろから作る点を除けばＡＣ法と同じである．ただし，図２．１０のアルゴリズムで用いるために，節点０からの深さを示す関数depth と，パターンの左端に対応する状態を示す関数emapと，パターンの最小長minlenも計算している．

パターンの集合を $\{y_1, y_2, \cdots, y_n\}$ とする．

状態ｓが作られるとき，$out(s)$=**empty**と仮定する．

$g(s,a)$ の初期値は **fail** と仮定する．

出力はgoto関数gと出力関数outである．

図２．１０のアルゴリズムで利用するminlen,depth,emapもここで計算される．

```
begin
  nst:=1;  depth(0):=0;
  minlen:=large;  /* large はパターン長に比べて充分大きい正整数 */
  for  i:=1  to  k  do  enter( y_i, i)
end

procedure  enter(a_1a_2…a_m, k)
begin
  state:=0;  j:=m;
  while ( g(state, a_j ) ≠ fail) and ( j > 0 )  do
    begin
      state:=g (state, a_j);
      j:=j-1
    end
  while  j > 0  do
    begin
      g (state, a_j):=nst;
      depth(nst):=m-j+1;
      state:=nst;
      nst:=nst+1;
      j:=j-1
    end
  emap(k):=state;
  out(state):={a_1a_2…a_m};
  minlen:=min{ minlen, m}
end
```

図2．9　goto関数を求めるアルゴリズム

図２．１０には図２．９のアルゴリズムの結果を利用してfailure関数を求めるアルゴリズムを示す．図２．１０でϕはＡＣ法のfailure関数に相当するものであり，状態stの場合，$depth(\phi(st))$はＢＭ法のアルゴリズムでシフト量（dd）を求める際に用いた補助関数fに相当している．ϕを利用してＦＡＳＴ法のfailure関数fを計算する．図２．１０のステートメントＡでfを更新しているのは図２．８の（ａ）の場合に相当している．つまり，パターンを照合していて，状態fst-入力cで照合に失敗した場合，次のポインタをパターンβの最後尾に移すための計算である．関数ddmは図２．８の（ｂ）に相当するシフト量を表わしている．ddmの計算の最初のループで，照合済みのパターンαのsuffix（後綴り）をprefix（前綴り）に持つパターンβを探して，その最後尾までのポインタのシフト量（の最小値）を求める．後のループでは（未決定のものを含めて）ddmの値をその１つ手前の状態（節点）から順に決めていくことによって，ddmに矛盾（後ろの状態のddmが直前の状態のddmより２つ以上大きくなること）がないようにしている．アルゴリズムの最後で$g(j,c)$=**fail**となる$f(j,c)$に対して，$\min(f(j,c), ddm(j))$をセットして，failure関数fが完成する．

---

goto関数gとminlen,depth,emapは図２．９のアルゴリズムによって求められているとする．
$f(s,a)$と$ddm(s)$の初期値は**large**（十分大きな数）とする．
出力はfailure関数fと出力関数outである．

```
begin
  queue:=empty;   φ(0):=-1;
  for each c such that g(0,c)=s≠fail do
    begin
      queue:=queue.s;
      φ(s):=0
    end;
```

```
while queue≠empty do
  begin
    let queue:=r.tail;
    queue:=tail;
    for each c such that g(r,c)=s≠fail do
      begin
        queue:=queue.s;
        fst:= φ(r);
        while (fst>=0) and (g(fst,c)=fail) do
          begin
            f(fst,c):=min{ f(fst,c), depth(r)}; <--- A
            fst:= φ(fst);
          end;
        if fst>=0 then φ(s):=g(fst,c);
                  else φ(s):= 0
        out(s):=out(s).out(φ(s))
      end
  end;
for j:=1 to k do
  begin
    fst:=φ(emap(j));
    while fst > 0 do
      begin
        ddm(fst):=min{ ddm(fst), depth(emap(j))};
        fst:=φ(fst)
      end
  end;
ddm(0):=minlen;
queue:={0};
```

```
  while  queue≠empty  do
    begin
      let  queue:=r.tail;
      queue:=tail;
      for each c such that g(r,c)=s≠fail do
        begin
          queue:=queue.s;
          ddm(s):=min{ ddm(s), ddm(r)+1};
        end
    end;
  for j:=0 to nst-1 do
    for each c such that g(j,c)=fail  do
      f(j,c):=min{ f(j,c), ddm(j)};
end
```

図2．10　failure関数を求めるアルゴリズム

図2．11に，パターンが図2．5と同じ場合，図2．9，図2．10のアルゴリズムで求められるＦＡＳＴ法のｐｍｍを示す．実線，破線，下線付き文字列の意味は図2．5と同じである．ただし，破線は関数fでなく関数ϕに相当している．

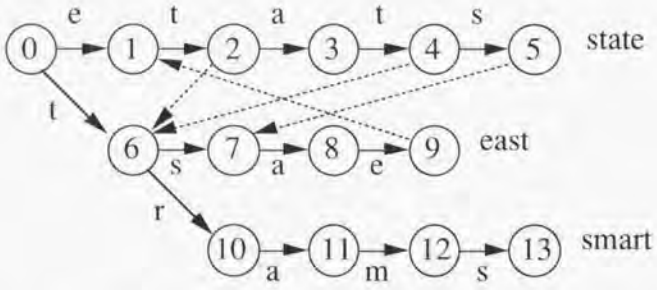


図2.11　ＦＡＳＴ法におけるパターン照合機械

ポインタの移動の向きが異なる点を除けば，ＦＡＳＴ法の照合に成功している間の動作はＡＣ法と全く同じである．しかし，照合失敗時の動作は異なり，ＦＡＳＴ法ではfailure関数によって，ポインタのシフト量を求める．ところで，failure関数はgoto関数が**fail**となる場合にしか定義されないからfailure 関数の値を負に変えることによってgoto関数と同一の関数にまとめることができる．この様にして求めたgoto関数gを表２．１に示す．

表２．１　goto関数の例

文字種／状態	0	1	2	3	4	5	6	7	8	9	10	11	12	13
a	-2	-4	3	-6	-7	-8	-2	8	-6	-7	11	-7	-8	-9
e	1	-4	-5	-6	-7	-8	-5	-5	9	-7	-6	-7	-8	-9
m	-3	-4	-5	-6	-7	-8	-5	-5	-6	-7	-6	12	-8	-9
r	-1	-4	-5	-6	-7	-8	10	-5	-6	-7	-6	-7	-8	-9
s	-1	-4	-5	-6	5	-8	7	-5	-6	-7	-6	-7	13	-9
t	6	2	-5	4	-7	-8	-5	-5	-6	-7	-6	-7	-8	-9
(other)	-4	-4	-5	-6	-7	-8	-5	-5	-6	-7	-6	-7	-8	-9

こうして求めたgoto関数gを用いたパターン照合のアルゴリズムは図２．１２のようになる．すなわち，ＡＣ法ではテキスト上のポインタは常に前方へ１だけ増加するが，ＦＡＳＴ法では照合に成功している間は後方へ１だけ戻り，失敗したときはgの絶対値分だけ前方に進む．

```
テキストの文字列x を x=a1a2…an とする.
goto関数をg, 出力関数をoutとする.
出力はx 中に出現するパターンとその位置である.

begin
  a0:=umc;  /*  umc は決して照合に成功しない文字  */
  g:=minlen;  /*  minlen はパターンの最小文字長  */
  while  q ≦ n  do
    begin
      state:=0;
      while  g(state, aq) > 0  do
        begin
          state:=g(state, aq);
          if out(state)≠empty  then  print q, out(state);
          q:=q-1;
        end
      q:=q-g(state, aq);
    end
end
```

図２．１２　文字列照合アルゴリズム

図２．１３に図２．２と同じテキストを用い，ＦＡＳＴ法の照合動作の例を示す．図２．１３でstateはｐｍｍの状態を，orderは照合の順番を示している．状態０のときテキスト上の文字が‘ｔ’であれば状態６に遷移し，探索対象から“state”が除外され，次に１つ手前のテキスト上の文字が照合される．照合に失敗すると状態とテキスト上の文字によって決まる量だけパターン群が右に移動する．こうして四角で囲んだところでパターンが検出される．図２．１３からわかるようにテキス

ト上の文字すべてを照合しないですむ場合がある．反対にテキスト上の同一文字が何回も照合されることもある．例えば図２．１３のテキスト“・・・seast・・・”の‘a’は３回照合される．

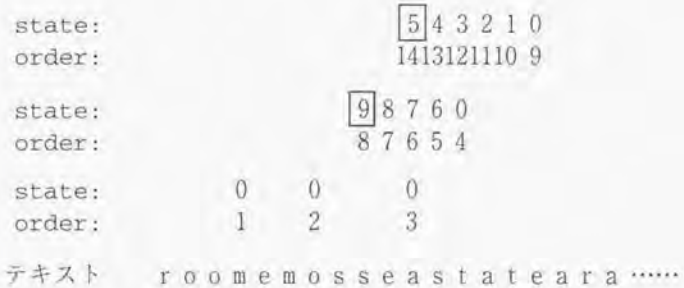


図２．１３　ＦＡＳＴ法による照合手順

２．４　あとがき

全走査法に属するいろいろな全文検索手法について解説し，ＢＭ法の「右端からの照合」というアイデアとＡＣ法の「パターン照合機械」のアイデアを結合することによって，複数パターンを同時に高速に探索する方法（ＦＡＳＴ法）を示した．

実は，ＦＡＳＴ法はパターン数が１つの場合にもＢＭ法より効率が良い．その理由は，テキストのa_jから後向きにa_iまで照合していって，そこで失敗した場合，ＢＭ法では$a_{i+1}a_{i+2}\cdots a_j$と一致する可能性がある位置までパターンを移動するわけだが，ＦＡＳＴ法では$a_ia_{i+1}\cdots a_j$と１文字分長くとっているためシフト量を大きくすることができるからである．ＦＡＳＴ法がＢＭ法より劣っている点はＦＡＳＴ法の方がパターンのシフト量を計算するために保持しておくための関数の記憶容量が大きくなることである．ＢＭ法では関数d，関数ddの為に「パターン長＋文字種の大きさ」の記憶容量を用意すればよいのに対し，ＦＡＳＴ法のgoto関数は「パターン長×文字

種の大きさ」の記憶容量を必要とする．このことは，文字種が多い場合には問題になりうる．記憶容量の削減法については第３章でふれる．

ＦＡＳＴ法がＢＭ法より実際の実行時間においても速い理由がもう一つある．それは，ＢＭ法では照合に失敗するたびに関数dと関数ddの大きい方を選ぶ演算を必要とするのに対し、ＦＡＳＴ法では１回の表探索で済むからである．そこで，文字種が多い場合にはより効果が期待できる関数dだけを用いることによって、ＢＭ法の実際の実行時間を短縮する方法（ＢＭＨ法）がHorspoolによって提案されている[5]．

複数パターンの高速探索法についてはKowalskiとMeltzerも提案している[6]．この方法は単純に言えば，パターン長が不揃いの場合，まず最短のパターンに合わせて後を切り揃え，それらを探索する．そして，切り揃えられた各パターンそれぞれにＢＭ法の関数d，関数ddを求めておき，照合に失敗したときはそれらを用いてパターン群をシフトする．この方法はかなり煩雑であり，そのうえ明らかにＦＡＳＴ法よりも効率が悪い．

Commentz-Walterも別の複数パターン探索法を提案している[7]．この方法は，シフト量の計算にｐｍｍを使うなど，ＦＡＳＴ法とかなり似ているが，関数dと関数ddを別々に求めていて，「$a_{i+1}a_{i+2}\cdots a_j$の情報しか用いない」こと，「照合に失敗するたびに関数dと関数ddの大きい方を選ぶ演算を必要とする」ことなどＢＭ法と同じ欠点を有しているので，明らかにＦＡＳＴ法よりも効率が悪い．そのうえ，ＦＡＳＴ法では最大の問題点は，探索漏れが生じない範囲でパターン群を動かせる最大の量をどのようにすれば求められるか，という点にあったが，Commentz-Walterのアルゴリズムにはこの計算法に間違いがあり，探索漏れが生じる可能性がある[8]．

ＦＡＳＴ法の効率の良さはアルゴリズムそのものから直感的には明らかであろうが，第３章で，効率について理論的に考察する．また，アルゴリズムの正当性を証明し，実験によって実際に高い探索効率が得られることを示す．

第3章　ＦＡＳＴ法の性質

3．1　まえがき

第2章では，いろいろな全文検索手法について解説し，複数パターンを同時に高速に探索する方法（ＦＡＳＴ法）の考え方と具体的なアルゴリズムについて示した．そして，パターン数が1つの場合でさえもＦＡＳＴ法の方がＢＭ法より効率が良いことも述べた．ＦＡＳＴ法の効率の良さはアルゴリズムそのものから直感的には明らかであるが，この章で効率について理論的に考察する．

いくら高速な探索方法であっても，常に正しく動作するものでなければ価値はない．この章ではＦＡＳＴ法のアルゴリズムの正当性についても証明する．さらに実験によって実際に高い探索効率が得られることを示す．

ＦＡＳＴ法がＢＭ法より劣っている点はgoto関数の記憶容量が大きくなることである．この削減方法についても検討を加える．

3．2　ＦＡＳＴ法の効率[1)]

アルゴリズムの効率を考える場合には，もちろん主動作の効率だけでなく，前処理に要する時間も考慮しなければならない．２．２．１で述べた「素朴な方法」は全く前処理時間を必要としない．ＫＭＰ法やＡＣ法ではｍを（全）パターンの長さとすると$O(m)$，ＢＭ法では$O(q+m)$（ｑは文字種の数）の前処理時間を必要とする．これに対してＦＡＳＴ法では$O(mq)$の前処理時間を必要し，他の方法と比べて劣っている．しかし，通常対象とするファイルはかなり大きいと考えられるので，（そうでなければ「素朴な方法」で十分である）前処理時間は主動作に要する時間に比べて無視できると仮定する．したがって，以下では文字列照合操作に直接関わる時間のみを考える．

文字列照合アルゴリズムの優劣は一般にprobe rate（テキスト１文字当りの照合回数）で評価される．この場合は，一般的にいう効率はprobe rateの逆数と考えてよい．

ＦＡＳＴ法のprobe rateをＱとすると，明らかに，最良時には

Ｑ＝１／minlen　　(minlen: パターンの最小長)

となる．最悪時としては照合の成功する確率が最大の場合を想定すれば良い．この

場合でもパターンが1つ見つかり，その直後で失敗する毎に最低1つは照合開始位置が右にずれるので，Qは

Q＝maxlen＋1　（maxlen: パターンの最大長）

となる．すなわち最悪時でもQがパターン数に依存しないという望ましい特徴を有している．

テキストもパターンもq種の文字集合からランダムに抽出されていると仮定して，Qの期待値を求めてみる．ただし，パターンの長さはすべてmとする．

今，パターンが1つの場合を考えて

テキストを　$\tau_1\tau_2\cdots\tau_j\cdots\tau_n$

パターンを　$\pi_1\pi_2\cdots\pi_s\cdots\pi_m$

で表すものとする．jがmより十分大きくてτ_jから始まるパターン照合の成功する確率がjに依存しない状態（定常状態）を考える．$P_{s,i}$をs回の照合成功の後s＋1回目で失敗して最初の照合位置からiだけずれたところから照合が再開される確率とすると，τ_jとπ_mから始まった照合はτ_{j-s+1}とπ_{m-s+1}との照合までは成功し，τ_{j-s}とπ_{m-s}の照合で失敗する．この時，パターンのシフト量がiとなることから，$\tau_{j-s}\tau_{j-s+1}\cdots\tau_j$は$\pi_{m-s}$から（後向きに）始まる照合では成功するが，それ以前から始まる照合で成功することはない．したがって，

s＝0のとき

$$P_{0,i}=P\Big(\big(\tau_j\neq\pi_m\big)\wedge\big(\tau_j\neq\pi_{m-1}\big)\wedge\cdots\wedge\big(\tau_j\neq\pi_{m-i+1}\big)\wedge\big(\tau_j=\pi_{m-i}\big)\Big)$$

0＜sのとき

$$\begin{aligned}P_{s,i}=P\Big(&\big(\tau_j=\pi_m\big)\wedge\big(\tau_{j-1}=\pi_{m-1}\big)\wedge\cdots\wedge\big(\tau_{j-s+1}=\pi_{m-s+1}\big)\wedge\big(\tau_{j-s}\neq\pi_{m-s}\big)\\&\wedge\big(\tau_{j-s}\tau_{j-s+1}\cdots\tau_j\neq\pi_{m-s-1}\pi_{m-s}\cdots\pi_{m-1}\big)\\&\wedge\big(\tau_{j-s}\tau_{j-s+1}\cdots\tau_j\neq\pi_{m-s-2}\pi_{m-s-1}\cdots\pi_{m-2}\big)\\&\qquad\vdots\\&\wedge\big(\tau_{j-s}\tau_{j-s+1}\cdots\tau_j\neq\pi_{m-s-i+1}\pi_{m-s-i}\cdots\pi_{m-i+1}\big)\\&\wedge\big(\tau_{j-s}\tau_{j-s+1}\cdots\tau_j=\pi_{m-s-i}\pi_{m-s-i+1}\cdots\pi_{m-i}\big)\Big)\end{aligned}\qquad(1)$$

ここで，$\tau_{j-s}\tau_{j-s+1}\cdots\tau_j\neq\pi_{m-s-1}\pi_{m-s}\cdots\pi_{m-1}$ 等は

$\big(\tau_{j-s}\neq\pi_{m-s-1}\big)\vee\big(\tau_{j-s+1}\neq\pi_{m-s}\big)\vee\cdots\vee\big(\tau_j\neq\pi_{m-1}\big)$ 等の略記であり，

$\tau_{j-s}\tau_{j-s+1}\cdots\tau_j=\pi_{m-s-i}\pi_{m-s-i+1}\cdots\pi_{m-i}$ は

$\left(\tau_{j-s}=\pi_{m-s-i}\right)\wedge\left(\tau_{j-s+1}=\pi_{m-s-i+1}\right)\wedge\cdots\wedge\left(\tau_j=\pi_{m-i}\right)$ の略記である．

（0＜s＜(m-1) で i＜(m-s) の場合）

ただし，(m-s)≦i のときは m-s-i≦0 となるから(1)式中の π_{m-s+i} 等に対応するものがなくなるので，その分を評価しなくてよいためP中の後半の項は短くなる．

s＝m-1 や s＝m のときも同様で，対応するものがない分，上述の式は短くなる．

ところで，(1)式のP中の各項は独立でない．例えば m＝3，s＝1，i＝2 のときは，$\tau_j=\tau_{j-1}$ の場合と，$\tau_j\neq\tau_{j-1}$ の場合を分けて考えれば

$$P\left(\left(\tau_j=\pi_3\right)\wedge\left(\tau_{j-1}\neq\pi_2\right)\wedge\left(\tau_{j-1}\tau_j\neq\pi_1\pi_2\right)\wedge\left(\tau_j=\pi_1\right)\right)$$

$$=P\left(\left(\tau_j=\tau_{j-1}\right)\wedge\left(\tau_j=\pi_3\right)\wedge\left(\tau_{j-1}\neq\pi_2\right)\wedge\left(\tau_{j-1}\tau_j\neq\pi_1\pi_2\right)\wedge\left(\tau_j=\pi_1\right)\right)$$

$$+P\left(\left(\tau_j\neq\tau_{j-1}\right)\wedge\left(\tau_j=\pi_3\right)\wedge\left(\tau_{j-1}\neq\pi_2\right)\wedge\left(\tau_{j-1}\tau_j\neq\pi_1\pi_2\right)\wedge\left(\tau_j=\pi_1\right)\right)$$

$\left[\left(\tau_j=\tau_{j-1}\right)\wedge\left(\tau_{j-1}\neq\pi_2\right)\right.$ のときも，$\left(\tau_j\neq\tau_{j-1}\right)\wedge\left(\tau_j=\pi_1\right)$ のときも

$\left.\tau_{j-1}\tau_j\neq\pi_1\pi_2\right.$ となるから］

$$=P\left(\left(\tau_j=\pi_3\right)\wedge\left(\tau_{j-1}\neq\pi_2\right)\wedge\left(\tau_j=\pi_1\right)\right)$$

$$=(q-1)/q^3$$

となって，各項の確率の積 $\left((q-1)(q^2-1)/q^5\right)$ とはならない．しかし，$q\gg1$ の場合はこの例からもわかるように各項を独立と考えてもその誤差は小さい☆．

☆　簡単に述べると，s＝0 の場合は各項は独立となるため誤差は生じない．

s＞0 のときは，($\tau_{j-s}\tau_{j-s+1}\cdots\cdots\tau_j\neq\pi_{m-s-k}\pi_{m-s-k+1}\cdots\cdots\pi_{m-k}$) のすべてを

($\tau_j\neq\pi_{m-k}$) で置き換えればPの下限(P_d)が，1で置き換えればPの上限(P_u)が得られる．各項を独立と見なしたPの推定値($\hat{P}$)は $P_d\leq\hat{P}\leq P_u$ となり，

絶対誤差は

(m-s)≦i のとき　$\dfrac{i-1}{q^{s+m-i+1}}$

(m-s)＞i のとき　$\dfrac{i-1}{q^{2s+2}}$

より小さくなるので，「各項を独立と考えてもその誤差は小さい」としてよい．

もう1つ考慮すべきことがある．例えば1つ前が $P_{0,1}$ から τ_j の照合に入る場合は $\tau_{j-1}=\pi_2$ となるから $P_{1,2}$ 等でP中の項を満たさない．τ_j から始まる照合は一般に(m-1)前の状態まで考えなくてはならないが，1つ手前の状態だけを考慮に入れる

こととする．つまり，時点kにおいて状態（x_k）がψ_jとなる確率は，（最悪の場合）時点k-1,k-2,…,k-m+1の(m-1)個の状態に依存するので

$P(x_k=\psi_j \mid x_{k-m+1}=\psi_{i,k-m+1}, x_{k-m+2}=\psi_{i,k-m+2}, \cdots\cdots, x_{k-1}=\psi_{i,k-1})$

と(m-1)重マルコフ連鎖を考えなければならないが，これを$P(x_k=\psi_j \mid x_{k-1}=\psi_i)$で近似するわけである．（2つ以上前の状態の影響は小さいのでこうしても誤差は少ない．なぜなら，i＝mのときはテキスト上で同じ場所の文字が照合されないから以前の状態に影響されないし，i≠mのときは，$P_{s,i} < 1/q$だから2つ以上前の影響が$1/q^2$を超えることはない．）

前述のように単純化のためにP中の各項を独立であると仮定しても誤差は小さいので，(1)式は

$$P_{s,i}=(t_s-t_{s+1})\left(\frac{q^{s+1}-1}{q^{s+1}}\right)^{i-1}\frac{1}{q^{s+1}}$$

と近似できる．

ここでt_sはτ_jから始まる照合がs回照合に成功する確率であり，

$$t_s=P\left(\left(\tau_j=\pi_m\right)\wedge\left(\tau_{j-1}=\pi_{m-1}\right)\wedge\cdots\wedge\left(\tau_{j-s+1}=\pi_{m-s+1}\right)\right)$$

である．上述のようにこのPは以前の状態に影響され各項の単純の積とはならない．τ_jから始まる照合の1つ手前の状態だけを考慮すれば，例えば$P_{2,2}$からτ_jの照合に入る場合には（パターン長が5以上の場合）5回照合に成功する確率はτ_{j-2}, τ_{j-3}, τ_{j-4}の照合成功は予め確定しているから，p_5ではなく条件付き確率のp_5 / p_3となる．したがって，t_s（の近似値）は，

$$\begin{aligned}
t_s=&\left(1-\sum_{h=0}^{m}P_{h,1}-\cdots-\sum_{h=0}^{m}P_{h,s-1}\right)p_s\\
&+\left(\sum_{h=0}^{m}P_{h,s-1}+\sum_{h=1}^{s-2}P_{0,h}\right)p_s/p_{s-1}\\
&\quad\vdots\\
&+\left(\sum_{h=k-1}^{m}P_{h,s-k}+\sum_{h=1}^{s-k-1}P_{k-1,h}\right)p_s/p_k\\
&\quad\vdots\\
&+\left(\sum_{h=s-2}^{m}P_{h,1}\right)p_s/p_{s-1}
\end{aligned}$$

となる．ここで，

$$p_s = 1/q^s$$

であり，p_s/p_k はｋ回の照合成功が分っている場合ｓ回照合が成功する条件付き確率である．また，$t_0 = 1$，$t_1 = p_1$，$t_{m+1} = 0$ である．

パターンがｋ個のときはパターン中のある連続したｓ個の文字がテキストの現在照合している部分と一致する確率 p_s は

$$p_s = 1 - \left(1 - 1/q^s\right)^k$$

となるので，以上をまとめると，（単純化のためパターン長はすべてｍとする）

$P_{s,i}$ の一般式は

●$0 \leqq s \leqq (m-1)$ のとき

$i < (m-s)$ の場合

$$P_{s,i} = \left(t_s - t_{s+1}\right) p_{s+1} \left(1 - p_{s+1}\right)^{i-1}$$

$i \geqq (m-s)$ の場合

$$P_{s,i} = \left(t_s - t_{s+1}\right) p_{m-i} \left(1 - p_{s+1}\right)^{m-s-1} \prod_{k=m-i+1}^{s} \left(1 - p_k\right)$$

●$s \leqq m$ のとき

$$P_{s,i} = \left(t_s - t_{s+1}\right) p_{m-i} \prod_{k=m-i+1}^{m-1} \left(1 - p_k\right)$$

（ただし $\prod_{k=a}^{b} x_k$ はａ＞ｂのときは１とする）

となる．

これらの $P_{s,i}$（$0 \leqq s \leqq m$，$1 \leqq i \leqq m$）と t_j（$1 \leqq j \leqq m$）に関する連立方程式を解けば $P_{s,i}$ と t_j のそれぞれの値が求まる．

位置 τ_j から始まる照合は失敗するまで延べ $\sum_{j=0}^{m} t_j$ 回の照合を必要とするが，失敗したときには $\sum_{i=1}^{m} i\left(\sum_{s=0}^{m} P_{s,i}\right)$ だけ次の照合開始位置が進むので probe rate Q は

$$Q = \sum_{j=0}^{m} t_j \Big/ \sum_{i=1}^{m} i\left(\sum_{s=0}^{m} P_{s,i}\right) \qquad (2)$$

となる．

ｑがｍ，ｋに対して十分大きい（$q \gg mk$）ときには $r = 1/q$ とすると

$$p_j = kr^j$$

となるから

$$P_{0,i} = (t_0 - t_1)kr \qquad (1 \leqq i \leqq (m-1) \text{ のとき})$$
$$P_{0,m} = (t_0 - t_1)(1 - (m-1)kr)$$

$1 \leqq s \leqq (m-1)$ のとき

$$P_{s,i} = (t_s - t_{s+1})kr^{s+1} \qquad (1 \leqq i \leqq (m-s-1) \text{ のとき})$$
$$P_{s,i} = (t_s - t_{s+1})kr^{m-i} \qquad ((m-s) \leqq i \leqq (m-1) \text{ のとき})$$

$s = m$ のとき

$$P_{m,i} = t_m kr^{m-i} \qquad (1 \leqq i \leqq (m-1) \text{ のとき})$$
$$P_{m,m} = t_m(1 - kr)$$

$t_j = kr^j$ $(1 \leqq j \leqq m)$ となるから

$$\sum_{j=0}^{m} t_j = 1 + kr + kr^2 + \cdots + kr^m \approx 1 + kr$$
$$\sum_{i=1}^{m} i\left(\sum_{s=0}^{m} P_{s,i}\right) \approx m - \frac{m^2 - m}{2}kr$$

したがって

$$Q \approx \frac{1 + kr}{m\left(1 - \frac{m-1}{2}kr\right)} \approx \frac{(1+kr)\left(1 + \frac{m-1}{2}kr\right)}{m}$$

$(q \gg mk$ より $\frac{m-1}{2}kr \ll 1$ だから$)$

$$\approx \frac{1}{m} + \frac{(m+1)k}{2mq} \qquad (3)$$

となる．

3．3　正当性の証明[2),3)]

アルゴリズムの正当性を証明するには，健全性，完全性，停止性を証明すればよい．文字列探索アルゴリズムに即して言えば，これらは

健全性：間違ったパターンを出力しないこと

完全性：対象テキスト中に含まれるパターンを漏れなく出力すること

停止性：無限ループに入ったりすることがなく，必ず停止すること

となる．

まず，停止性は図２．１０のアルゴリズムにより，どの f(j,c) も depth(j) より大きくなることから明らかである．健全性についても，図２．９のアルゴリズムに基づく状態遷移は図２．１１のオートマトン上を動くだけであるから，関係ないパターンを検出することは決してないことがわかる．

完全性の証明のためには準備が必要である．証明に必要な集合や写像，関係などの定義や関数の特徴付けを行ないつつ，完全性の証明を与える．

３．３．１　完全性の証明

文字列 w の prefix を PRE(w)，suffix を SUF(w) とする．つまり，アルファベット集合をΣとしたとき，Σ*の任意の要素 w に対し，

$$\mathrm{PRE}(w)=\left\{x\in\Sigma^{*}\middle|\exists y\in\Sigma^{*}:xy=w\right\}$$

$$\mathrm{SUF}(w)=\left\{y\in\Sigma^{*}\middle|\exists x\in\Sigma^{*}:xy=w\right\}$$

と定義する．文字列集合 X に対しては，

$$\mathrm{PRE}(X)=\bigcup_{w\in X}\mathrm{PRE}(w),\quad \mathrm{SUF}(X)=\bigcup_{w\in X}\mathrm{SUF}(w)$$

とする．パターンの集合を K = { a_1, a_2, …, a_k }，状態遷移グラフに現われる状態全体の集合を S とし，写像 state : SUF(K) →S を次式で定義する．

$$\begin{cases} state(\varepsilon)=0, \\ state(\sigma w)=g(state(w),\sigma) & (w,\sigma w\in \mathrm{SUF}(K),\sigma\in\Sigma) \end{cases}$$

S 上の二項関係 $\vdash_{\phi}$ を次のように定義する．s, t を S の任意の要素とするとき，

$$s \vdash_{\phi} t \quad\Leftrightarrow\quad \phi(s)=t$$

とする．

図２．９で求めた関数 depth, emap と上記の定義には以下の関係がある．ただし，u は S の任意の要素で，st=state(u) である．

$$\mathrm{depth}(st)=|u|$$

$$\mathrm{emap}(j)=state(a_j)$$

さて，完全性を証明するには次にあげる定理１を用いて，その後の補題１を証明すればよい．

定理１．$u\in\mathrm{SUF}(K)$，$a\in\Sigma$, $au\notin K$ とする．いま，

(1)　$auv\in\mathrm{SUF}(K)$

(2)　$\exists y\in\mathrm{SUF}(u):yv\in K$

のいずれかを満たす $v \in \Sigma^+$ のうち長さが最小となる v を選び，st = state(u) とおけば，

$$f(st,a) = |u| + |v|$$

である．

この定理の証明を与えるためには，さらに3つの補題を証明しなければならないが，それを後回しにして，先に補題1を証明する．

補題1．テキストの文字列を $a_1a_2\cdots a_n$ とし，a_0=**umc**（決してパターンに含まれない文字）とする．いま，

$$u = a_{p+1}\cdots a_q \in \mathrm{SUF}(K) \quad (1 \le p \le q \le n)$$

$$st = state(u)$$

とおく（ただし，p=q のときは u=ε とする），

また，$g(st,a_p)$=**fail**（または0）であるとする．このとき，次の式が成り立つ．

$$\left[a_i \cdots a_j \in K(1 \le i \le j \le n), q < j\right] \Rightarrow p + f(st,a_p) \le j$$

（これは，探索漏れが起きないことを示している）

証明

(1) $i \le q$ のとき，$v = a_{q+1}\cdots a_j$ とおく．定理1より，

$$\begin{aligned} f(st,a_p) &\le |u| + |v| \\ &= (q-p)+(j-q) \\ &= j-p \end{aligned}$$

である．

(2) $q < i$ のとき，$v = a_i \cdots a_j$ とおく．$v \in K$ であるから，定理1より，

$$\begin{aligned} f(st,a_p) &\le |u| + |v| \\ &\le (q-p)+(j-i+1) \\ &= (q-i+1)+j-p \\ &\le j-p \end{aligned}$$

となる．

（ＱＥＤ）

以上より，定理1が証明できれば完全性が証明できることがわかった．それでは，定理1の証明に戻ろう．そのためにまず次の補題2を証明する．これは，

図２．１０のアルゴリズムで用いた関数 ϕ を特徴付けるものである．

補題２．$u \in \mathrm{SUF}(K) - \{\varepsilon\}$，$s = state(u)$　とする．集合

$$(\mathrm{PRE}(u) - \{u\}) \cap \mathrm{SUF}(K)$$

の要素のうち長さが最大のものを v とすれば，

$$\phi(\mathrm{s}) = state(v)$$

となる．

証明　|u| に関する帰納法により証明する．

|u| = 1 のとき，$\phi(s) = 0, (\mathrm{PRE}(u) - \{u\}) \cap \mathrm{SUF}(K) = \{\varepsilon\}$ であるから明らかである．

|u| > 1 のとき，$u = ax$ $(a \in \Sigma,\ x \in \mathrm{SUF(K)})$ とする．集合

$$(\mathrm{PRE}(u) - \{u\}) \cap \mathrm{SUF}(K)$$

の要素を長さの長い順に，　$x_1, x_2, \cdots, x_n = \varepsilon$　とし，

$r_0 = state\,(x)$，

$r_i = state\,(x_i)$　　(i = 1,2,⋯,n)

とすれば，帰納法の仮定により，

$\phi\,(r_i) = r_{i+1}$　(i = 0,1,2,⋯,n-1)

となる．

(1) $g\,(r_m, a) \neq$ **fail** となる m ($1 \leqq m < n$) が存在するとき，m をそのような最小のインデックスとすれば，

$$\phi\,(r_i) = g\,(r_m, a) = state\,(ax_m),$$

$$ax_m \in (\mathrm{PRE}(u) - \{u\}) \cap \mathrm{SUF}(K)$$

となる．m のとり方から，ax_m は $(\mathrm{PRE}(u) - \{u\}) \cap \mathrm{SUF}(K)$ の要素のうち長さが最大のものであることがわかる．

(2) 任意の i ($1 \leqq i < n$) に対して，$g\,(r_m, a) =$ **fail** であるとき，

$$\phi(\mathrm{s}) = 0,\ (\mathrm{PRE}(u) - \{u\}) \cap \mathrm{SUF}(K) = \{\varepsilon\}$$

であるから明らかである．

（ＱＥＤ）

補題２から直ちに次の系が得られる．

系．　$u, v \in \mathrm{SUF}(K)$，$u \neq \varepsilon$ であるとき次が成立する．

$$v \in \mathrm{PRE}(u) - \{u\} \iff state\,(u) \vdash_{\phi}^{+} state\,(v)$$

補題３．　$au \notin \mathrm{SUF}(K)$ なる任意の対 $(u,a) \in \mathrm{SUF}(K) \times \Sigma$ に対して，

$$H(u,a) = \left\{ v \in \Sigma^+ \middle| auv \in \mathrm{SUF}(K) \right\}$$

とおく．このとき，st = state (u) ならば，

$$shift1(st,a) = \begin{cases} \min\{|u+v| \mid v \in H(u,a)\} & (H(u,a) \neq \emptyset \text{ のとき}) \\ \textbf{large} \ \text{(undefined)} & (\text{そうでないとき}) \end{cases}$$

が成立する．

（関数shift１は図２．１０のアルゴリズムの関数 ddm で更新される前の（つまり，図２．１０のAで更新される）関数 f を表し，図２．８(a)の場合のシフト量に相当している．　図３．１参照）

証明　g(st,a)=**fail**（または0）となる各(st,a)に対し，

$$A(st,a) = \left\{ r \in S \;\middle|\; \begin{array}{l} r \neq 0, \ \ g(r,a) \neq \textbf{fail} \\ \exists r_0, \ldots, \exists r_n \in S : \\ \ r_0 = r, \ \ r_n = st, \\ \ r_{i+1} = \phi(r_i) \ \ (0 \leq \forall i < n), \\ \ g(r_i,a) = \textbf{fail} \ \ (1 \leq \forall i < n) \end{array} \right\}$$

とおく．図２．１０のアルゴリズムから

$$shift1(st,a) = \begin{cases} \min\{depth(r) \mid r \in A(st,a)\} & (A(st,a) \neq \emptyset \text{ のとき}) \\ \textbf{large} & (\text{そうでないとき}) \end{cases}$$

であることは明らかである．また，

$A(st,a) \subseteq \{state(uv) | v \in H(u,a)\}$　なので，

$$H(u,a) = \emptyset \ \Rightarrow \ A(st,a) = \emptyset$$

は明らかである．したがって，

$$\min\{|u+v| \mid v \in H(u,a)\} \ = \ \min\{depth(r) \mid r \in A(st,a)\}$$
$$(H(u,a) \neq \emptyset \text{ のとき})$$

が成り立つことを示せばよいことになる．

$$|v| = \min\{|z| \mid z \in H(u,a)\}$$

となる $v \in H(u,a)$ をとり，r = state (uv) とおく．このとき，

$$\exists r_0, \ldots, \exists r_n \in S : \left[r_0 = r, \ r_n = st, \ r_{i+1} = \phi(r_i) \ \ (0 \leq \forall i < n) \right]$$

である．$r \in A(st,a)$ であることを示したい．そのためには，

$$1 \leq \forall i < n : \ g(r_i,a) = \textbf{fail}$$

を示せばよい．そこで，

$$1 \le \exists j < n:\ g(r_j, a) \neq \mathbf{fail}$$

と仮定する．このとき，$state(uv') = r_j$ なる $v' \in \mathrm{SUF}(K) - \{\varepsilon\}$ をとると

$$v' \in H(u,a),\ |v'| < |v|$$

となり，矛盾する．これによって，

$$A(st,a) \neq \varnothing,$$

$$\min\{|u+v| \mid v \in H(u,a)\} \ \geq\ \min\{depth(r) \mid r \in A(st,a)\}$$

が示された．逆向きの不等号は明らかである．

（ＱＥＤ）

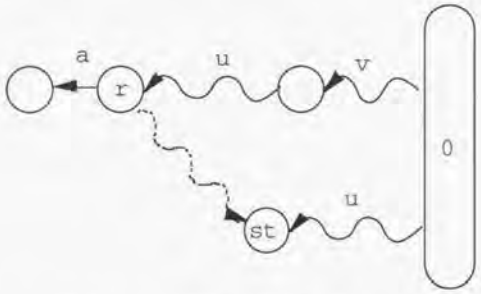


図３．１　**shift1** 関数の構成

次の補題は，図２．１０のアルゴリズムの関数 ddm を特徴付けるものである．

補題４． SUF(K) の空語 ε でない任意の要素 u に対して，

$$V(s) = \{v \in \Sigma^+ \mid \exists y \in \mathrm{SUF}(u) : yv \in K\}$$

とおく．ただし，s = state (u) である．このとき，

$$ddm(s) = \min\{|uv| \mid v \in V(s)\}$$

が成り立つ．

（この関数は図２．８ (b) の場合のシフト量に相当している．　図３．２参照）

証明　任意の状態 s について，

$$B(s) = \{r \in state(K) \mid r \vdash_\phi^+ s\}$$

とおくと，

$$ddm(s) = \begin{cases} minlen & \text{if } s = 0 \\ \min\left(\{depth(r) \mid r \in B(s)\} \cup \{ddm(fath(s)) + 1\}\right) & \text{if } s > 0 \end{cases}$$

となる．ただし，fath (s) は s の親の状態を示す．

$s=0$ のとき，成り立つことは明らかである．$s>0$ のとき，$s' = fath(s)$ とすれば，

$$V(s) = \left\{ v \in \Sigma^+ \mid uv \in K \right\} \cup V(s')$$

となるので，この場合にも成り立つ．ゆえに，補題は明らかである．

（ＱＥＤ）

補題３と補題４が成り立てば，定理１が成り立つことは自明である．これで，完全性も証明されたことになる．

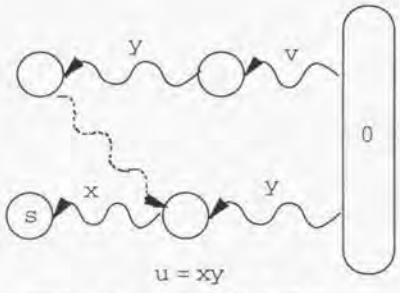


図３．２　*ddm* 関数の構成

３．４　検索実験

３．２で述べたようにＦＡＳＴ法の効率は文字種の数ｑ，パターン長ｍ，パターン数ｋの組み合わせによって大きく変化する．そこでｑ，ｍ，ｋをいろいろに変えて，ＦＡＳＴ法による文字列照合の実験を行った．実験には一様乱数で発生された１Ｍバイトのテキストを３種，パターンを１６種使用した．ｑとｍの組み合わせに対して，ｋを変化させて効率（probe rate）を測定した結果を図３．３に実線で示す．図中には推定式（３．２の(2)式）から得られる計算値も一点鎖線で示す．近似式（３．２の(3)式）による計算値も破線で示す．ＡＣ法では効率は常に１であるので，図中の二点鎖線より下ではＡＣ法より効率が良いことになる．ｑ＝２の場合を除いて推定式と実験結果がよく一致することがわかる．ｑ＝２のとき誤差が大きくなるのは，３．２で述べたように推定式を求める際に単純化のために近似を入れているためで，

この影響はqが小さいほど大きくなる．近似式の方もqが大きくkが小さい時にはよく一致することがわかる．

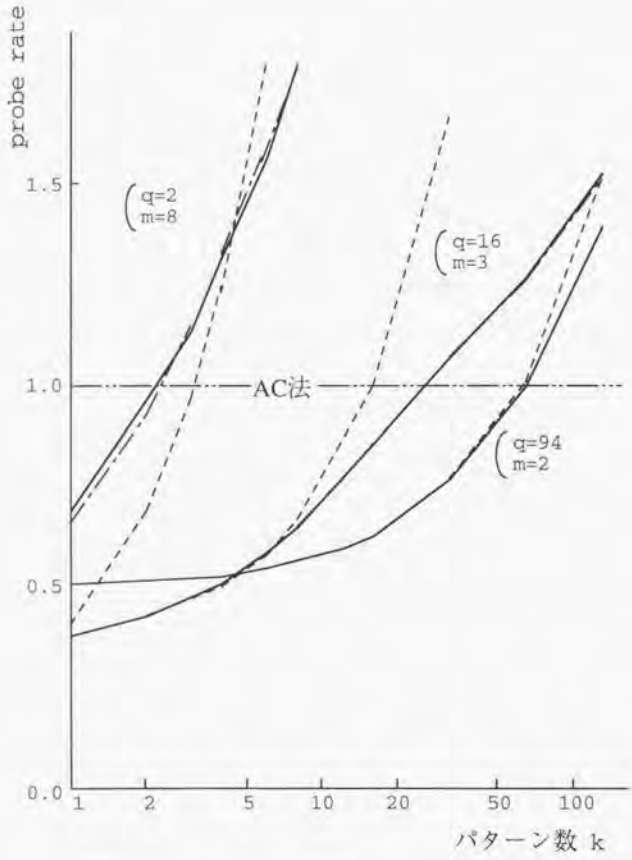


図3．3　ＦＡＳＴ法の **probe rate**

図３．４には，推定式から求められるprobe rateが１となる，ｋとｍとの関係をｑをパラメータとして示す．実線より下の領域でＡＣ法より効率が良いことになる．ｑがかなり小さい場合（例えばｑ＝５）でもパターン長が長くなるにつれＡＣ法より有利となる領域が急速に拡大することがわかる．

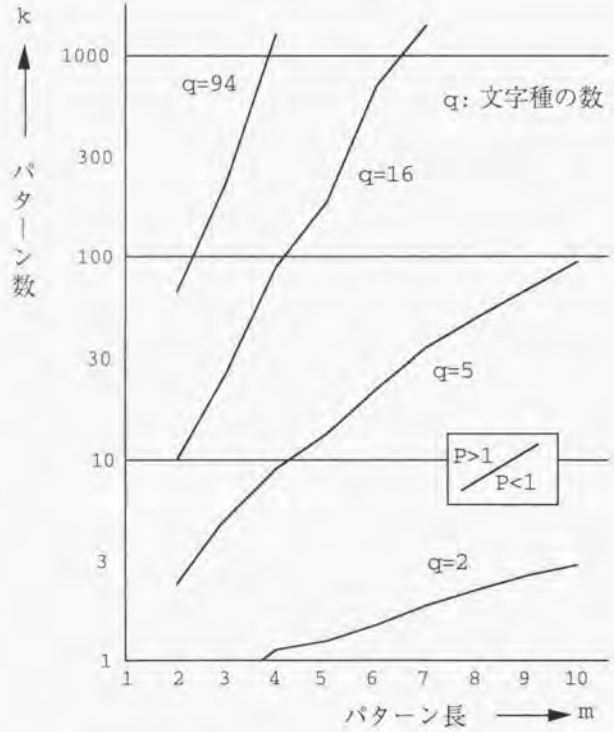


図３．４　**probe rate**が１となるパターン長**(m)**とパターン数**(k)**の関係

ところで，probe rate は文字列が探索される割合を示したものに過ぎないので，実際に計算機上で実行した場合に上記の効率が得られるかは疑問である．２．４で述べたようにHorspoolはＢＭ法にprobe rate が下がるような改修を行って，実行時間の向上を図っている．そこで，Ｃ言語でコーディングして，ＡＣ法とＦＡＳＴ法の実際の実行時間の比較を行った．図３．５，図３．６にその結果を示す．ＡＣ法で１個のパターンを検索したときを１として，それとの比で実行時間を示している．ただし，ｐｍｍを作成するための前処理時間やディスクからデータを読み取るための時間は除いている．前者はデータ量が大きくなれば，無視できるものであるし，データをメモリー上に転送する時間はハードウェア構成に異存するだけでＡＣ法とＦＡＳＴ法では変わらないからである．つまり、データがメモリー上にあるという条件で探索時間のみを測定した．使用した計算機は Sun Sparc Station 5 である．

図３．５，図３．６を見るとprobe rate と実測値が良く一致していることが分かる．特に、パターンが少ない場合には誤差が極めて少ない．パターン数が増加するにつれてprobe rate より大きな値を取るようになるが、これはパターンを発見したときに出現パターンの種類と出現位置などを記録する処理時間が必要となるからである．ＡＣ法の方も同様で、２点鎖線で示すように実際にはパターン数が増加すると探索時間がパターンの記憶処理の分だけ増加している．各パターン数毎にＡＣ法との比を求めたものを破線で示してあるが、この線はprobe rate よりも下にくることが分かる．つまり、ＡＣ法との比較でいうなら図３．４で示したよりも若干ＦＡＳＴ法の方が有利な領域が広いことが想像される．

実際の探索時間であるが、Sun Sparc Station 5 で１ＭＢのランダムデータを探索するのに，文字種１６，パターン長３の場合，パターン１個で 0.158秒，８個で 0.264秒，６４個で 0.580秒であった．実際の探索例で考えて見ると，ＡＰ電１年分の記事（約２５０ＭＢ）から，８個のキーワードを探索するなら（データをディスクから転送する時間を除いて）約６６秒で済むことが分かる．（英語の文字の単純頻度を考慮したエントロピーは４ビット程度と言われている．８個のキーワードの最短の長さはせいぜい３くらいだと思われるから、文字種１６，パターン長３，パターン数８と考えても大差はない．もし、最短のパターン長が４なら約５３秒になる．）これは、通常の用途には十分に実用的な速度であるということができる．

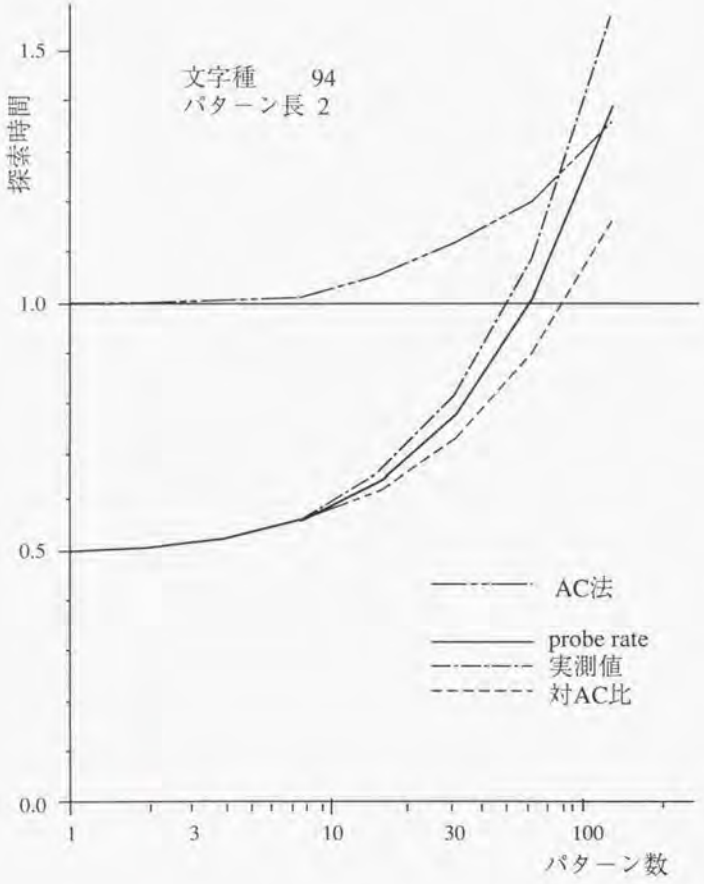


図３．５　実際の探索時間（文字種９４、パターン長２の場合）

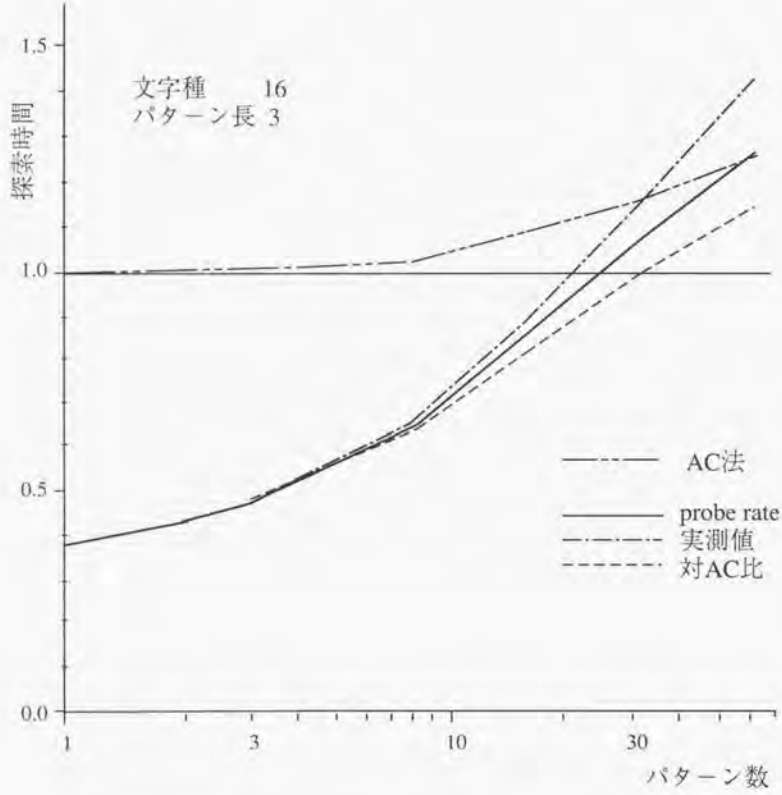


図３、６　実際の探索時間（文字種１６、パターン長３の場合）

さて，直感的には，ＦＡＳＴ法の probe rate は，ｍやｋに対して単調増加，ｑに対しては単調減少するように思われる．しかし，ｑが大きく，パターン長が長いときにはｋに対して単調増加にはならない．この１例（q=94，m=10のとき）を図３．７に示す．実測値（実験規模は図３．３と同様）は×印で示すが，推定値とよく一致していることがわかる．これを見ると，パターン数に関して探索効率が単調増加でないこと，つまりパターン数が増えると反対に効率が増す部分が存在することがわかる．この理由は定性的には次のように考えられる．

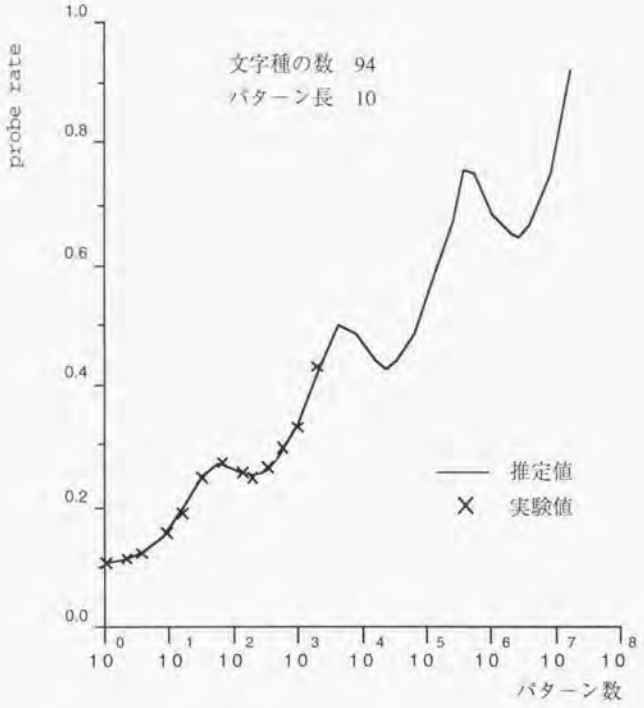


図３．７　長パターン時のＦＡＳＴ法の**probe rate**

一般にパターン数が増すと最初の照合に成功する可能性が多くなる（あるいは失敗してもパターン群のどこかと一致するのでシフト量が小さくなる）ので効率が低下する．パターンがある程度以上多くなると最初の照合で失敗する可能性は極めて小さくなる．ところが，この場合２文字連続して照合に成功する可能性は小さいので，照合に失敗したときのパターン群のシフト量は（パターン群のどこかと２文字連続して一致する可能性は小さいので）大きくなる．この結果，最初の照合で失敗するがパターン群のどこかと一致するために少ししかパターン群をシフトできない場合に比べて，最初の照合に成功するけれども次の照合（２文字分の照合）に失敗するために大きなシフトが起きるときの方が平均的に照合回数が少なくなることが起こり得るのである．もちろん，パターン数がさらに増加すると２文字連続の照合成功可能性が増してくるので再び効率は低下する．より詳細に述べると，推定式（３．２の(2)式）の分子（照合回数の期待値）は単調増加となるが，分母（シフト量の期待値）はそうならない．例えば，図３．７（q=94，m=10）の場合，パターン数が57，186，4638のとき，分子はそれぞれ1.4649，1.8849，2.4161となり分母はそれぞれ5.4297，7.4253，4.8735となる．

この現象はパターンが長いほど，また文字種が多いほど起こりやすくなる．例えばｑ＝９の時にはｍ≧１７でこの現象が起こるが，ｑ＝５７ならｍ≧６で起こる．また，パターンが非常に長い場合には，図３．７のようにこの現象が２回以上起こることがある．これは，ｊ回の照合に成功しやすく，ｊ＋１回の照合は成功しにくいとき（つまり $t_j >> t_{j+1}$ のとき）に起こりうるのである．

この現象は，例えば図３．７でprobe rateの曲線が山となるようなパターン数の場合，ダミーのパターンを加え谷の部分に移動させることにより，効率の改善が図れることを示唆している．実際にそうなりうることを例で示す．図３．７で，最初の山はパターン数が57のところで，このときのprobe rateは0.2698である．最初の谷はパターン数が186のところでprobe rateは0.2539である．したがって，もし探したいパターンの数が57付近であれば，ダミーのパターンを130程度加えて探索すれば0.016近くprobe rateが低下することになる．もちろん，この場合パターンが検出された後で，本当のパターンかダミーパターンかを判定する処理が必要となる．このためには本当のパターンだけで作ったgoto関数（パターン照合機械）を用意しておいて，検出されたパターンをもう１度これで照合すればよい．これは，等価的なprobe rateの増加を招くが，この例の場合，パターン数が186のときの，パターン検

出確率はわずか3.47×10^{-18}であり，最悪でも１回当りパターン長分しか検出の手間はかからないから，全く問題にはならない．ただし，以上の考察は３章での前提（つまりテキストもパターンもランダム）に基づいていることに注意を要する．

３．５　メモリー削減手段[4]

ＦＡＳＴ法の欠点はgoto関数の記憶に多大のメモリー容量と前処理時間を必要とする点である．例えば，goto関数１個当たり２バイトのメモリーを使用すれば，ｐｍｍの状態数がｓで文字種がｑの場合２ｓｑバイトのメモリーが必要である．パターン数をｋ，平均のパターン長をｍとすれば，ｓは大体ｋｍ程度になる．したがって文字種が少なくパターンも少ないときには記憶容量もあまり大きくならない（q=53，k=10，m=3 なら約３Ｋバイト）が，文字種が多くパターン数も多いときには莫大な記憶容量が必要となる．例えば，ＪＩＳ漢字コードを扱うときにはｑは約7000になり，k=1000，m=3 なら約４２Ｍバイトも必要である．この場合単に表が大きくなるばかりでなく，表の初期化等の前処理時間も長くなってしまう．

有川らはこの様なときにこれを改善する方法を提案している．１つは“２×ｎ実現”法[5]を用いる方法である．パターンが第２章と同じ“state”，“east”，“smart”の場合の例を表３．１に示す．これはｐｍｍを２×ｎの表で実現する方

表３．１　ｐｍｍの２×ｎ実現

（ａ）ＡＣ法の場合

state	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
input	!	s	t	a	t	e	?	m	a	r	t	?	e	a	s	t	?
failure	/	12	7	1	1	1	13	1	1	1	1	1	0	1	1	7	3

（ｂ）ＦＡＳＴ法の場合

state	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
input	!	e	t	a	t	s	?	t	s	a	e	?	r	a	m	s	?
failure	/	7	1	8	1	8	9	0	12	1	1	2	1	1	1	1	1

法で，goto関数はgoto(s,input)=s+1として復元できる．この表においては，初期状態は1であり，！は任意の文字を，／は無定義を，？は計算機で使われない文字を示している．この方法を用いれば，画期的にメモリーを削減することが可能であるが，この方法では表の参照回数が多くなってＦＡＳＴ法の利点が著しく損なわれてしまう．参照回数が多い状態0（表3．1では状態1に相当）からの遷移だけを表2．1の形で残し，それ以外を“2×n実現”法で行えば，この欠点はかなり救済されると思われる．

有川らは別のメモリー節減手段も提案している[6]．この方法は文字コードを2つの部分コードに分割し，1回の遷移を2回の連続した遷移に置き換える方法で，“文字符号分割による実現”と有川らは呼んでいる．パターン照合機械（ｐｍｍ）をこの方法で表現した例をもとのｐｍｍとともに図3．8に示す．もとのパターンは“AC”，“BAA”，“BB”であり，文字コードはそれぞれ部分コードで

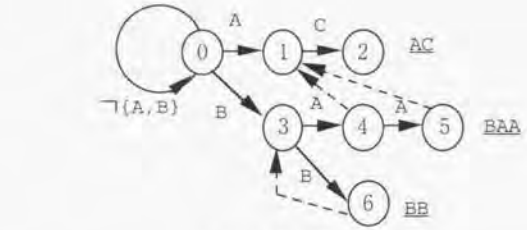
A=aa，B=ab，C=ba

で表されると仮定している．図3．8で大きな円は主状態を小さな円は中間状態を示している．主状態と中間状態を分けて扱っているのは誤検出を防ぐためであり，主状態からのfailure遷移の相手は主状態でなければならず，中間状態からのfailure遷移の相手は中間状態でなければならない．破線はfailure遷移を示しているが，主状態2，4，5から状態0への遷移，8以外の中間状態から状態11への遷移は省略してある．

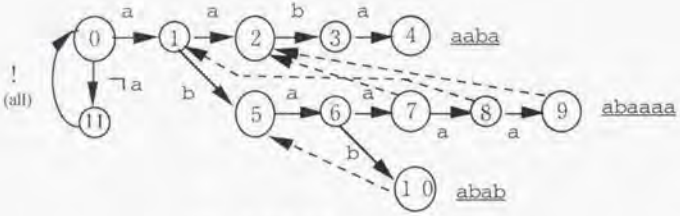
この方法はＦＡＳＴ法にとってより現実的だと考えられる．例えば，ＪＩＳ漢字コードの2バイトを上位バイトと下位バイトに分割すれば，文字種は94で済むことになる．前と同様k=1000，m=3(×2)を仮定すれば，約1．1Mバイトですむことになる．つまりこの場合の記憶容量は元の約$2/\sqrt{q}$倍に減少する．ただし，その様な分割符号に対してＦＡＳＴ法を直接適用すれば1バイトずれた誤った検出が起こる可能性がある．これを回避するためには，テキストおよびパターン中の文字を牛島らの方法[7]等によってすべて同じ符号長に変換しておいてから，上述したように中間状態を導入して，上位バイトから下位バイトへの遷移と下位バイトから上位バイトへの遷移を分離すればよい．あるいは牛島らが行なったように，パターン検出後にテキスト上の位置情報からずれ読みか否かを判定すればよい．

メモリーと効率の間にはトレードオフが存在する．例えば，文字種が少なければ文字をまとめて扱うことによって，走査回数を減らすことによって効率化できる．

この場合，当然のことだが，文字種が増えるのでパターンを保持するためのメモリー容量が増大する．その逆は上述した通りである．文字種が多いときは文字コードを分割してメモリーを減らすことができるが，これは等価的にテキスト増大（ｎ分割ならｎ倍）を招くので，効率が低下する．しかし，ＦＡＳＴ法の場合（ＡＣ法とは違って），パターン長は長くなるのでその分効率が良くなり，分割による効率低下の影響は少ない方法であるといえる．



（ａ）もとのｐｍｍ



（ｂ）文字符号分割によるｐｍｍ

図３．８　パターン照合機械の文字符号分割による実現

3．6　あとがき

本章では，ＦＡＳＴ法に関する諸性質，つまり効率，正当性の証明，実験結果，メモリー削減方法について述べた．まず，効率を求めるために，照合の過程をマルコフモデルで近似した．これをもとに文字の照合に成功する確率と文字列のシフト量の期待値を計算し，多元一次方程式を解くことにより，効率（probe rate）が推定できることを示した．さらに，文字種がパターン長とパターン数の積より十分大きい場合には簡単な式で表せることを示し，１／パターン長 に近い probe rate で検索が可能であることを証明した．

アルゴリズムが本当に正しく動作するか否かは重大な関心事である．いくら，高速な方法でも，誤検索（不要なものも出力する，あるいは検索漏れがある）があるのでは何にもならない．ＦＡＳＴ法の場合，停止性と健全性はほとんど自明であった．完全性（検索漏れがないこと）を証明するために，３つの補題と１つの定理を証明し，そこから導きだされる補題を証明することでアルゴリズムの正当性を証明した．

ＦＡＳＴ法の効率の良さは直感的には明らかであるが，実際にはどれくらいなのかは興味のあるところである．また，３．２で示された効率の計算法が実際とどれくらい一致するのかも気にかかるところである．実験を行なって，実際に効率の良い検索が可能なこと，効率の推定式は文字種が少なくない場合にはよく実験値と一致することを示した．さらに，文字種をパラメータとしてパターン長を横軸として，probe rate が１となるパターン数をグラフで示し，ＡＣ法より優れている領域を明らかにした．文字種が少ない場合（例えば５）でもパターン長が長くなると急速にＡＣ法より優れた領域が拡がることが明らかになった．このグラフを用いれば，ＦＡＳＴ法を使うべきか，ＡＣ法を使うべきかをユーザはあらかじめ検討することができる．probe rate と実際にプログラミングしたときの探索時間の差異も気になるところである．実験によって、ＡＣ法に対してほとんどprobe rate と一致した効率で検索が可能なことを示した．また，ＦＡＳＴ法はパターンが長い場合，直感に反してパターン数に対して効率が単調減少にならないことを示した．さらに，その理由についても考察を加えた．

ＦＡＳＴ法の最大の欠点は，goto関数の記憶に多大なメモリーと前処理時間を必要とすることである．言い換えればメモリーを多く消費することで，効率を上げて

いる方法と言うことができる．文字種が少ない場合には問題とはならないが，日本語コードなど文字種が多い場合には問題となってくる．メモリー削減手段として２つの方法を上げ，そのうち“文字符号分割による実現”がより現実的な方法であることを示した．また，２バイトコードを扱うときの問題点と対処方法についてもふれた．

ＦＡＳＴ法のもう１つの欠点はワイルドカードが使えないことである．文字列検索では"abc*xyz","table?.1"（*は任意の文字列；?は任意の１文字）のように“任意”を指定したいことがよくある．ＡＣ法ではどの文字とも一致する状態を作ってやり，それを考慮してfailure関数を作成すれば簡単にワイルドカードに対応できるがＦＡＳＴ法ではそれができない．もともと，完全一致を前提にしてそれを拠り所として効率を上げているアルゴリズムであるからワイルドカードへの対処が困難であることは当然である．しかし，"abc*"や"*xyz"のようにワイルドカードが後ろあるいは前にしかないのなら，"abc"や"xyz"とし、ワイルドカードを無視すれば良いし，"abc?vwxyz"のようにパターン長が長くてワイルドカードが前方にあり（後続文字が短くないという意味），任意長でなければ、?の部分を全文字種に置き換えても（その分パターン数が増えるが），この章で示した理論や実験結果からそれほどＦＡＳＴ法の利点は損なわれないことが予想される．

ＦＡＳＴ法は実際にすでに機械翻訳の訳語候補を目標言語のテキストと照合して決定する手法[8)]の中でも使われている．現在，著者は自然言語処理の研究に従事しており，文字列検索が必要な様々な処理にＦＡＳＴ法を利用していくつもりである．

第４章　ＦＡＳＴ法の２次元パターン検索への拡張

４．１　まえがき

最近，土地利用区分図などのメッシュデータのコンピュータ利用や，デスクトップパブリッシングの発展に伴ない，文字列のような１次元パターンだけでなく，２次元パターンの高速な探索法も求められている．

１次元パターンの探索アルゴリズムを２次元に拡張した例としては，BirdやZhu-Tadaoka や Baeza-Yates の仕事がある．Bird は行方向をＡＣ法で，列方向をＫＭＰ法で検索することによって２次元パターンの探索を実現している[1]．Zhu-Tadaokaは２つの方法を提案している．どちらもテキストをあらかじめパターンの列方向の長さ分の文字列を用いてハッシュ値を計算しておいて，パターンのハッシュ値を１つはＫＭＰ法でもう１つはＢＭ法の変形版を用いて検索する方法である[2]．Baeza-Yates はｍ（パターンの列方向の長さ）行毎にCommentz-Walterの方法で行パターンを検索して，それが見つかった場合にはパターンの検出されうるエリアを素朴な方法で検索する方法を提案している[3]．これらの方法はすべて１つの２次元パターンだけを検索対象としていて，パターンが複数の場合は考慮されていない．Zhu-Tadaokaは列方向もＡＣ法を使うことによって，Birdの方法を複数の２次元パターンの探索に拡張する方法を考案している[4]．この方法はＡＣ法を基にしているため，探索されるデータ空間の大きさに比例した探索時間がかかる．ＦＡＳＴ法は前述したように複数の１次元パターンを効率良く探索する方法である．これを２次元パターンの探索に拡張できれば，もっと効率の良い探索が期待できる．この拡張方法を考案し，実験によってその有効性を確認したので，それについて述べる．

４．２　ＦＡＳＴ法の２次元パターン検索への拡張[5]

ＦＡＳＴ法を２次元に拡張するための基本的な考え方は以下のとおりである．２次元パターンとしては，説明の便宜上，ここでは２次元文字列を例にとる．

今，図４．１のようなパターンを（２次元の）テキストから探索する問題を考える．PT_1，PT_2，PT_3が３つの２次元のパターンである．このパターン探索に２次元ＦＡＳＴ法では２段階の探索を行う．まずパターンを列方向と行方向に分解し

て考える．すなわちパターンの各行を１次元パターンと見なしてパターン（行パターン）それぞれに固有の記号　（行パターン記号：ＲＰ記号）を割り当てる．例えば図４の行パターン“ａｂｃｄｅ”にはX_1，“ｆｇｈｉｊ”にはX_2……とＲＰ記号を付ける．このとき，同じ行パターンには同じＲＰ記号が付くようにする．このＲＰ記号を列方向に並べたものを列パターンと呼ぶことにする．つまり，PT_1の列パターンは（$X_1X_2X_3X_4X_5$）^{t}，PT_2の列パターンは（$X_6X_7X_8X_9X_7X_8$）^{t}，PT_3の列パターンは（$X_{10}X_4X_5$）^{t}となる．探索の第一段階では行パターンを（通常の）ＦＡＳＴ法で探索する．行パターンが見つかれば，列方向にＦＡＳＴ法を用いて列パターンを探索する．列パターンが見つかったときに元の２次元パターンが存在していることがわかる．ＦＡＳＴ法と同様にテキストの照合位置を決めるためにgoto関数を用いるが，この作成法はＦＡＳＴ法と同じなので説明を省略する．た

PT_1:　abcde $=X_1$
　　fghij $=X_2$
　　hikde $=X_3$
　　demlk $=X_4$
　　bcamj $=X_5$

PT_2:　abcd $=X_6$
　　mcag $=X_7$
　　dlfk $=X_8$
　　ghij $=X_9$
　　mcag $=X_7$
　　dlfk $=X_8$

PT_3:　kgcde $=X_{10}$
　　demlk $=X_4$
　　bcamj $=X_5$

図４．１　２次元パターンの一例

だしgoto関数は行パターン，列パターンそれぞれに用意しておく．図４．２に行パターン用のパターン照合機械の一部を示す．表４．１に行パターンの，表４．３に列パターンのgoto関数を示す．表４．２には行パターンの出力関数を示す．

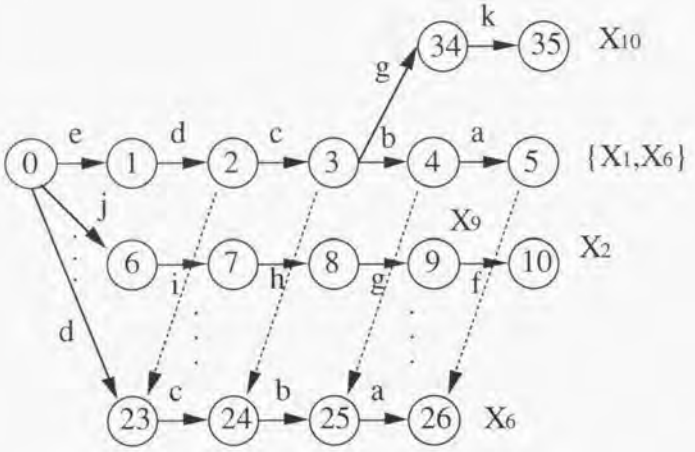


図４．２　２次元ＦＡＳＴ法における
行パターン用照合機械

表４．１　行パターンのgoto関数の例

文字種／状態	0	1	2	3		23	24	25		31	32	33	34	35
a	-1	-5	-5	-6		-4	-5	26		-6	-7	-8	-7	-8
b	-2	-5	-5	4		-4	25	-6		-6	-7	-8	-7	-8
c	-1	-5	3	-6		24	-5	-6		-6	-7	-8	-7	-8
d	23	2	-5	-6		-4	-5	-6		-6	33	-8	-7	-8
e	1	-5	-5	-6		-4	-5	-6		-6	-7	-8	-7	-8
f	-1	-5	-5	-6		-4	-5	-6		-6	-7	-8	-7	-8
g	27	-5	-5	34		-4	-3	-6		-6	-7	-8	-7	-8
h	-2	-5	-5	-6		-4	-5	-6		-6	-7	-8	-7	-8
i	-1	-5	-5	-6		-4	-5	-6		-6	-7	-8	-7	-8
j	6	-5	-5	-6		-4	-5	-6		-6	-7	-8	-7	-8
k	14	-5	11	-6		-2	-5	-6		-6	-7	-8	35	-8
l	-1	-5	-5	-6		-4	-5	-6		32	-7	-8	-7	-8
m	-1	-5	-5	-6		-4	-5	-6		-6	-7	-8	-7	-8
(other)	-4	-5	-5	-6		-4	-5	-6		-6	-7	-8	-7	-8

表４．２　行パターンの出力関数の例

state	rout
5	$\{X_1, X_6\}$
9	X_9
10	X_2
13	X_3
18	X_4
22	X_5
26	X_6
30	X_7
33	X_8
35	X_{10}
(other)	0

表４．３　列パターンのgoto関数の例

文字種／状態	0	1	2	3	4	5	6	7	8	9	10	11	12
X_1	-3	-4	-5	-6	5	-8	-4	-5	-6	-7	-8	-9	-6
X_2	-3	-4	-5	4	-7	-8	-4	-5	-6	-7	-8	-9	-6
X_3	-2	-4	3	-6	-7	-8	-4	-5	-6	-7	-8	-9	-6
X_4	-1	2	-5	-6	-7	-8	-4	-5	-6	-7	-8	-9	-6
X_5	1	-4	-5	-6	-7	-8	-4	-5	-6	-7	-8	-9	-6
X_6	-3	-4	-5	-6	-7	-8	-4	-5	-6	-7	11	-9	-6
X_7	-1	-4	-5	-6	-7	-8	7	-5	-6	10	-8	-9	-6
X_8	6	-4	-5	-6	-7	-8	-4	-5	9	-7	-8	-9	-6
X_9	-2	-4	-5	-6	-7	-8	-4	8	-6	-7	-8	-9	-6
X_{10}	-2	-4	12	-6	-7	-8	-4	-5	-6	-7	-8	-9	-6
(other)	-3	-4	-5	-6	-7	-8	-4	-5	-6	-7	-8	-9	-6

4．2．1　2次元ＦＡＳＴ法のアルゴリズム

上述の考え方を厳密にアルゴリズム（照合部分のみ）で記述すると，図4．3のようになる．ただし変数の宣言や関数の引数や初期化の一部は省略してある．a_{ij}はテキストのｉ行ｊ列目の文字を示している．ｍａｐは行パターンの探索結果を保持しておくための2次元配列で，その容量は最大の列パターンを検出できるように，

テキストの行長（m）×（列パターンの最大長（ｓ）＋1）

となっている．

テキストの文字列xを $x=a_{11}a_{12}\cdots\cdots a_{1n}$ とする．

$$\cdots\cdots a_{ij}\cdots\cdot$$

$$a_{m1}a_{m2}\cdots\cdots a_{mn}$$

すなわちm×nの2次元データである．

行パターンのgoto関数を*rg*，列パターンのgoto関数を*cg*，

出力関数を*cout*とする．行パターンに対する出力関数

（つまり列パターン探索の入力）は*rout*である．

出力はx中に出現するパターンとその位置である．

*length*は行パターンの長さを求める関数である．

mapはmm行（現在の探索対象行に相当）は0で，その他の行は#で初期化されているものとする．

```
 1 begin
 2   qc:=cmin;  /*  cmin は列パターンの最小長  */
 3   while qc ≦ n  do
 4     begin
 5       mm:=cmax;  /*  cmax は列パターンの最大長  */
 6       rst:=0;
 7       dm:=cmin;
 8       qr:=rmin;  /*  rmin は行パターンの最小長  */
 9       while  qr ≦ m do
10         begin
```

```
11          if rg (rst, a_{qc,qr}) < 0 then
12            begin
13              qr:=qr-rg (rst, a_{qc,qr});
14              rst:=0
15              end
16            else
17              begin
18                rst:=rg (rst, a_{qc,qr});
19                if rout(rst) > 0 then
20                  begin
21                    if map(mm,qr) = 0 then
22                        map(mm,qr):=rout(rst);
23                    else if map(mm,qr) > 0 then
24                        map(mm,qr):= #;
                    /* # は"未探索"を示す記号 */
25                    end
26                cst:=cg (0, rout(rst));
27                if cst < 0 then
28                     dm:= min(dm,-cst);
29                else
30                  begin /* 列パターンの探索 */
31                    len:=length (rout(rst));
32                    dm:=csearch (cst,mm,dm,len);
33                  end
34                qr:=qr-1;
35              end
36          end
37        qc:=qc + dm;
38        maprar (map, dm)  /* map の並べ直しと初期化 */
39      end
40  end
```

```
41 function csearch (cst,mm,dm,len)
42   begin
43    mc:=mm;  qs:=qc-1;
44    while mc ≦ cmax do
45      begin
46       if cout(cst) ≠ empty then
47          print qr,qc-clen(cst)+1,cout(cst);
        /*  clen(cst) は cst に対応する列パターンの長さ */
48       mc:=mc-1;
49       if map(mc,qr) = # then
50          begin  /*  map の部分計算 */
51           j:=qr+len-1;
52           tst:=0;
53           while j ≦ (qr+len-1) do
54             begin
55               tst:=rg (tst, a_{qs,j});
56               if tst < 0 then  return(dm);
57               else
58                 begin
59                   if j = qr and rout(tst) > 0 then
60                     begin
61                       if cg (cst,cout(tst)) < 0 then
62                         return(min(dm,mc-cst-cmax));
63                       else break
64                     end
65                   j := j-1;
66                   if j < qr then return(dm)
67                 end
68             end
69          end
```

```
70        else
71          begin   /*  map 上の探索  */
72           if cg (cst,map(mc,qr)) < 0 then
73                 return(min(dm,mc-cst-cmax));
74          end
75        qs := qs - 1;
76       end
77    end
```

図４．３　２次元ＦＡＳＴ法のパターン照合アルゴリズム

２次元ＦＡＳＴ法ではまずテキストのｃｍｉｎ行目（ｃｍｉｎは列パターンの最小の長さで図４．１の場合には３）から行パターン探索（図４．３の８行～３５行）を始める．何故なら，もしｃｍｉｎ行目に２次元パターンの最下段に当たる行パターン（例では，X_3とX_8に相当）が見つからなければ，１～ｃｍｉｎ行の間に２次元パターンが存在するはずがないから，それまでの行を探索対象から外すことによって効率を上げることができるからである．ｃｍｉｎを列パターンの最小の長さに取るのは，最小の列パターンを検出し損なわないようにするためである．したがって，次に探索対象とする行までのスキップ幅（dm）もこれより大きくすることはできない．

行パターンが見つかったときには結果（ＲＰ記号）はｍａｐに書き込まれる（図４．３の１９行～２５行）．このときｍａｐの対応行（mm）は，最大の列パターンも検出するために列パターンの最大長に当たるところにとらなければならない．それがqcの初期値とmmの初期値が（cmax-cmin）分ずれている理由である．１行分の検索が終わると，qcもｍａｐもdm分シフトされる（図４．３の３７行，３８行）．

ｍａｐに検索結果を格納する際，すでにＲＰ記号が書き込まれている場合があるので注意が必要である．これは１つの行パターンが他の行パターンのprefixである場合に起こる．例えば図４．１の“ａｂｃｄｅ”が検出されるときには必ず“ａｂｃｄ”も検出される．この場合にはｍａｐ上に２つのＲＰ記号X_1とX_6を記憶しなければならないが，２次元ＦＡＳＴ法ではこうした場合，ＲＰ記号を“未探索”

に変えることで対処している．もちろん，リスト構造などを用いて２つ以上のＲＰ記号を同一場所に記憶できるようにすることも可能であるが，ｍａｐ上の情報はスパースなので記憶容量の節約のためこの方法の方が望ましいと思われる．確かに，“未探索”に変え情報を消滅させてしまうのは非効率だが，このために生じる余分な探索の手間はごくわずかで，問題になるとは思われない．

もう１つの解決手段は小さいパターン（X_6）が大きいパターン（X_1）と同じ位置で検出される場合，小さいパターンを含む列パターンの替わりに大きいパターンを含むものも列パターンとして登録する方法である．すなわち，PT_2の列パターンとして$(X_6X_7X_8X_9X_7X_8)^1$だけでなく，$(X_1X_7X_8X_9X_7X_8)^1$も考慮するわけである．こうすれば，探索すべき列パターンが多くなるという欠点が生ずるが，いずれにせよX_1が検出されればX_6も検出されるわけであるから効率の低下は，単に見かけ上のことに過ぎない．ｍａｐ上に２つのＲＰ記号を記録しなければならない場合を考慮する必要がなくなるので，この方法の方が実用的かもしれない．

１行の行パターンの探索が終わった後で次にどの行を探索するかは，どの行パターンが検出されたかによって決まる．例えば行パターン探索でX_2とX_9のパターンが検出されたならば，表３のgoto関数から次の行パターン探索は２行先から始めれば良いことがわかる（図４．３の２６行～２８行）．もし表３のgoto関数が正ならば列パターンで最後尾となりうるＲＰ記号が見つかったことになるのでｍａｐを対象に列パターン探索を開始する（図４．３の*csearch*）．すなわち列を１つずつ戻りながら（mc←mc-1），ｍａｐ上の値によって列パターンの探索を進めれば良い（図４．３の７１行～７４行）．しかし，前述のように２次元ＦＡＳＴ法では行をスキップすることが起こるので，全ての行が探索済みであるとは限らない．この場合ｍａｐには“未探索”であることを示す記号が入っているので，この値を求めるために部分的に行パターンの探索を行う．（図４．３の５０行～６９行）これは，主プログラムの行パターンの探索と本質的に同じだが，探索の範囲はlen文字分だけでよい．

また，上記の説明はパターンやテキストが１バイトコードで表現できることを前提としているが，１次元ＦＡＳＴ法と同様な方法（３．５参照）で２バイトコードを扱うことも可能である．

４．２．２　検索実験

テキスト中の文字の平均照合回数で２次元ＦＡＳＴ法の効率Ｐを評価すると，図４．３のアルゴリズムからわかるように最良時には

Ｐ＝１／（ｃｍｉｎ×ｒｍｉｎ）

となる．ここでｃｍｉｎ（ｒｍｉｎ）は列（行）パターンの最小の長さである．

２次元ＦＡＳＴ法の効率はテキストやパターンの性質と大きさによって大きく変化する．テキストもパターンも一様乱数によって生成し，実験した結果を図４．４と図４．５に示す．図４．４はパターンの大きさを３×３に固定し，文字種の数をパラメータとし，パターン数によるprobe rateの変化を示したものである．図４．５は文字種の数を１６に固定し，パターンの大きさをパラメータとし，パターン数によるprobe rateの変化を示したものである．ただし，probe rateを単に文字照合回数で計るのは不適当であると考えたので，

（文字照合回数＋ｍａｐの照合回数）／テキスト文字数

によって計算している．

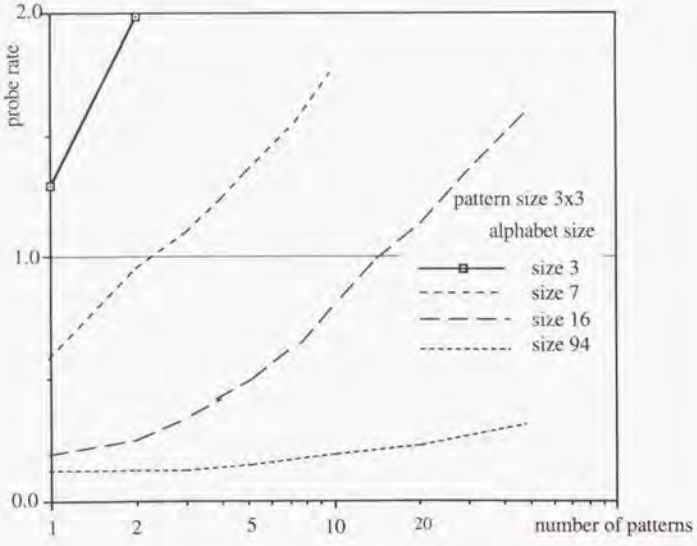


図４．４　２次元ＦＡＳＴ法の探索効率

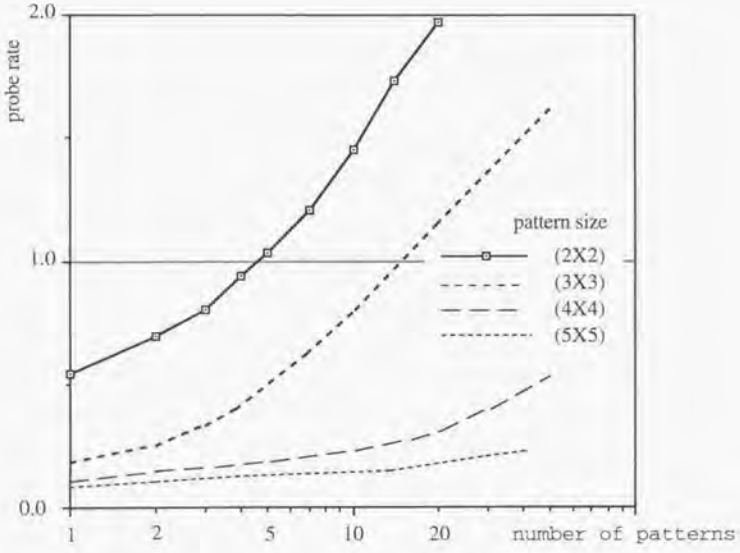


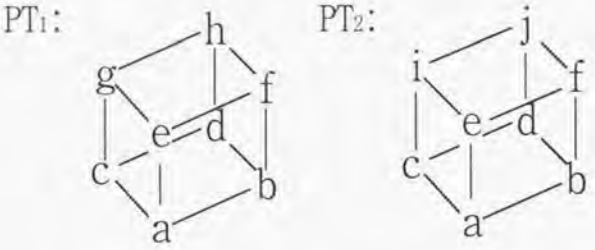
図４．５　文字種１６の場合の効率

アルゴリズムから分かるようにテキストの行方向の長さよって列方向のスキップ幅に影響を与えるので効率にも変化する．この実験ではテキストの行方向の幅は512文字で行なっている．

これを見ると当然予想されることながら文字種が多いほど，あるいはパターンが大きいほど２次元ＦＡＳＴ法の効率が良いことがわかる．文字種が少なく（例えば１６），パターンが小さい（例えば３×３）場合でもパターン数が少なければ（１５以下），Zhu-Tadaokaの方法（常にprobe rateが１）よりも効率が良いことが分かる．

4．3　ＦＡＳＴ法のｎ次元パターン検索への拡張

上記のアルゴリズムは簡単にｎ次元パターンの探索に拡張することができる．パターンを$PT_j[P_{1j}][P_{2j}]\cdots[P_{nj}]$，テキストを$T[T_1][T_2]\cdots[T_n]$とする．パターンの第１次元目から第ｎ－１次元目までの配列の要素を固定して考えれば，（重複を含めて）$P_{1j}\times P_{2j}\times\cdots\times P_{(n-1)j}$個（×パターン数）の１次元パターンが得られる．これが，３章で述べた行パターンに相当する．図４．６に３次元パターンの一例を示す．図４．６では２×２×２（このうち３つは重複するので５つ）の１次元パターンが得られる．これにユニークな記号（ＲＰ記号）をふり（図４．６の例ではX_1からX_5に相当），次にパターンの第１次元目から第ｎ－２次元目までの配列の要素を固定して考えれば，$P_{1j}\times P_{2j}\times\cdots\times P_{(n-2)j}$個のＲＰ記号を基にした１次元パターンが得られる．（図４．６ではX_1からX_5で記述されるY_1，Y_2，Y_3に相当）



ab $=X_1$
cd $=X_2$
ef $=X_3$
gh $=X_4$
ij $=X_5$

PT_1: $X_1X_2=Y_1$
$X_3X_4=Y_2$

PT_2: $X_1X_2=Y_1$
$X_3X_5=Y_3$

図４．６　３次元パターンの一例

同様にして，１次元ずつ次元が減じたパターンが得られる．これらの１次元パターンに対してＦＡＳＴ法によってgoto関数と出力関数を求める．前節から推測できるように最初の１次元目の出力関数は２次元パターン探索の入力記号となる．順次，(n-i-1)次元パターンの出力関数は(n-i)次元パターン探索の入力記号となる．途中の探索結果を取っておくために，それぞれ，

$T_1 \times T_2 \times \cdots \times T_{(n-1)} \times \max_j(PT_{nj}+1)$,

$T_1 \times T_2 \times \cdots \times T_{(n-2)} \times \max_j(PT_{(n-1)j}+1)$,

・・・，$T_1 \times \max_j(PT_{2j}+1)$，の（ｎ－１）の配列を用意しておけばよい．(n-i)次元目のパターンの探索は$T_1 \times T_2 \times \cdots \times T_{(n-i-1)} \times \max_j(PT_{(n-i)j}+1)$の(n-i)次元配列上で行なえばよいことが分かる．

４．４　あとがき

２次元パターン探索を１次元探索の２つ（行探索と列探索）に分解することにより，ＦＡＳＴ法を２次元に拡張するアルゴリズムを考案した．実験によって複数の２次元パターンを同時に高速に探索できることを確認した．この方法はＦＡＳＴ法と同様に文字種が多いほど，パターンが大きいほど効率が良く，パターン数が多くなるにつれて効率が低下することが分かった．

この方法を一般のｎ次元パターンの探索に拡張する方法についても述べた．これらの手法は単に文字列の検索にとどまらず，様々な多次元パターンの検索にも利用が可能である．

第５章　結論

本論文では，文字列処理やパターン検索の基本である複数文字列の効率的な探索アルゴリズムに関する研究について述べた．情報化社会の進展に伴って，情報検索の重要性がますます増大している．筆者は情報検索のもっとも基礎であるパターン検索を効率化することを目指して検討を行なった．

データベースは必要な情報を，必要な時に利用できるようにするための強力なツールである．データの検索の手掛かりとしては，現在キーワードが最も利用されている．この場合，ユーザに自由にキーワードを指定できるようにするためには全文を対象としてキーワードの出現の有無を調べる手段を提供しなければならない．また，社会には電子化されていてもデータベース化されていない文書データもまだまだ多量に存在している．これを対象に何らかの語句が含まれているか否かを調べるには全文探索よりほかに手段はない．

本論文の主要な目的は，任意の複数の文字列（パターン）を電子化されているデータから効率よく検索する手法を提供することである．

第１章では，研究の背景と目的，特にキーワード検索手法について概観し，インデックス検索手法，長尾・森の方法および全文検索手法の違いと技術要件を明らかにした．合わせて，本論文の意義および位置付けについても明らかにした．

第２章では，全文検索手法のアルゴリズムについて検討した．従来のアルゴリズムのうち，素朴な方法，Knuth-Morris-Prattの方法（ＫＭＰ法），Boyer-Mooreの方法（ＢＭ法），Aho-Corasickの方法（ＡＣ法）について簡単に説明を行なった．そして，ＡＣ法の状態遷移という考え方とＢＭ法の「パターンの右端からのテキストの逆向きの照合」というアイデアを結合することによって，複数パターンを効率良く検索できることを示した．提案した方法（ＦＡＳＴ法）の考え方とアルゴリズムについて詳述した．他の複数パターン検索法よりも効率が良いことを明らかにし，パターンが１つだけの場合でもＢＭ法よりも効率が良いことを示した．

第３章では，第２章で提案したＦＡＳＴ法の効率について数学的に検討した．そして，文字種（q）がパターンの長さ（m），パターン数（k）に対して十分大きい

($q \gg mk$) ときには probe rate が $1/m+(m+1)k/2mq$ で近似できることを示した．つづいて，アルゴリズムの正当性を厳密に証明し，このアルゴリズムが信頼に足ることを示した．実験を行ない，効率の推定式がこれとよく一致することを示し，さらに多パターン時には効率が単調に減少しないという現象が起きることについて説明した．ＦＡＳＴ法の欠点は他のアルゴリズムよりメモリーを余計に必要とすることである．文字種とパターンの長さとパターン数の積が小さい場合は問題にはならないが，ＪＩＳ漢字コードなどの２バイトコードを対象にする場合には利用が困難になる場合も生じる．こういう場合でも，ＦＡＳＴ法を利用できるようにするために“文字符号分割による実現”などのメモリ節減手段を示した．

第４章では，２次元パターン探索の従来のアルゴリズムについて概観した後，パターンを列方向と行方向に分解し，行パターンと列パターンという概念を導入することで，ＦＡＳＴ法を２次元パターン探索に拡張する方法について述べた．このアルゴリズムについて詳述し，実験により従来の手法より効率が良いことを示した．さらに同様な発想でＦＡＳＴ法を一般のｎ次元パターンの検索に拡張する方法についても述べた．

情報化社会の進展はデータ情報の指数関数的に増大し，個人に供給される１日分の情報さえ，２４時間を費やしてももはやその全てにアクセスすることができなくなっていると言われている．これに対処するため使い勝手の良い高効率な情報検索手法が強く求められている．文字列の検索は，指定した文字列（パターン）が文書中のどこに現われているかを調べるという単純な作業であるが，電子化された文書の処理にとっては全体の効率に大きく関与する基本操作なので，現在でも重要な研究テーマである．本研究では，有限オートマトンによるパターンの記述と「右端からの照合」というアイデアを結合した新しい全文検索手法について検討し，実験も行ない，その有効性，実現性について検証した．

今後，本研究の概念を生かしながら，様々な情報検索の実際の応用の上で本アルゴリズムの利用を図っていきたいと考えている．また，２次元ＦＡＳＴ法（あるいはｎ次元ＦＡＳＴ法）については，今後はさらに理論的および実験的に考察を進めて，性質を明らかにするとともに，実用面でも利用を図っていきたいと考えている．

謝　辞

本論文の作成にあたり，終始御指導，御鞭撻を賜りました東京大学工学部電子情報工学科羽鳥光俊教授に深謝いたします．

東京大学工学部電子工学科齋藤忠男教授，電気工学科田中英彦教授，生産技術研究所今井秀樹教授，電子情報工学科相田仁助教授，相澤清晴助教授には，適切な御指導と御助言を賜りました．深謝します．

本研究はＮＨＫ放送技術研究所において行ったものである．本研究を開始するきっかけを与えていただいた江原主任研究員に感謝いたします．本研究を進めるにあたり御指導いただいた沓沢淳之助元画像研究部部長（現，シャープ），相沢輝昭前研究主幹（現，広島市立大学），二宮佑一研究主幹に感謝いたします．

有益な御助言をいただいた九州大学理学部有川節夫教授，ＮＨＫ放送技術研究所の榎並和雅先端制作技術研究部部長，藤原正雄元副部長，井上誠喜氏（現，ＡＴＲ），加藤直人研究員（現，ＡＴＲ）に感謝いたします．

第３章で述べた正当性の証明を研究してくださった九州大学工学部竹田正幸助教授（当時大学院生）に感謝いたします．

３年間のＡＴＲ出向の間，研究でお世話になったＡＴＲ自動翻訳電話研究所樽松明前社長（現，電気通信大学），ＡＴＲ音声翻訳通信研究所山崎泰弘社長，森元逞第４研究室長，谷戸文廣主幹研究員（現，ＫＤＤ），田代敏久研究員（現，マイクロソフト）を初めとする自動翻訳電話研究所に在籍された方々に感謝いたします．また，日頃御討論いただく，畑田のぶ子研究員，金澗培研究員，田中英輝研究員，熊野正研究員，柴田正啓研究員を初めとする画像研究部各位に感謝いたします．

本研究の機会を与えていただいたＮＨＫ放送技術研究所杉本昌穂元所長（現，パイオニア），泉武博前所長，西澤台次所長，中林克己前次長，山田宰次長，大島英男次長に深謝いたします．

参考文献

第１章

（１）J. Aoe："Computer Algorithm -Key Search Strategies-", IEEE Computer Society Press (1991)

（２）有川節夫，篠原武：”文字列パターン照合アルゴリズム”，コンピュータソフトウェア，Vol.4, No.2, pp.2-23 (1983)

（３）Boyer,R.S. and Moore,J.S.："A Fast Searching Algorithm", Comm.ACM, Vol.20, pp.762-772 (1977)

（４）Aho,A.V. and Corasick,M.J.："Efficient String Matching: An Aid to Bibliographic Search", Comm.ACM, Vol.18, No.6, pp.333-340 (1975)

（５）Kowalski,G.,Meltzer,A.："New Multi-Term High Speed Text Search Algorithms", First International Conference on Computer and Applications (CAT.No.84ch2039-6), pp.514-522, XIV+905(1984)

（６）Commentz-Walter,B.："A String Matching Algorithm Fast on the Average", Lecture Notes in Computer Science, Vol.71, Springer, pp.118-132 (1979)

（７）吉村賢治，日高達，吉田将：”日本語科学技術文における専門用語の自動抽出システム”，情報処理学会論文誌，Vol.27, No.1, pp.33-40 (1986)

（８）木本晴夫：”日本語新聞記事からのキーワード自動抽出と重要度評価”，電子情報通信学会論文誌D-I，Vol.J74-D-I, No.9, pp.556-566 (1991)

（９）Nagao M. and Mori S.："A New Method of N-gram Statistics for Large Number of n and Automatic Extraction of Words and Phrases from Large Text Data of Japanese", COLING94, pp.611-615(1992)

（１０）Roberts C.S.："Patial-Match Retrieval via the Method of Superimposed Codes", Proceedings of the IEEE, Vol.67, No.12, (1979)

（１１）有田　健，津田和彦，入口浩一，青江順一：“複数キーワードによる検索の一高速化手法”，情報処理学会研究会資料，NL104-7 (1994)

（１２）Burkowski F. J.："A Hardware Hashing Scheme in the Design of a Multiterm String Comparator", IEEE Trans. Comput, COM-31, 9, pp.825-834 (1982)

（１３）井上　潮，速水治夫，福岡秀樹，鈴木健司，松永俊雄：
“データベースプロセッサＲＩＮＤＡの設計と実現”，
情報処理学会論文誌，Vol.31, No.3, pp.373-380 (1990)

第２章

（１）Knuth D. E., Morris J. H. and Pratt V. R.："Fast Pattern Matching in Strings",
SIAMJ. Comput, 6, 2, pp.322-350　(1977)
（２）Boyer R.S. and Moore J.S.："A Fast Searching Algorithm",
Comm.ACM, Vol.20, pp.762-772 (1977)
（３）Galil Z："On Improving the Worst Case Running Time of Boyer-Moore String
Searching Algorithm", Comm.ACM, Vol.22, No.9, pp.505-508 (1979)
（４）Aho A.V. and Corasick M.J.："Efficient String Matching: An Aid to Bibliographic
Search", Comm.ACM, Vol.18, No.6, pp.333-340 (1975)
（５）Horspool R.N.："Practical Fast Seaching in Strings", Software-Practice and
Experience, Vol.10, No.6, pp.501-506 (1980)
（６）Kowalski G. and Meltzer A.："New Multi-Term High Speed Text Search Algorithms",
First International Conference on Computer and Applications (CAT.No.84ch2039-6),
pp.514-522, XIV+905(1984)
（７）Commentz-Walter B.："A String Matching Algorithm Fast on the Average",
Lecture Notes in Computer Science, Vol.71, Springer, pp.118-132 (1979)
（８）竹田正幸：” 複数文字列照合に関する浦谷アルゴリズムの正当性について”，
夏のＬＡシンポジウム予稿，pp.1-26(1988)
（９）浦谷則好：” 高速な複数文字列照合アルゴリズム：ＦＡＳＴ”，
情報処理学会論文誌，Vol.30, No.9, pp.1119-1125 (1989)

第３章

（１）浦谷則好：” ＦＡＳＴ法の効率の推定と長パターン時のふるまい”，
情報処理学会論文誌，Vol.32, No.9, pp.1073-1079 (1991)

（2）Uratani N. and Takeda M.："A Fast String-Searching Algorithm for Multiple Patterns", Info. Proc. & Manag., Vol.29, No.6, pp.775-791 (1993)
（3）Takeda M.："A Proof of the Correctness of Uratani's String Searching Algorithm", Tech. Report RIFIS-TR-CS-7, Kyushu University (1988)
（4）浦谷則好：" 高速な複数文字列照合アルゴリズム"，
ＮＨＫ技研Ｒ＆Ｄ，No.9, pp.56-68 (1990)
（5）Arikawa S.："One-Way Sequential Search Systems and Their Powers", Bull. Math. Stat., Vol.19, No.3-4, pp.69-85 (1981)
（6）Arikawa S. and Shinohara T.："A Run-Time Efficient Realization of Aho-Corasick Pattern Matching", New Generation Comput., Vol.2, No.2, pp.171-186 (1984)
（7）尹志熙，高木利久，牛島和夫：" ５種類のパターン・マッチング手法をＣ言語の関数で実現する－第２回日本語テキストの場合"，
日経バイト１９８７年９月号，pp.233-243 (1987)
（8）加藤直人：" 目標言語のフルテキスト検索による機械翻訳の訳語選択"，
信学技報，NLC93-32, pp.9-14 (1993)

第４章

（1）Bird R.S.："Two Dimensional Pattern Matching", Inf. Process. Letters, Vol.6, No.5, pp.168-170 (1977)
（2）Zhu R.F. and Takaoka T.："A technique for two-dimensional pattern matching", Communications of ACM, Vol.32, No.9, pp.1110-1120 (1989)
（3）Baeza-Yates R. and Regnier M.："Fast Algorithms for Two Dimensional and Multiple Pattern Matching", Lect. Notes Comput. Sci., Vol.447, pp.332-347 (1990)
（4）Zhu R.F. and Takaoka T.："The Extension of the Aho-Corasick Algorithm to Multiple Rectangular Pattern Arrays of Different Sizes and N-Dimensional Cases", J. Information Proccessing, Vol.11, No.4, pp.271-277 (1988)
（5）浦谷則好：" ２次元パターン高速探索アルゴリズム"，
第３７回情報処理学会全国大会論文集，pp.1541-1542 (1988)

本論文関連の対外発表

学会誌論文

（1）浦谷：“高速な複数文字列照合アルゴリズム：ＦＡＳＴ”
情報処理学会論文誌，Vol.30,No.9,pp.1119-1125(1989)

（2）浦谷：“ＦＡＳＴ法の効率の推定と長パターン時のふるまい”
情報処理学会論文誌，Vol.32,No.9,pp.1073-1079(1991)

（3）N. Uratani and M. Takeda:"A FAST STRING-SEARCHING ALGORITHM FOR MULTIPLE PATTERNS"
Information Proc. & Manag., Vol.29,No.6,pp.775-791(1993)

（4）浦谷：“複数２次元パターンの高速探索アルゴリズム”
電子情報通信学会に投稿中

研究会・国内発表

（1）浦谷：“複数文字列照合アルゴリズム”
情処全大，pp.57-58(1987)

（2）浦谷：“複数文字列照合アルゴリズム”
信学研報，SS87-27,pp.1-7(1988)

（3）浦谷：“２次元パターン高速探索アルゴリズム”
情処全大，pp.1541-1542(1988)

その他の対外発表

（1）浦谷：“高速な複数文字列照合アルゴリズム”
ＮＨＫ技研Ｒ＆Ｄ，No.9,1990-5月号,pp.56-68(1990)

出願特許

（1）複数パターンの検索方法　(1987.9)

（2）複数２次元パターンの検索方法　(1988.9)

本論文関連以外の主な対外発表

学会誌論文

（1）浦谷，柴田，野口，相沢："静止画検索システムＦＯＲＫＳの試作"
情報処理学会論文誌，Vol.28,No.7,pp.758-767(1987)

研究会等

（1）浦谷："ランドサットＭＳＳデータによる土地利用区分メッシュデータの細分"，第７回リモートセンシングシンポジウム，pp.81-84(1981)
（2）浦谷："Ｐｒｏｌｏｇ処理系の試作と日本語処理への応用"
ＮＨＫ技研月報，Vol.27,No.10,pp.424-429(1984)
（3）浦谷，柴田，江原："日本語ワードプロセッサーにおける放送用辞書の利用"
ＮＨＫ技研月報，Vol.30,No.9,pp.301-306(1987)
（4）柴田，浦谷，井上，野口："放送用静止画検索システムＦＯＲＫＳ"
ＴＶ学技報，Vol.11,No.31,pp.57-62,OPT'87-18(1987)
（5）浪岡，浦谷，相沢："ニュース文における深層格抽出法"
情処研資，NL77-3,pp.1-8(1990)
（6）浦谷，相沢："英語ニュースの機械翻訳"
情処研資，NL78-18,pp.137-143(1990)
（7）相沢，浦谷，田中："英語ニュースへの機械翻訳システムの適用"
信学研報，NLC91-20,pp.31-38(1991)
（8）浦谷，相沢，田中，加藤，畑田，住吉："英語ニュースの機械翻訳"
ＮＨＫ技研Ｒ＆Ｄ，No.14,1991-8月号pp.1-12(1991)
（9）相沢，鎌田，浦谷："外電経済ニュースの英日機械翻訳：新しいアプローチ"
信学研報，NLC91-45,pp.65-72(1991)
（１０）浦谷，鈴木，永田，森元，田窪，定延，成田：
"目的指向型会話文解析システムの機能評価法"
信学研報，NLC92-10,pp.33-40(1991)

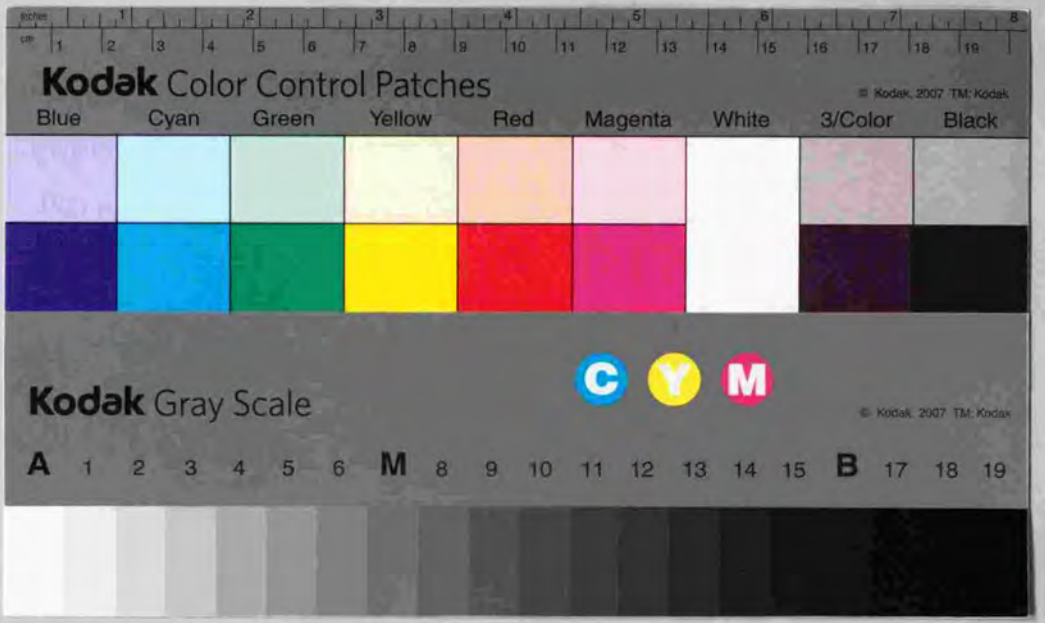
（１１）保坂，竹沢，浦谷：“対話データベースを使った無助詞名詞句の分析”
人工知能学会研究会，SIG-SLUD-9203-1,pp.1-7(1992)
（１２）浦谷，森元，谷戸：“音声翻訳システムＡＳＵＲＡの開発”
情処研資，CH18-7,pp.49-54(1993)
（１３）谷戸，竹沢，嵯峨山，鷹見，Singer，浦谷，森元，樽松：
“自動翻訳電話国際共同実験”，信学研報，SP93-23,pp.73-80(1993)
（１４）浦谷：“ＡＴＲ会話音声データにおける言語情報表記”
情処研資，93-SLP-2,pp.1-4(1993)

国際会議

（１）T. Aizawa, T. Ehara, N. Uratani, H. Tanaka, N. Kato, S. Nakase, N. Aruga and
T. Matsuda:"A Machine Translation System for Foreign News in Satellite Broadcasting"
COLING90, pp.308-310(1990)
（２）N. Katoh, N. Uratani and T. Aizawa:
"Processing Proper Nouns in Machine Translation for English News"
International Conf. on CICL, pp.431-439(1991)
（３）J. Hosaka, T. Takezawa and N. Uratani:
"Analyzing Postposition Drops in Spoken Japanese"
ICSLP'92, pp.1251-1254(1992)
（４）T. Tashiro, N. Uratani and T. Morimoto:
"Restructing Tagged Corpora with Morpheme Adjustment Rules"
COLONG94, pp.569-573(1994)
（５）O. Furuse,Y. Sobashima,T. Takezawa and N. Uratani:
"Bilingual Corpus for Speech Translation"
AAAI-94 Workshop,(Integration of NL and SP), pp84-90, (1994)
（６）T. Morimoto, N. Uratani,T. Takezawa,O. Furuse, Y. Sobashima, H. Iida,
A.Nakamura,Y. Sagisaka,N. Higuchi and Y. Yamazaki:
"a Speech and Language DataBase for Speech Translation Research"
ICSLP94, pp.1791-1794(1994)

その他

（1）匂坂，浦谷：“ＡＴＲ音声・言語データベース”
日本音響学会誌，Vol.48,No.12,pp.878-882(1992)
（2）浦谷：“自動翻訳電話の現状”
情報通信ジャーナル，１９９３年３月号，pp.42-45(1993)
（3）浦谷，江原：“音声処理と翻訳”
テレビジョン学会誌，Vol.47,No.12,pp.1588-1591(1993)
（4）“自動翻訳電話”，共著（ＡＴＲ国際電気通信基礎技術研究所編）
オーム社(1994)


Kodak Color Control Patches
© Kodak, 2007 TM: Kodak
Blue
Cyan
Green
Yellow
Red
Magenta
White
3/Color
Black
Kodak Gray Scale
C
Y
M
© Kodak, 2007 TM: Kodak
A 1 2 3 4 5 6 M 8 9 10 11 12 13 14 15 B 17 18 19