高速な複数文字列照合アルゴリズム：FAST†

浦谷 則好††

コンピュータによる文書処理にとって文字列の照合は最も基本的な操作である．文書処理の高速化への寄与が大きいので，効率の良い照合手法が求められている．パターンが1つの場合には Boyer-Moore 法が最も効率の良い手法として知られているが，この方法では複数パターンを同時に照合することはできない．複数パターンを同時に照合する方法としては Aho-Corasick 法が有名であるが，効率は Boyer-Moore 法より劣る．筆者らは「パターンの後方からの照合」という Boyer-Moore 法の基本的なアイデアと，「パターン照合機械による照合」という Aho-Corasick 法の基本的なアイデアを結合して，FAST 法 (A Flying Algorithm for Searching Terms) を考案した．FAST 法では複数のパターンを同時に効率良く照合することができる．この論文では FAST 法の基本的な発想と具体的なアルゴリズムについて述べる．FAST 法の効率についても考察し，実験による結果も示す．文字種が多いときやパターンが長いときには高い照合効率が得られることを確認した．例えば，パターン長が2と短くても文字種が94のときには，パターン数60以下では Aho-Corasick 法より効率がまさっている．また，この方法は多大なメモリを必要とする欠点を有しているが，この軽減方法についてもふれた．

1. まえがき

コンピュータ技術の発達によって文字情報の電子化が進んでいる．電子化された文書の処理にとって最も基本的な操作は文字列の照合である．文字列の照合は問題としては単純であるが，文書データベースにおけるキーワードの検索やプログラミング言語のコンパイラやインタプリタにおける予約語の処理などさまざまな場面で用いられている．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[1]．

1個の文字列（以下パターンと呼ぶ）を探す手法としては Boyer-Moore の方法 (BM 法)[2] が最も効率の良い方法として知られている．この方法の probe rate(テキスト1文字当たりの照合回数) 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[4]．また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 probe rate は4以下になることが証明されている[5]．しかし，この方法で複数パターンの照合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である．

複数のパターンを同時に効率良く探索する方法としては，Aho-Corasick の方法 (AC 法)[3] が有名である．これは与えられたパターンの集合からパターン照合機械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1回走査させて各パターンの出現位置を検出するアルゴリズムであり，probe rate は常に1である．

筆者らは 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考案した．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[6] や Commentz-Walter による方法[7] があるがどちらも我々の方法より probe rate が劣っている．

以下，第2章では考案したアルゴリズム (FAST 法) の基本的な考え方を示し，第3章ではアルゴリズムの詳細について述べ，効率を考察し，実験結果を示し，メモリ節減手段についてもふれる．

2. BM 法と AC 法の結合

BM 法の基本的な考え方は，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，という点にある．例えば図1の場合，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

† FAST: A Fast Algorithm for Matching Multiple Patterns by NORIYOSHI URATANI (Science and Technical Research Laboratories, NHK (Japan Broadcasting Corporation)).

†† 日本放送協会放送技術研究所

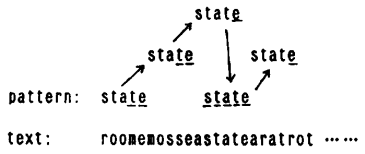

図 1 BM 法による照合手順
Fig. 1 Matching by Boyer-Moore method.

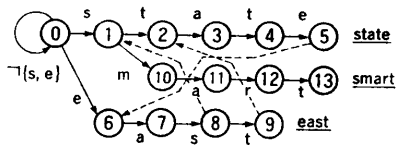

図 2 AC 法におけるパターン照合機械
Fig. 2 The pattern matching machine of Aho-Corasick method.

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．

一方，AC 法はテキストの文字を入力とし，照合の途中段階を状態として記憶する一種のオートマトン（パターン照合機械と呼ぶ：以下 pmm と略記）を作成することによって，複数文字列の同時照合を可能にしている．例えば，パターンが "state", "east", "smart" の場合には pmm は**図 2** のようになる．図 2 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 では状態 5，9 以外から状態 0 への破線は省略してある．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．

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 を得られることが期待できる．

パターンの集合は図 2 の場合と同じで，テキストは図 1 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そろえて**図 3** 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 3 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

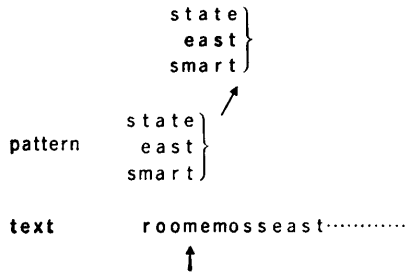

図 3 複数パターンの右端からの照合
Fig. 3 Matching from the rightmost of patterns.

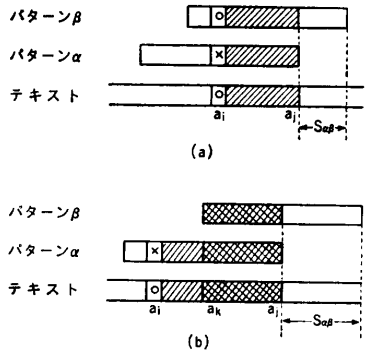

図 4 パターン群シフト量の計算
Fig. 4 Calculation for shift of patterns.

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．

これには**図 4** 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 $a_k\ a_{k+1} \cdots a_j$ $(i<k\leqq j)$ 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 4 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．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）に対する $S_{\alpha\beta}$ を求め，その最小値 S_α つまり，

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

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とき，テキスト上の文字 (a_i) に対するシフト量を delta 1 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 delta 2 という関数で別々に取り扱っているが，FAST 法では $a_i\ a_{i+1} \cdots a_j$ 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ある．

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のかの情報は 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第3章で述べるが，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．

この論文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 delta 関数を利用するというものである．

3. FAST 法

第2章で述べた考え方に基づいて，複数文字列を同時に効率良く探索す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 法の具体的なアルゴリズムを述べ，効率について考察し，実験結果を示し，メモリ節減手段についてもふれる．

3.1 FAST 法のアルゴリズム

図5 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6** 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．

図6には図5のアルゴリズムの結果を利用して

```
パターンの集合を {y1, y2, ・・・yn} 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6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十分大きい正整数 */
   for i:=1 to k do enter( yi, i)
end

procedure enter(a1a2・・・・am, k)
begin
   state:=0; j:=m;
   while ( g(state, aj) ≠ fail ) and ( j > 0 ) do
      begin
         state:=g(state, aj);
         j:=j-1;
      end
   while j>0 do
      begin
         g(state, aj):=nst;
         depth(nst):=m-j+1;
         state:=nst;
         nst:=nst+1;
         j:=j-1
      end
   emap(k):=state;
   out(state):={a1a2・・・・am};
   if minlen > m then minlen:=m
end
```

図5 goto 関数を求めるアルゴリズム
Fig. 5 The algorithm for constructing goto function.

failure 関数を求めるアルゴリズムを示す．図6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 (ϕ (st)) は BM 法のアルゴリズムでシフト量 (delta 2) を求める際に用いる補助関数に相当している．

ϕ を利用して FAST 法の failure 関数 f を計算するが，図6のステートメントAで f を更新しているのは図4の(a)の場合に相当している．つまり，パターンα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4の(b)に相当するシフト量で，その状態(j)までの文字列の長さ (depth (j)) と最小のパターン長 (minlen) の和で初期化される．関数 ddm の更新ループは，図4の(b)のパターンαの照合に失敗した場合，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（の最小値）までのポインタのシフト量を求めるためのものである．関数 check はあるパターンの影響で別の（あるいは同一の）パターンに属する状態（節点）のシフト量が変化する場合，このアルゴリズムでは最小となるものから順に計

```
goto関数 g と minlen, depth, enap は図5のアルゴリズムによ
って求められているとする.
f(s,a) の初期値は large, check(s)の初期値は0とする.
出力は failure 関数 f 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
              if ( f(fst,c) > depth(r) ) then
                   f(fst,c):=depth(r);   <----A
              fst:=φ(fst);
            end;
          if fst >= 0 then φ(s):=g(fst, c);
                      else φ(s):= 0
        end
    end;
  for j:=0 to nst-1 do
    ddm(j):=depth(j)+minlen;

    for j:=1 to k do
      begin
        jst:=enap(j);
        bst:=φ(jst);
        jlen:=depth(jst);
        while bst≠0 do
          begin
            slen:=jlen-depth(bst);
            queue:={bst};
            while queue≠empty do
              begin
                let queue:=r.tail;
                queue:=tail;
                if check(r)≠j then
                  begin
                    if ddm(r) > slen+depth(r) then
                       begin
                          ddm(r):=slen+depth(r);
                          for each c such that
                                g(r,c)=s≠fail do
                             queue:=queue.s;  <---B
                       end;
                    check(r):=j
                  end
              end
            bst:=φ(bst)
          end
      end
  for j:=0 to nst-1 do
     for each c such that g(j,c)=fail do
        if f(j,c) > ddm(j) then f(j,c):=ddm(j)
end
```

図 6 failure 関数を求めるアルゴリズム
Fig. 6 The algorithm for constructing failure function.

算するので同一節点に対する計算を2回以上行う無駄を省くために導入されたものである．また，図6のステートメントBで $g(r,c)=s\neq$fail となる s を queue に入れているのは親節点のシフト量が変わればその子節点も影響を受けるからである．アルゴリズムの最後で $g(j,c)=$fail となる $f(j,c)$ に対して，$\min(f(j,c), \mathrm{ddm}(j))$ をセットして，failure 関数 f が完成する．

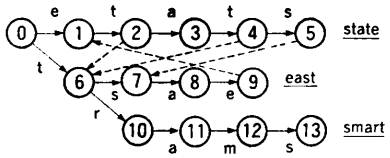

図 7 FAST 法におけるパターン照合機械
Fig. 7 The pattern matching machine of FAST.

図7に，図5，図6のアルゴリズムで求められるパターンが図2と同じ場合の FAST 法の pmm を示す．実線，破線，下線付き文字列の意味は図2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．

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．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．このようにして求めた goto 関数 g を**表1**に示す．これを用いるとパターン照合の方法は**図8**のアルゴリズムになる．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．

図9は図1と同じテキストを用いて FAST 法の照合動作の例を示す．図9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9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9のテキスト "…seast…" の 'a' は3回照合される．

FAST 法のアルゴリズムの正当性（健全性，完全性，停止性）の厳密な証明はすでに竹田によってなさ

表 1 goto 関数の例
Table 1 An example of goto function.

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

```
テキストの文字列xを　x = a1 a2 ・・・ an　とする.
goto関数をg, failure 関数をf, 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:=unc;
    /* unc は決して照合に成功しない文字 */
  state:=0;
  q:=minlen;
    /* minlen はパターンの最小文字長 */
  while q <= n do
    begin
      if g(state, aq ) < 0 then
           begin
              q:=q-g(state, aq);
              state:=0
           end
      else begin
              state:=g(state, aq);
              if out(state)≠empty then
                    begin
                       print q;
                       print out(state)
                    end
              q:=q-1;
           end
    end
end
```

図 8 文字列照合アルゴリズム
Fig. 8 The algorithm for matching patterns.

```
state:                  5 4 3 2 1 0
order:                  1413121110 9
state:                9 8 7 6 0
order                   8 7 6 5 4
state:      0     0       0
order:      1     2       3
text:  r o o m e m o s s e a s t a t e a…
```

図 9 FAST 法による照合手順
Fig. 9 Matching sequence by FAST.

れている[8]ので，ここでは簡単にふれる．まず，停止性は図6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5のアルゴリズムに基づく状態遷移は図7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．次に完全性であるが，2章で述べたように，検出漏れがないようにパターン群を動かすには図4の(a)と(b)の場合を考慮すればよい．図6のアルゴリズムは図4の(a)と(b)の場合に相当するシフト量の最小値を $f(j,c)$ にセットする（この厳密な証明は文献8)を参照）ので，完全性も満足することがわかる．

3.2 FAST 法の効率

probe rate で FAST 法の効率 P を求めると，図8のアルゴリズムからわかるように，最良時には

$P=1/\text{minlen}$ (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よい．この場合でもパターンが1つ見つかり (maxlen 回の照合)，その直後で失敗するごとに最低1つは照合開位置が右にずれるので，P は

$P=\text{maxlen}+1$ (maxlen：パターンの最大長)

となる．すなわち最悪時でも P 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，P の期待値を求めると以下のようになる[10]．ただし，単純化のため k 個のパターンの長さはすべて m と仮定している．

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

ここで，t_j は状態0から始まるパターン照合が j 回成功する確率であり，$P_{s,i}$ は 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 だけ右にずれたところから照合が再開される確率である．

文字種 q がパターン長 m，パターン数 k より十分大きいときには，P は近似的に

$$P=\frac{1}{m}+\frac{(m+1)k}{2mq}$$

となる．$P<1$ となるので AC 法 $(P=1)$ より効率が良いことがわかる．

推定式(*)を基に $P=1$ となる m と k の関係を q をパラメータとして求めると図 10 のようになる．これを見ると $q=2$ の場合には m が相当大きくならないと

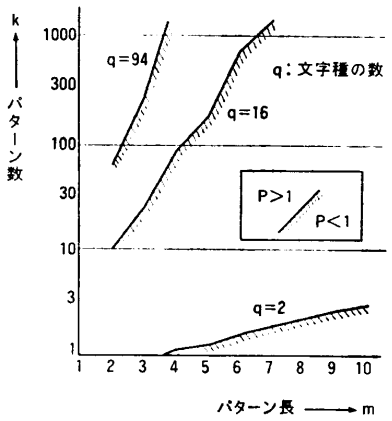

図 10 効率が1となるパターン長(m)とパターン数(k)の関係
Fig. 10 The relation between length of patterns (m) and number of patterns (k) of which probe rate is 1 for FAST.

FAST 法の効率は良くならないことがわかる．反対に q が大きいときには m が小さいときにもかなりの効率が得られることがわかる．例えば $q=94$ で $m=2$ のときにはパターン数 k が60以下で AC 法より効率が良いことになる．

3.3 実験結果

3.2 節で述べたように FAST 法の効率は q, m, k の組み合わせによって大きく変化する．そこで FAST 法による文字列照合の実験を行った．実験に使用したテキストとパターンは一様乱数により生成した．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 11** に示す*．図中には推定式から得られ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，図中の点線より下では AC 法より効率が良いことになる．$q=16$, $m=4$ の場合に BM 法を複数回適用したときの期待値も一点鎖線で示す．当然のことであるが，BM 法を複数回使用するのに比して著しい効率改善が見られる．また $q=2$ の場合を除いて実験結果と推定値は良く一致している．

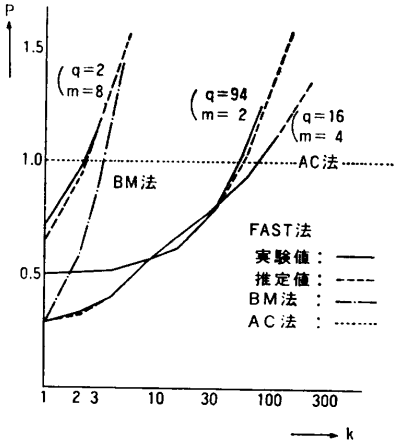

図 11 FAST 法の効率
Fig. 11 The probe rate of FAST.

3.4 FAST 法のメモリー節減手段

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になる．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．

有川らはこのようなときにこれを改善する方法を提案している[1]．“$2\times n$ 実現” 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からの遷移だけを表1の形で残し，それ以外を “$2\times n$ 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．

“文字符号分割による実現” は FAST 法にとってより現実的だと考えられる．これは文字符号を2つの部分符号に分割し，1回の遷移を2回の連続した遷移で置き換える方法である．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 $k=1000$, $m=3(\times 2)$ を仮定すれば，約 1.1M バイトになる．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，そのよう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

* C言語でコーディングして SUN-3 上で行った我々の実験では，AC 法と比較して，探索時間の実測値においても図11とほぼ同様の結果が得られている．なお，goto 関数，failure 関数の計算時間（オーバヘッド）は，m と k の積に大体比例し，AC 法の1.15倍程度であった．

およびパターン中の文字を牛島らの方法[9]等によってすべて同じ符号長に変換しておいてから，有川らが行っ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ったように[9]，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．

4. むすび

BM 法の「後方からの照合」というアイデアと AC 法の「パターン照合機械」のアイデアを結合して，複数パターンを高速に探索できる FAST 法を考案した．両者の結合に対する基本的な考え方を示し，具体的なアルゴリズムについて述べた．効率について考察し，実験によって，文字種が多いときやパターンが長いときにはパターン数が相当多いときでも高い探索効率が得られることを確認した．

この方式の欠点は多大のメモリを必要とすることであるが，この節減手段についても検討を加えた．

今後はさらに理論的および実験的に考察を進めて FAST 法の性質を明らかにしていきたいと考えている．また，FAST 法を2次元パターン探索にも適用できるように拡張することも検討している[11]．

謝辞 おわりに，日頃ご指導をいただく当所画像研究部二宮部長，沓沢前部長，尾関主任研究員に深く感謝する．江原主任研究員にはアルゴリズムを考案するに至る示唆をいただき，九州大学の有川教授には貴重なご助言をいただいた．井上誠喜職員をはじめ同僚諸氏には議論等を通してご援助をいただいた．心から感謝する．

参考文献

1) 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).
2) Boyer, R. S. and Moore, J. S.: A Fast Searching Algorithm, *Comm. ACM*, Vol. 20, No. 11, pp. 762-772 (1977).
3) 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).
4) Knuth, D. S., Morris, J. H. and Pratt, V. R.: Fast Pattern Matching in Strings, *SIAM J. Comput.*, Vol. 6, No. 2, pp. 323-350 (1977).
5) 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).
6) Kowalski, G. and Meltzer, A.: New Multi-Term High Speed Text Search Algorithms, *First International Conference on Computer and Applications* (CAT. No. 84 ch 2039-6), 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, pp. 118-132, Springer (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) 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，日経マグロウヒル社 (1987).
10) 浦谷則好：複数文字照合アルゴリズム，信学技報，SS 87-27, pp. 1-7 (1988).
11) 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).

(昭和63年10月3日受付)
(平成元年6月13日採録)

浦谷 則好（正会員）

昭和25年生．昭和50年東京大学大学院修士課程（電気工学）修了．同年NHKに入局．富山放送局を経て，昭和54年より放送技術研究所に勤務し，現在に至る．これまでに，リモートセンシングデータ解析，日本語処理および情報検索の研究に従事．現在，機械翻訳システムの研究開発に従事．電子情報通信学会，テレビジョン学会各会員．