Institute of Electronics, Information, and Communication Engineers

論文

# 複数パターンの文字列照合におけるマッチングマシンの動的構成法

広瀬 貞樹[†] 小栗 伸幸[††] 吉澤 寿夫[†] 山淵 龍夫[†]

# Dynamic Construction of Matching Machine on Multiple String Pattern Matching

Sadaki HIROSE[†], Nobuyuki OGURI[††], Tosio YOSIZAWA[†], and Tatuo YAMABUTI[†]

**あらまし** 複数パターンの文字列照合とは，テキストと呼ばれる比較的長い一つの文字列の中にパターンと呼ばれる複数個の文字列のうちのどれかが文字部分列として含まれるか否かを判定し，含まれていればテキスト上の位置を見つける問題である．代表的なアルゴリズムとして知られる Aho-Corasick 法 [1]（以下 AC 法と略記する）は，与えられたパターンからマッチングマシンと呼ばれるオートマトンを構成し，それを用いて照合する方法である．また AC 法を動的なパターン集合に対処できるようにするために，マッチングマシンを局所的に更新する動的構成法も既に提案されている [2]．最近，パターンが一つの場合の効率の良いアルゴリズムとして知られる Boyer-Moore 法 [3] に，決定性オートマトンの概念を導入して複数パターンへ拡張したアルゴリズム FAST 法 [4]～[6]，MBM 法 [7] 等が提案され，AC 法より高速で実用的なアルゴリズムであることが示された．本論文では，MBM 法を動的なパターン集合に対処できるようにするために，マッチングマシンを動的に構成するアルゴリズムを提案する．10 個から 1500 個のパターン集合に対する実験により，パターンを一つ追加する場合でマッチングマシンの構成時間が約 2.7 倍～78.2 倍，削除する場合で約 2.3 倍～30.3 倍高速になることが確認された．また，テキストとの照合時間も含めると，AC 法の動的構成法よりも高速であることが確認された．

**キーワード** 文字列照合，ストリングパターンマッチング，テキスト編集，情報検索

## 1. まえがき

文字列照合（ストリングパターンマッチング）とは，テキストと呼ばれる比較的長い一つの文字列の中にパターンと呼ばれる文字列が文字部分列として含まれるか否かを判定し，含まれていればテキスト上の位置を見つけるという問題で，文章の編集やデータの検索などに用いられる基本的な処理の一つである．この問題の処理アルゴリズムは，照合されるパターンの数が一つの場合と複数の場合とで大きく二つに分類される．

パターンの数が一つの場合の代表的なアルゴリズムとして，Knuth-Morris-Pratt 法 [8]（以下 KMP 法と略記する）と Boyer-Moore 法 [3]（以下 BM 法と略記する）がよく知られている．KMP 法は，テキストとパターンの先頭（左端）をそろえ，左から順にテキストとパターンの対応する文字が一致しているか否かを比較していき，一致しなかったらパターンを右に移動して同様のことを繰り返し，パターンと一致する位置を探す方法である．照合に失敗したらパターンをどれだけ右に移動できるかをあらかじめ計算しておき，テキストを戻ることなく 1 回走査するだけで済むようにした点が特長である．テキストの長さを $n$，パターンの長さを $m$ とすると，KMP 法の最大時間計算量（最悪の場合の計算時間）は $O(n+m)$ である．BM 法は，テキストとパターンの先頭をそろえて比較を開始するが，KMP 法と違ってパターンの末尾（右端）から先頭に向かって比較をする方法である．照合に失敗した場合，パターンを何文字分も右に移動し，必ずしもテキストのすべての文字を走査する必要がない点が特長である．BM 法の最大時間計算量は $O(n+r*m)$（$r$ はテキスト中に見つかるパターンの数）であるが，実際上は BM 法の方が格段に高速であることが知られている [9], [10]．

一方，照合されるパターンの数が複数の場合についても多くのアルゴリズムが知られているが，その中でも Aho-Corasick 法 [1]（以下 AC 法と略記する）が有

† 富山大学工学部知能情報工学科，富山市
Faculty of Engineering, Toyama University, Toyama-shi, 930 Japan
†† 富士ソフト ABC 株式会社八王子事業所，八王子市
Hachioji Office, FujisoftABC Inc., Hachioji-shi, 192 Japan



NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers



名である．AC 法は，与えられたパターンからマッチングマシンと呼ばれるオートマトンを構成し，それを用いて照合する方法であり，テキストを戻ることなく1回走査する．その意味で KMP 法の複数パターンへの拡張アルゴリズムであると考えられる．

AC 法で1度照合した後，パターンのうちのいくつかを更新（追加，削除）して再度照合する場合(注1)，マッチングマシンを1から作り直すより前の照合に用いたマッチングマシンを局所的に変更して構成時間を短縮する動的構成法も既に提案されている [2]．

最近，BM 法に決定性オートマトンの概念を導入して複数パターンへ拡張したアルゴリズム FAST 法 [4]～[6]，MBM 法 [7] 等が提案され，AC 法より高速で実用的なアルゴリズムであることが示された．

本論文では，MBM 法においてマッチングマシンを動的に構成するアルゴリズムを提案する．まず **2.** では MBM 法の概略を説明する．**3.** では追加処理，**4.** では削除処理におけるマッチングマシンの動的構成法について詳しく説明する．また，**5.** では実験によって本提案アルゴリズムの有効性を実証する．

## 2. MBM法

MBM 法は，与えられたパターンからマッチングマシンを構成し，それを用いてテキストとの照合を行うという手法である．本章ではその概略を説明する．

マッチングマシンは，基本的には決定性オートマトンであり，状態遷移関数に対応する goto 関数，出力関数に対応する output 関数に加え，照合の際にテキストとの比較が失敗した場合のテキストポインタの移動量を与える skip1，skip2 関数からなる．

これらの関数は形式的に次のように定義される．

goto 関数　$g$ ：$S \times I \rightarrow S$

output 関数　$o$ ：$S \rightarrow P$

skip1 関数　$skip1$ ：$I \rightarrow N$

skip2 関数　$skip2$ ：$S \times I \rightarrow N$

ここで，S は状態の有限集合，I は使用する文字の有限集合，P は与えられたパターンの集合，N は自然数の集合である．

これ以後の記述を簡単にするために goto 関数の定義域を次のように拡張しておく．

goto 関数　$g$ ：$S \times I^* \rightarrow S$

但し，$I^*$ は I の文字を使ってできる有限の長さのすべての文字列の集合を表す．

これ以後，与えられた複数個のパターンの長さの最小値を $pm$ とする．MBM 法では照合の際，テキストとの比較が失敗した場合にテキストポインタの位置を移動し再度比較を開始するが，長さの一番短いパターン（長さ $pm$ のパターン）の照合を抜かさないようにテキストポインタを移動することに注意されたい．

```
begin
  s:=0; tp:=pm;
  while tp ≦ n do
    begin
      if g(s, text(tp)) is defined then
        begin
          s := g(s, text(tp));
          if o(s) ≠ empty then print tp, o(s);
          tp := tp - 1;
        end
      else
        begin
          if s = 0 then
            tp := tp + skip1(text(tp));
          else
            tp := tp + skip2(s, text(tp));
          s := 0;
        end
    end
end
```

図1　照合アルゴリズム
Fig. 1　Matching algorithm.

前処理でマッチングマシンを構成し，テキストとの照合を行う．照合アルゴリズムを図1に示す．

まず，マッチングマシンのオートマトンの状態 $s$ を初期状態 0 とし，テキストポインタの位置 $tp$ を $pm$ として照合を開始する．$tp$ がテキストの長さ $n$ 以下である間次のことを繰り返す．$g(s, text(tp))$（状態 $s$ で，$tp$ の指しているテキスト上の文字 $text(tp)$ による状態遷移）が定義されていれば，$g(s, text(tp))$ へ状態遷移する．遷移した状態で出力が定義されていれば，テキストポインタ $tp$ と出力 $o(s)$ をプリントする．そして，テキストポインタを一つ左に移動する．また，$g(s, text(tp))$ が定義されていなければ，テキストポインタを移動し，初期状態 0 へ遷移する．このときのテキストポインタの移動量は，状態 $s$ が初期状態 0 なら skip1 関数で，それ以外の状態なら skip2 関数でそれぞれ与えられ，その値だけテキストポインタを右に移動する．

(注1)：このような場合，追加されたパターンだけで再度照合し，その結果を最初の照合結果に加える，あるいは，削除パターンだけの照合結果を最初の照合結果から削除するという方法が考えられる．しかしこの方法では，パターンを加えたり，削除する手間が最悪の場合テキストの長さに比例することになり，かえって効率が悪いことに注意されたい．



NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers



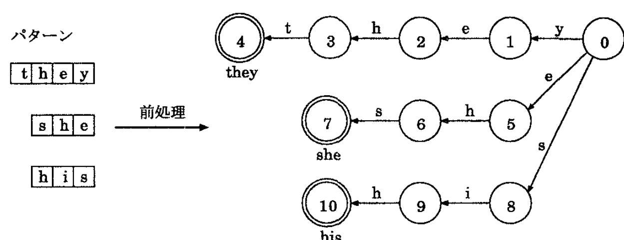


図2　オートマトンの例
Fig. 2　An example of automaton.

それぞれの関数についてもう少し詳しく説明する．

### 2.1　オートマトン（goto関数，output関数）

goto関数はオートマトンの状態遷移関数に対応する．MBM法はパターンの末尾から照合を開始するので，goto関数はパターンの末尾から順に構成される．

output関数はオートマトンの出力関数に対応する．初期状態からパターンの各文字を末尾からたどって遷移した状態（受理状態）の出力としてそのパターンが割り当てられる．

図2に，与えられたパターンが“*they*”，“*she*”，“*his*”の場合のオートマトンを示す(注2)．

ここで，丸印は状態，特に2重丸は受理状態を表し，矢印はgoto遷移（goto関数による状態遷移）を表している．但し，図に記された以外にgoto遷移は定義されていない（未定義である）．また，受理状態の下にある文字列はその状態の出力を表し，受理状態以外の状態の出力はない（空（empty）である）．

### 2.2　skip1関数

skip1関数は，オートマトンが初期状態0にいて，$tp$の指しているテキスト上の文字$text(tp)$による次の遷移先が未定義である場合のテキストポインタの移動量を与える．

$g(0, text(tp))$が未定義であるということは，$text(tp)$がどのパターンの一番右端の文字とも異なっているということを意味する．従って，この位置（パターンの右端の文字をテキストの$tp$の指している文字に合わせた位置）ではどのパターンもテキストと照合しないことがわかるので，テキストポインタを移動する．このとき，$text(tp)$と同じ文字がいずれかのパターンに含まれていれば，それと$text(tp)$の位置を一致させるように（オートマトンを右に移動し，パターンの右端から照合を再開できるように）テキストポインタを移動する．$text(tp)$と同じ文字がパターンの何箇所にも含まれている場合には右端に最も近いものを採用する．また，この値は$pm$の値を超えない．

表1　skip1関数の例
Table 1　An example of skip1 function.

| 文字 | $e$ | $h$ | $i$ | $s$ | $y$ | その他 |
|---|---|---|---|---|---|---|
| **skip1** | 0 | 1 | 1 | 0 | 0 | 3 |

表1に，与えられたパターンが“*they*”，“*she*”，“*his*”の場合のskip1関数を示す．

オートマトンが初期状態0にいて，$text(tp) = i$とする．$g(0, i)$は未定義である．すなわち，$i$はどのパターンの一番右端の文字（$y$，$e$，$s$）とも異なっているので，この位置でのテキストとの照合に失敗しテキストポインタを移動する．ここで，$skip1(i) = 1$であるから，テキストポインタを右に一つ移動する．この動作は，文字$i$がパターン“*his*”の$i$として照合する可能性をチェックするための動作である．

### 2.3　skip2関数

skip2関数は，オートマトンが初期状態以外の状態$s$にいて，$text(tp)$による次の遷移先が未定義である場合のテキストポインタの移動量を与える．

初期状態からテキスト上の文字列（$v$とする）で状態$s$に遷移した後，$g(s, text(tp))$が未定義であるということは，どのパターンも右文字部分列（文字列$u$が$u = u_1 \cdot u_2$と書けるとき，$u_1$（$u_2$）を$u$の左（右）文字部分列と呼ぶ）として$text(tp)$と$v$を連接した文字列$w$　（$w = text(tp) \cdot v$）を含んでいないことを意味する．従って，この位置ではどのパターンもテキストと照合しないことがわかるので，テキストポインタを移動する．このとき，$w$と同じ文字列（正確には同じでなくてよいが，詳細は次章で説明する）がいずれかのパターンに含まれていれば，その文字列を一致させるようにテキストポインタを移動する．同じ文字列がパターンの何箇所にも含まれている場合には右端に最も近いものを採用する．またこの値は$pm + depth(s)$の値を超えない．但し，$depth(s)$は，初期状態から状態$s$までの遷移に用いる文字列$v$の長さである．

表2に，与えられたパターンが“*they*”，“*she*”，“*his*”の場合のskip2関数を示す．

オートマトンが状態6にいて$text(tp) = t$とする．オートマトンは，初期状態から文字$e$，$h$（すなわち

(注2)：簡単のため，このような単純な例で説明するが，パターンが共通の接尾語をもてば，その接尾語に対応する状態は共通になる．また，一つのパターンが他のパターンの部分語になっている場合もあるので，この後で説明するskip1関数やskip2関数の計算はより複雑になる．



NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers



表 2　skip2 関数の例
Table 2　An example of skip2 function.

| 文字＼状態 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $h$ | 4 | 5 | 6 | 7 | 2 | 5 | 6 | 3 | 4 | 5 |
| $t$ | 4 | 5 | 6 | 7 | 4 | 3 | 6 | 3 | 4 | 5 |
| その他 | 4 | 5 | 6 | 7 | 4 | 5 | 6 | 3 | 4 | 5 |

テキスト上の文字列 $v$ ="$he$") で状態 6 に遷移した後，$g(6,t)$ が未定義であるので，この位置での照合に失敗（どのパターンも右文字部分列として $w$ ="$the$" を含んでいないことが判明）し，テキストポインタを移動する．ここで，$skip2(6,t)=3$ であるから，テキストポインタを右に三つ移動する．この動作は，文字列 "$the$"（パターン "$she$" の一部である可能性を調べていた）がパターン "$they$" の一部である可能性があるので，これをチェックするための動作である．

## 3. 追加処理の動的構成法

本章では，1 度テキストと照合した後，更にパターンを追加して再度照合する場合，前のマッチングマシンを局所的に変更して，マッチングマシンを再構成する方法について説明する．

新しく追加するパターンを $y=a_1a_2\cdots a_k$ とする．

### 3.1 goto，output 関数の変更アルゴリズム

新しく追加するパターン $y$ について，初期状態 0 から文字 $a_k, a_{k-1}, \cdots, a_d$ による goto 遷移は既に定義されており（$g(0, a_d \cdots a_{k-1}a_k) = s_d$ とする），かつ，状態 $s_d$ で文字 $a_{d-1}$ による goto 遷移が未定義であるとする．この場合は，新しい状態 $s_{d-1}$, $s_{d-2}$, $\cdots$, $s_1$ を作成し，文字 $a_{d-1}$, $a_{d-2}$, $\cdots$, $a_1$ に対する goto 遷移を定義する（$g(s_d, a_{d-1}) = s_{d-1}$，$g(s_{d-1}, a_{d-2}) = s_{d-2}$，$\cdots$，$g(s_2, a_1) = s_1$）．また，状態 $s_1$ を受理状態とし，その output 関数値を $y$ とする（$o(s_1) = y$）．

前章の例で更にパターン "$hers$" が追加されたとする．まず，パターンの末尾から文字 $s$, $r$, $e$, $h$ について，初期状態からの goto 遷移が定義されているかどうかを調べる．$g(0,s)=8$ であり，$g(8,r)$ は未定義である．$g(8,r)$ の遷移先として状態 11 を作り，$g(8,r)=11$ とする．同様にして $g(11,e)=12$，$g(12,h)=13$ とする．また，状態 13 を受理状態とし，その出力を "$hers$" とする（$o(13)$ = "$hers$"）．

再構成されたオートマトンを図 3 に示す．

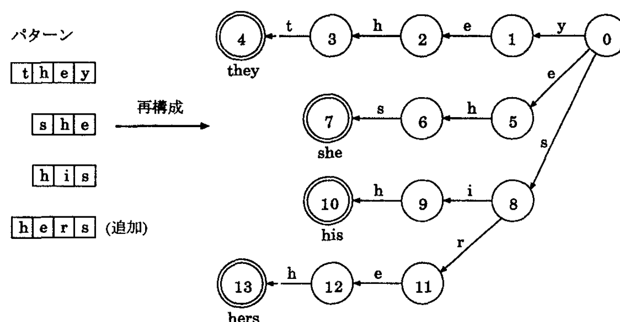



図 3　再構成されたオートマトン
Fig. 3　Reconstructed automaton.

### 3.2 skip1 関数の変更アルゴリズム

前節では新しい状態 $s_{d-1}, s_{d-2}, \cdots, s_1$ を作成し，文字 $a_{d-1}, a_{d-2}, \cdots, a_1$ に対する goto 遷移を定義した．この文字 $a_i$ ($d-1 \geqq i \geqq 1$) については，skip1 関数値が変わる可能性があるので求め直す必要がある．

表 3　再構成された skip1 関数
Table 3　Reconstructed skip1 function.

| 文字 | $e$ | $h$ | $i$ | $r$ | $s$ | $y$ | その他 |
|---|---|---|---|---|---|---|---|
| **skip1** | 0 | 1 | 1 | 1 | 0 | 0 | 3 |

パターン $p$ と文字 $a_i$ について，$num(p, a_i)$ を $p$ の右端の文字を 0 番目として $a_i$ が $p$ の右から何番目の文字であるのかを表すものとする．$skip1(a_i)$ は，すべてのパターンについての $num(p, a_i)$ の最小値である．すなわち，$skip1(a_i) = min\{num(p, a_i) \mid p \in P\}$ である．$y$ 以外のパターンについては，既に $skip1(a_i)$ が求められているので，その値と $y$ についての $num(y, a_i)$ の小さい方をあらためて $skip1(a_i)$ とする．すなわち，$skip1(a_i) = min\{skip1(a_i), num(y, a_i)\}$ とする．

更に，$pm$ の値が変更され $k$ となる（$y$ の長さ $k$ が $pm$ より小さい）場合には，$y$ との照合を抜かさないように，skip1 関数値が $k$ $(=pm)$ よりも大きいものについてはその値を $k$ とする．すなわち，すべての文字 $a$ について，$skip1(a) = min\{skip1(a), k\}$ とする．

前章の例で更にパターン "$hers$" が追加されたとする．文字 $r$, $e$, $h$ について skip1 関数値を求め直す．$skip1(r) = min\{skip1(r),\ num(\text{"}hers\text{"}, r)\} = min\{3, 1\} = 1$，同様に $skip1(e) = 0$，$skip1(h) = 1$ とする．また，"$hers$" の長さは 4 で，$pm$ の値 3 より大きいので，これで skip1 関数の再構成を終了する．

再構成された skip1 関数を表 3 に示す．

### 3.3 skip2 関数の変更アルゴリズム

前章で簡単に説明したが，skip2 関数は，初期状態からテキスト上の文字列（$v$ とする）で初期状態以外の状態 $s$ に遷移した後，$text(tp)$ による goto 遷移が



NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers



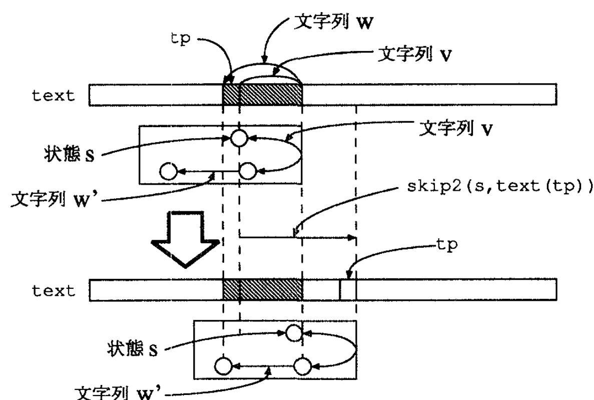



図4　skip2 関数によるテキストポインタの移動（場合 1）
Fig. 4　Updating *tp* by skip2 function (case 1).

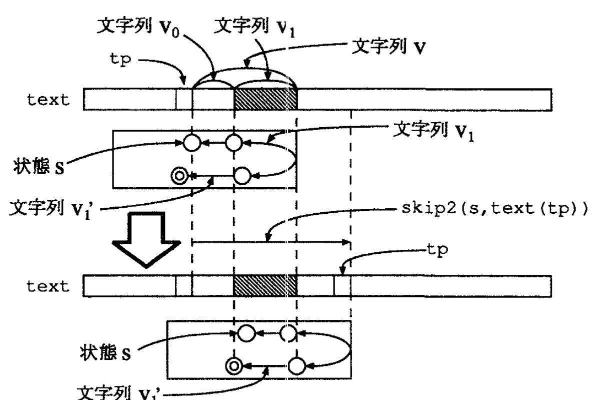



図5　skip2 関数によるテキストポインタの移動（場合 2）
Fig. 5　Updating *tp* by skip2 function (case 2).

未定義である場合のテキストポインタの移動量を与える．次の二つの場合を考慮する必要がある．

まず，$w = text(tp) \cdot v$ とするとき，$w$ と同じ文字列が，あるパターンの右端でない部分に含まれている（便宜上その文字列を $w'$ とするが，$w$ と $w'$ は同じ文字列である）場合である（図 4，上）．この場合，$w$ が $w'$ と照合する可能性があるので，これを考慮してテキストポインタを移動する（図 4，下）．

またもう一つは，$v = v_0 \cdot v_1$ と書け，あるパターンの左文字部分列として $v_1$ が含まれている（便宜上その文字列を $v_1'$ とするが，$v_1$ と $v_1'$ は同じ文字列である）場合である（図 5，上）．この場合，$v_1$ が $v_1'$ と照合する可能性があるので，これを考慮してテキストポインタを移動する（図 5，下）．

ここで，skip2 関数を求める際に用いる failure 関数（$f: \mathrm{S} \rightarrow \mathrm{S}$）について説明する．

$s$ を初期状態から文字列 $v$ で goto 遷移する状態とする．このとき $s$ の failure 関数値 $f(s)$ を，初期状態から goto 遷移できる $v$ の左文字部分列のうち最長の部分列（$v_0$ とする）により遷移する状態（$f(s) = g(0, v_0)$）とする．

表 4 に図 3 から求められる failure 関数を示す．例えば $f(3) = 6$ であるのは，初期状態から状態 3 まで文字列 $v =$“*hey*” で goto 遷移するが，$v$ の左文字部分列のうち $v_0 =$“*he*” が初期状態から goto 遷移できる最長の文字列であり，$g(0,$“*he*”$) = 6$ ということである．

この failure 関数を用いてある状態 $s$ の skip2 関数値を求める方法を説明する．

まず，図 4 の場合を考える．状態 $s$ に何回かの failure 遷移（failure 関数による状態遷移）で遷移する受理状態以外の状態 $r_1, r_2, \cdots, r_q$ と，次の goto 遷移を定義している文字 $a_{r_1}, a_{r_2}, \cdots, a_{r_q}$ を求める（図 6(1)，点線は failure 遷移を表す）．初期状態から状態 $r_j$ ($1 \leq j \leq q$) に文字列 $v_{r_j}$ で goto 遷移するものとすると，failure 関数の性質から，$v$ は $v_{r_j}$ の左文字部分列である．よってもし $text(tp) = a_{r_j}$ であれば，$w = text(tp) \cdot v$ は $a_{r_j} \cdot v_{r_j}$ の左文字部分列となる．この場合，状態 $s$ の位置に $r_j$ を移動させ，照合を再開できるようにテキストポインタを移動すればよい．従って，テキストポインタを開始地点（初期状態でテキストポインタが指していた位置）に戻すための移動量 $depth(s)$ と開始地点の右方向への移動量 $depth(r_j) - depth(s)$ との和とこれまでの skip2 関数値の最小値をあらためて skip2 関数値とする．すなわち，$skip2(s, a_{r_j}) = min\{skip2(s, a_{r_j}), depth(r_j)\}$ とする．

表4　failure 関数の例
Table 4　An example of failure function.

| 状態 | 2 | 3 | 7 | 12 | 13 |
|---|---|---|---|---|---|
| failure | 5 | 6 | 8 | 5 | 6 |

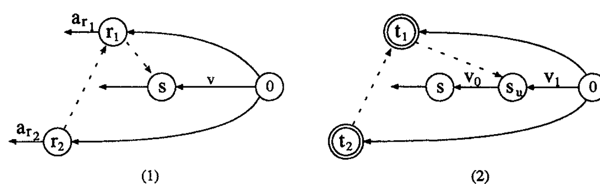



図6　skip2 関数の変更
Fig. 6　Updating skip2 function.

次に，図 5 の場合を考える．$s_u$ を初期状態から文字列 $v_1$ で goto 遷移する状態，$s$ を状態 $s_u$ から長さ 0 以上の文字列 $v_0$ で goto 遷移する状態とする（図 6(2)）．



NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers



状態 $s_u$ に何回かの failure 遷移で遷移する状態のうち受理状態であるものを $t_1, t_2, \cdots, t_q$ とする．初期状態から文字列 $v_{t_j}$ で状態 $t_j$ $(1 \leqq j \leqq q)$ に goto 遷移するものとすると，failure 関数の性質から，$v_1$ は $v_{t_j}$ の左文字部分列となる．この場合，状態 $s_u$ のあった位置に $t_j$ を移動させ，照合を再開できるようにテキストポインタを移動すればよい．従って，テキストポインタを開始地点に戻すための移動量 $depth(s)$ と開始地点の右方向への移動量 $depth(t_j) - depth(s_u)$ との和とこれまでの skip2 関数値の最小値をあらためて skip2 関数値とする．すなわち，すべての文字 $a$ に対して $skip2(s, a) = min\{skip2(s, a), depth(s) + depth(t_j) - depth(s_u)\}$ とする．

これらのことを考慮して skip2 関数の再構成を行う．

前々節で，パターン $y$ に対して新しい状態 $s_{d-1}, s_{d-2}, \cdots, s_1$ を作成した．これらの状態 $s_i$ $(d-1 \geqq i \geqq 1)$ についての skip2 関数値は定義されていないので新しく求める必要がある．また，$y$ が追加されたことにより，以前の skip2 関数値を変更しなければならない場合があるので，それについても求め直す必要がある．

まず，状態 $s_i$ の skip2 関数値を初期化する．オートマトンが状態 $s_i$ にいるということは，テキストポインタが開始地点から $depth(s_i)$ だけ左に移動しているということである．テキストポインタを開始地点に戻すための移動量 $depth(s_i)$ と開始地点から右方向への最大移動量 $pm$ との和が skip2 関数の初期値（最大値）となる．従って，すべての文字 $a$ に対して $skip2(s_i, a) = depth(s_i) + pm$ とする．

状態 $s_i$ について図 4 の状況を考える．何回かの failure 遷移で状態 $s_i$ に遷移する受理状態以外の状態 $r$ $(f(f(\cdots f(r))) = s_i)$ と，その状態から次の goto 遷移を定義している文字 $a_r$ を求め，$skip2(s_i, a_r) = min\{skip2(s_i, a_r), depth(r)\}$ とする．

状態 $s_i$ について図 5 の状況を考える．初期状態から $y$ で goto 遷移する経路にある状態の集合を $S(y)$ とする．$S(y)$ に含まれる状態 $s_u$ に何回かの failure 遷移で遷移する状態のうち受理状態である状態 $t$ $(f(f(\cdots f(t))) = s_u \in S(y))$ を求める．状態 $s_u$ から長さ 0 以上の文字列 $v_0$ で goto 遷移できる状態 $s_i$ と，すべての文字 $a$ に対して $skip2(s_i, a) = min\{skip2(s_i, a), depth(s_i) + depth(t) - depth(s_u)\}$ とする．

次に，$y$ が追加されたことにより，以前の skip2 関数値を変更しなければならない場合の変更方法を示す．

まず，図 4 の場合を考える．新しい状態 $s_i$ $(d-1 \geqq i \geqq 1)$ に goto 遷移する状態 $s_{i+1}$ から何回かの failure 遷移により遷移する状態 $s$ $(f(f(\cdots f(s_{i+1}))) = s)$ と，状態 $s_{i+1}$ で次の goto 遷移を定義している文字 $a_i$ について $skip2(s, a_i) = min\{skip2(s, a_i), depth(s_{i+1})\}$ とする．

次に，図 5 の場合を考える．新しい受理状態 $s_1$ から何回かの failure 遷移で遷移する状態を $s_u$ とする $(f(f(\cdots f(s_1))) = s_u)$．ここで，状態 $s_u$ から長さ 0 以上の文字列 $v_0$ で goto 遷移できる状態 $s$ とすべての文字 $a$ に対して $skip2(s, a) = min\{skip2(s, a), depth(s) + depth(s_1) - depth(s_u)\}$ とする．

更に，$pm$ が変更され $k$ となる（$y$ の長さ $k$ が $pm$ より小さい）場合には，$y$ との照合を抜かさないように，すべての状態 $s$ とすべての文字 $a$ に対して $skip2(s, a) = min\{skip2(s, a), depth(s) + k\}$ n とする．

前章の例で更にパターン "$hers$" が追加されたとする．はじめに，新しく作成された状態 11，12，13 について skip2 関数値を求める．まず，すべての文字 $a$ について $skip2(11, a) = depth(11) + pm = 2 + 3 = 5$，同様に $skip2(12, a) = 6$，$skip2(13, a) = 7$ として，関数値を初期化する．次に，図 4 の場合，新しい状態 11，12，13 に failure 遷移してくる状態を探す．この例ではそのような状態がないので特に何もしない．図 5 の場合，初期状態から "$hers$" によって goto 遷移する経路にある状態 8，11，12，13 に，何回かの failure 遷移で遷移する状態のうち受理状態であるものに 7 があり $(f(7) = 8)$，状態 8 から goto 遷移する新しい状態に 11，12，13 があるので，すべての文字 $a$ に対して $skip2(11, a) = min\{skip2(11, a),\ depth(11) + depth(7) - depth(8)\} = min\{6, 2 + 3 - 1\} = 4$，同様に $skip2(12, a) = 5$，$skip2(13, a) = 6$ とする．

次に，"$hers$" が追加されたことにより，以前の skip2 関数値を変更しなければならない場合を考える．図 4 の場合，新しい状態に goto 遷移する状態 8，11，12 から何回かの failure 遷移で遷移する状態に 5 がある $(f(12) = 5)$．状態 12 で次の goto 遷移を定義している文字は $h$ であるので $skip2(5, h) = min\{skip2(5, h),\ depth(12)\} = min\{2, 3\} = 2$ とする．次に図 5 の場合，新しい受理状態 13 から何回かの failure 遷移で遷移する状態に 6 がある $(f(13) = 6)$．状態 6 から goto 遷移する状態に 7 があ



NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers



表 5 再構成された skip2 関数
Table 5 Reconstructed skip2 function.

| 文字＼状態 | 1,9,11 | 2,7,10,12 | 3,13 | 4 | 5 | 6 | 8 |
|---|---|---|---|---|---|---|---|
| $h$ | 4 | 5 | 6 | 7 | 2 | 4 | 3 |
| $t$ | 4 | 5 | 6 | 7 | 4 | 3 | 3 |
| その他 | 4 | 5 | 6 | 7 | 4 | 4 | 3 |

る（$g(6,s)=7$）．この状態 6，7 について skip2 関数値を求め直す．まず，$skip2(6,t)=min\{skip2(6,t), depth(6)+depth(13)-depth(6)\}=min\{3,2+4-2\}=3$ とし，$t$ 以外のすべての文字 $a$ について $skip2(6,a)=min\{skip2(6,a), depth(6)+depth(13)-depth(6)\}=min\{5,2+4-2\}=4$ とする．同様に，すべての文字 $a$ について $skip2(7,a)=5$ とする．

再構成された skip2 関数を表 5 に示す．

## 4. 削除処理の動的構成法

本章では，1 度テキストと照合した後，パターンを削除して再度照合する場合，前のマッチングマシンを局所的に変更して，再構成する方法について説明する．

削除するパターンを $z=b_1b_2\cdots b_l$ とする．

### 4.1 goto，output 関数の変更アルゴリズム

パターン $z$ について，初期状態から文字 $b_l$, $b_{l-1}$, $\cdots, b_d$ による goto 遷移は他のパターンと共通で（$g(0, b_d\cdots b_{l-1}b_l)=s_d$），状態 $s_d$ から文字 $b_{d-1}$, $b_{d-2}$, $\cdots$, $b_1$ による goto 遷移（$g(s_d, b_{d-1})=s_{d-1}$, $g(s_{d-1}, b_{d-2})=s_{d-2}$, $\cdots$, $g(s_2, b_1)=s_1$）は $z$ だけの遷移であるとする（このような状態 $s_d$ は，初期状態から $z$ で goto 遷移する経路にある状態の中で，その状態からの goto 遷移が複数個定義されているかどうかで見つけることができる）．このとき，状態 $s_{d-1}$, $s_{d-2}$, $\cdots$, $s_1$ を削除し，これらの状態に遷移する goto 遷移を削除する．更に，受理状態 $s_1$ の output 関数値 $o(s_1)=z$ を削除する．(skip1，skip2 関数を再構成するのに必要なので，実際には最後に削除する．)

### 4.2 skip1 関数の変更アルゴリズム

skip1 関数を求めるとき，テキストポインタの移動量の最小値をとるようにした．従ってパターン $z$ が削除されたことによって移動量を大きくすることができる場合には，skip1 関数を求め直さなければならない．

削除する goto 遷移に用いられる文字 $b_i$ ($d-1\geq i\geq 1$) について，skip1 関数値が $num(z, b_i)$ である（$skip1(b_i)=num(z,b_i)$）ものについては，skip1 関数値が変わる可能性があるのでこれを求め直す．まず，$skip1(b_i)=pm$ として初期化し，次に，文字 $b_i$ が含まれているパターン $p$ について $skip1(b_i)=min\{skip1(b_i), num(p,b_i)\}$ とする．

また，$pm\ (=l)$ が変更される場合には，$skip1(a)=l$ である文字 $a$ については，skip1 関数値が変わる可能性があるのでこれを求め直す．まず，$skip1(a)=pm$ として初期化し，次に，文字 $a$ が含まれているパターン $p$ について $skip1(a)=min\{skip1(a), num(p,a)\}$ とする．

### 4.3 skip2 関数の変更アルゴリズム

パターン $z$ の長さが最小であった場合には，$pm$ が変更される．$pm$ が変更される場合には，$skip2(s,a)=depth(s)+l$ である状態 $s$ と文字 $a$ について skip2 関数値が変わる可能性がある．しかし，このような状態は一般に多くあり，状態一つひとつについて順に skip2 関数値を求め直すのはかえって効率が悪いので，この場合には再構成せずに 1 から skip2 関数を作り直すことにする．以下では，$pm$ が変更されない場合の skip2 関数の再構成法について述べる．

パターン $z$ に対して状態 $s_{d-1}, s_{d-2}, \cdots, s_1$ を削除した．これらの状態 $s_i$ ($d-1\geq i\geq 1$) についての skip2 関数値を削除する．また，$z$ が削除されたことにより，以前の skip2 関数値を変更しなければならない場合があるので，それについては求め直す必要がある．

まず，削除する状態 $s_i$ ($d-1\geq i\geq 1$) とすべての文字 $a$ に対して $skip2(s_i, a)$ を削除する．

次に，パターン $z$ が図 4 の状況で，ある状態の skip2 関数値に関与している場合を考える．削除する状態 $s_i$ ($d-1\geq i\geq 1$) に goto 遷移する状態 $s_{i+1}$ から何回かの failure 遷移で遷移する状態を $s$ とすると，その状態の skip2 関数値が変わる可能性があるので求め直す．状態 $s_{i+1}$ で次の goto 遷移を定義している文字 $b_i$ について $skip2(s,b_i)=pm+depth(s)$ とし，skip2 関数値を初期化する．

状態 $s$ について図 4 の状況を考える．何回かの failure 遷移で状態 $s$ に遷移する受理状態以外の状態 $r$ と，その状態から次の goto 遷移を定義している文字 $a_r$ を求め，$skip2(s,a_r)=min\{skip2(s,a_r), depth(r)\}$ とする．

状態 $s$ について図 5 の状況を考える．初期状態から状態 $s$ まで goto 遷移する経路にある状態 $s_u$（状態 $s$ を含む）に，何回かの failure 遷移で遷移する状態のうち受理状態であるものを $t$ とする．すべて



NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers



表6　マッチングマシンの構成時間の比較
Table 6　Comparison on construction time of matching machine.

| | | | | | | |
|---|---|---|---|---|---|---|
| パターン集合 | | | | | | |
| パターンの個数 | 10 | 50 | 100 | 500 | 1000 | 1500 |
| パターン長の総和 | 58 | 304 | 541 | 2702 | 5471 | 8509 |
| MBM法 | | | | | | |
| マッチングマシンの状態数 | 57 | 267 | 449 | 2011 | 3937 | 6007 |
| マッチングマシンの構成時間（ms） | 83 | 300 | 700 | 9383 | 31250 | 72766 |
| 動的構成法（追加処理） | | | | | | |
| マッチングマシンの再構成時間の平均（ms） | 30.00 | 31.53 | 43.00 | 182.10 | 458.80 | 929.71 |
| 効果（構成時間/再構成時間の平均） | 2.76 | 9.51 | 16.27 | 51.52 | 68.11 | 78.26 |
| 動的構成法（削除処理） | | | | | | |
| マッチングマシンの再構成時間の平均（ms） | 35.00 | 40.33 | 59.78 | 426.11 | 1191.80 | 2399.24 |
| 効果（構成時間/再構成時間の平均） | 2.37 | 7.43 | 11.70 | 22.02 | 26.22 | 30.32 |

の文字 $a$ に対して，$skip2(s,a) = min\{skip2(s,a), depth(s) + depth(t) - depth(s_u)\}$ とする．

次に，パターン $z$ が図5の状況で，ある状態のskip2関数値に関与している場合を考える．削除する状態の受理状態 $s_1$ から何回かのfailure遷移により遷移する状態を $s_u$ とする．ここで，$s_u$ から長さ0以上の文字列 $v_0$ でgoto遷移できる状態を $s$ とすると，その状態のskip2関数値が変わる可能性があるので求め直す．すべての文字 $a$ について $skip2(s,a) = pm + depth(s)$ と初期化し，更に，先ほどと同様に図4，5の場合を考慮に入れて求め直す．

## 5. 実　　験

本章では，マッチングマシンの動的構成法の有効性を示すために行った実験とその結果について説明する．なお，実験にはSUN Sparc 5（110 MHz）を使用した．プログラムはC言語で開発したが，追加処理は299ステップ，削除処理は442ステップであった．

### 5.1 マッチングマシンの構成時間の比較

MBM法と本論文で提案した動的構成法について，マッチングマシンの構成時間を比較する実験を行った．

10個，50個，100個，500個，1000個，1500個のパターンをそれぞれランダムに作成し，そのマッチングマシンを構成して，その構成時間をMBM法におけるマッチングマシンの構成時間とした．動的構成法においては，追加処理の場合は一つのパターンを除いて構成したマッチングマシンにそのパターンを追加するという処理をすべてのパターンについて実行し，その平均時間を再構成時間とした．また削除処理の場合は，MBM法で構成したマッチングマシンから一つのパターンを削除するという処理をすべてのパターンについて実行し，その平均時間を再構成時間とした．

表6に実験の結果を示す．パターンの個数が10個，50個，100個，500個，1000個，1500個と増えるに従って動的構成法の効果が増大し，追加処理で約2.7倍～78.2倍，削除処理で約2.3倍～30.3倍高速になることが確認される．

このことは次のように理解することができる．パターンの個数が増えるとマッチングマシンの状態数が増え，MBM法でマッチングマシンを構成する時間が増大する．一方動的構成法においては，マッチングマシンの状態数が増えても，更新するパターン1個に対応するマッチングマシンの変更部分はあまり変化しないので，マッチングマシンを再構成する時間はそれほど増大しない．すなわち，パターンの個数が増えるとマッチングマシン全体に対する動的構成法による変更部分の比率が小さくなるからであると考えられる．

### 5.2 パターンの長さと構成時間

前節の実験結果から，パターンの個数が増えるに従って動的構成法の効果が増大することがわかったが，本節では，パターンの長さによって動的構成法の効果がどのようになるかを調べる実験を行った．パターンの長さは一般には異なるが，パターンの長さによって動的構成法の効果がどのようになるかを調べるために，一つのパターン集合についてはすべてのパターンの長さが一定であるものと仮定した．また，マッチングマシンが大きくなると動的構成法の効果が現れやすいので，小さいところでも効果があるか否かを調べるために，パターンの数を2から10とし，パターンの長さを1から10とした．

1から10までの長さのパターンを2から10個ランダムに作成し，そのマッチングマシンを構成するという試行を100回繰り返し，その平均時間を構成時間とした．但し，動的構成法において，例えばパターンが5



NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers



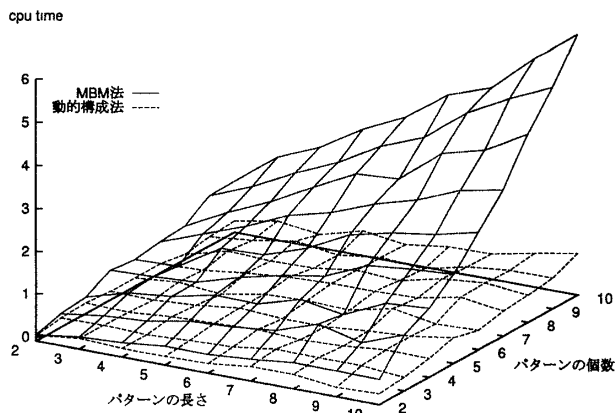



図7 マッチングマシンの構成時間の比較（パターンの長さが一定の場合：追加処理）
Fig. 7 Comparison on construction time of matching machine (pattern length is constant : add).

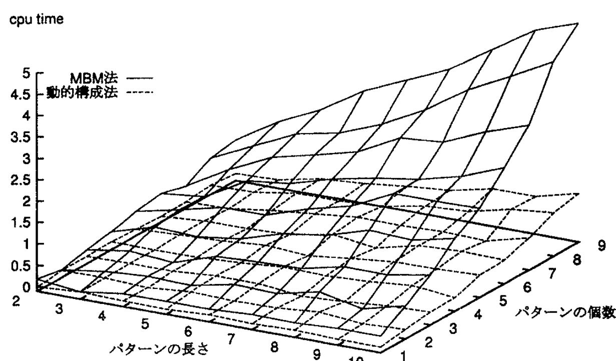



図8 マッチングマシンの構成時間の比較（パターンの長さが一定の場合：削除処理）
Fig. 8 Comparison on construction time of matching machine (pattern length is constant : remove).

個であるというのは，追加処理の場合にはパターンが4個で構成されたマッチングマシンに更に1個追加したときの再構成時間を，削除処理の場合にはパターンが6個で構成されたマッチングマシンから1個削除したときの再構成時間を示している．追加処理，削除処理について比較したグラフをそれぞれ図7，8に示す．

図7，8から，パターンの個数が同じであれば，パターンの長さが長くなると，MBM法でマッチングマシンを構成する時間が増大することがわかる．一方，動的構成法でマッチングマシンを再構成する時間はあまり増大しない．すなわち，パターンが長くなればなるほど，動的構成法の効果が増大することがわかる．

このことと前節の実験結果から，パターンの個数が多くなればなるほど，またパターンの長さが長くなればなるほど動的構成法の効果が増大することがわかる．

また，パターンの長さが短く，パターンの個数が少ない場合でも動的構成法が有効であることもわかる．

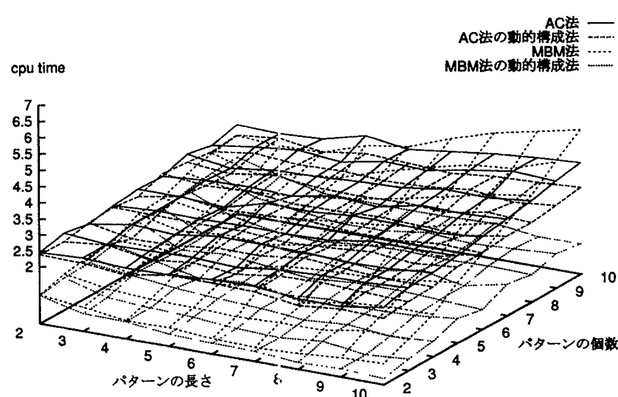



図9 AC法とMBM法の照合時間
Fig. 9 Comparison with AC method including matching time (add).

### 5.3 AC法との照合時間も含めた比較

これまでの実験では，MBM法と動的構成法におけるマッチングマシンの構成時間だけの比較を行った．今度は，マッチングマシンの構成時間だけでなく，実際にテキストとの照合時間も含めて，AC法やAC法の動的構成法とも比較する実験を行った．

テキストは，英単語を連接した長さ10000の文字列を用い，パターンは，テキストファイルの文字と同じ出現確率でランダムに作成した文字列を用いた．また，実験値は，それぞれ試行を100回繰り返し，その平均時間とした．図9に追加処理の場合の実験結果を示す．

AC法とAC法の動的構成法においては，処理時間があまり増大しない．これに対して，MBM法はパターンの長さが長く，かつ個数が多くなった場合，マッチングマシンの構成時間が増大し，それを含めた照合時間は，AC法よりも効率の悪いものとなってしまう．一方，このような場合でも，MBM法の動的構成法を用いた場合には，処理時間がそれほど増大せず，一番高速であることがわかる．

なお，削除処理においても同様の結果を得た．

## 6. むすび

本論文では，複数パターンの文字列照合アルゴリズムであるMBM法において，1度テキストと照合した後パターンのうちのいくつかを更新（追加，削除）して再度照合する場合に，前のマッチングマシンを局所的に変更して再構成するアルゴリズムを提案した．

実験の結果，10個から1500個のパターン集合に対して，MBM法でマッチングマシンを再構成する時間



NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers



に比べて，パターンを一つ追加する場合で約 2.7 倍～78.2 倍，削除する場合で約 2.3 倍～30.3 倍高速になることが確認された．また本提案アルゴリズムは，パターンの長さが長く，個数が多い場合にその有効性が顕著に現れただけでなく，パターンの長さが短く，個数が少ない場合でも有効であることが確認された．更に，本提案アルゴリズムを用いてマッチングマシンを動的に構成する方法は，照合時間も含めると，既に提案されている AC 法の動的構成法よりも高速であることが確認された．

今後は，実際のデータを用いて，パターン集合のうちのどれだけを更新した場合に動的構成法の効果が現れるのかを調べ，動的なパターン集合に対して，動的構成法で再構成するか新規のパターン集合として構成し直すかを自動的に選択するアルゴリズムを開発する予定である．

**謝辞** 査読者の先生方から多くの有益な御助言を頂いた．ここに記して謝意を表する．

## 文献


[1] A.V. Aho and M.J. Corasick, "Efficient string matching: An aid to bibliographic search," Commun. ACM, vol.6, no.18, pp.333–340, 1975.

[2] 津田和彦，入口浩一，青江順一，"ストリングパターンマッチングマシンの動的構成法，" 信学論（D-I），vol.J77-D-I, no.4, pp.282–289, April 1994.

[3] R.S. Boyer and J.S. Moore, "A fast string searching algorithm," Commun. ACM, vol.10, no.20, pp.762–772, 1977.

[4] 浦谷則好，"高速な複数文字列照合アルゴリズム：FAST，" 情処学論，vol.30, no.9, pp.1119–1125, 1989.

[5] 浦谷則好，"FAST 法の効率の推定と長パターン時のふるまい，" 情処学論，vol.32, no.9, pp.1073–1079, 1991.

[6] N. Uratani and M. Takeda, "A fast string-searching algorithm for multiple patterns," Info. Proc. Manag., vol.29, no.6, pp.775–791, 1993.

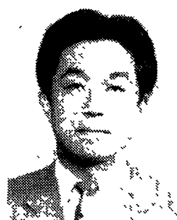
[7] J.-J. Fan and K.-Y. Su, "An efficient algorithm for matching multiple patterns," IEEE Trans. on knowledge and data engineering, vol.5, no.2, pp.339–351, 1993.

[8] D.E. Knuth, J.H. Morris, and V.R. Pratt, "Fast pattern matching in strings," SIAM J. Comput., vol.2, no.6, pp.323–350, 1977.

[9] G.De V. Smit, "A comparison of three string matching algorithms," Software-Practice and Experience, 12, pp.57–66, 1982.

[10] 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.


（平成 8 年 12 月 2 日受付）

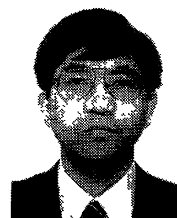



**広瀬　貞樹**　（正員）

昭 49 富山大・工・電子卒．昭 51 東北大大学院工学研究科修士課程了．昭 55 同博士課程了．同年富士通研究所入社．昭 59 神奈川大学工学部助教授．平 1 富山大学工学部助教授．工博．オートマトン・形式言語理論，アルゴリズム理論などに関する研究に従事．情報処理学会会員．

**小栗　伸幸**

平 6 富山大・工・電子情報卒．平 8 同大大学院工学研究科博士前期課程了．同年富士ソフト ABC（株）入社．アルゴリズム理論に関する研究に従事．



**吉澤　寿夫**　（正員）

昭 46 富山大・工・電気卒．同年日本ミニコンピュータ（株）入社．電算機周辺機器のインタフェース等の設計に従事．昭 50 富山大学工学部文部技官．平 5 同助手．有限要素法を用いた電界，電磁界の解析に関する研究に従事．日本シミュレーション学会会員．

**山淵　龍夫**　（正員）

昭 40 東北大・工・通信卒．昭 47 同大大学院工学研究科博士課程了．同年富山大学工学部電気工学科助手．昭 54 同講師．昭 58 同助教授．平 1 同電子情報工学科助教授．平 2 同教授．工博．有限要素法を用いた圧電振動子や音場などの解析に関する研究に従事．IEEE，電気学会，日本音響学会，日本シミュレーション学会，情報処理学会各会員．




NII-Electronic Library Service