転置ファイルとビット配列を用いた高速文字列あいまい照合アルゴリズム

下村秀樹[†] 福島俊一[†]

本稿では，文字ベースの転置ファイルとビット配列を用いた高速文字列あいまい照合アルゴリズムを提案し，実験によってその有効性を示す．文字列あいまい照合の代表的アルゴリズムに，Wu と Manber のアルゴリズム（以下「*Bit-Array* アルゴリズム」）がある．これは，照合状況を示すビット配列に単純なシフト演算と論理演算を逐次的に施すことで，長さ n のテキストと長さ m の検索文字列に対する k 回以内の挿入/削除/置換を許した文字列あいまい照合を，ワード長が m ビット以上の計算機において $O(nk)$ の計算量で実行できるという優れた特長を持つ．しかし，照合のためにテキストの全文字を参照する必要があり，そのままで大量テキストのあいまい検索に適した方法とはいえない．これに対して本稿では，*Bit-Array* アルゴリズムの考え方を，大量の固定的テキストの検索に有効な文字ベースの転置ファイルと組み合わせた，「スキップ型 *Bit-Array* アルゴリズム（*SBA* アルゴリズム）」を提案する．*SBA* アルゴリズムは転置ファイルを参照することで，検索文字列の構成文字以外のテキスト位置でのビット配列計算をスキップする．照合処理の計算量は $O(kmn/|\Sigma|)$ であるが（$|\Sigma|$ は文字セット規模），$|\Sigma|$ に比べて m が十分小さければ *Bit-Array* アルゴリズムよりも高速な検索速度が得られる．1000 万文字の日本語テキストに対する検索実験（$2 \leq m \leq 10$, $k \leq m$）では，従来の *Bit-Array* アルゴリズムと比べて，照合処理時間が約 1/28〜約 1/178 に短縮され，*SBA* アルゴリズムの優位性を確認できた．

A Fast Approximate String Matching Algorithm Using an Inverted File and Bit-arrays

Hideki Shimomura[†] and Toshikazu Fukushima[†]

This paper proposes a new approximate string matching (ASM) algorithm using a character-base inverted file and bit-arrays, and also describes experimental results to show its effectiveness. One of the well-known ASM algorithms is Wu and Manber's (*Bit-Array* algorithm), which applies bit-shift and bit-AND/OR operations on bit-arrays representing matching situations. It is implemented on a computer with more than m bits for the word size, in time of $O(nk)$, where n is the text length, m is the pattern length, and mismatches are allowed within k times. However, it is not suitable for some applications searching a large amount of text, because it has to scan all characters in the text at least once. The new algorithm (*SBA* algorithm), proposed in this paper, solves the problem by combining the idea of the *Bit-Array* algorithm with a character-based inverted file method. It requires $O(kmn/|\Sigma|)$ time for ASM, where $|\Sigma|$ is the size of a character set. When m is smaller enough than $|\Sigma|$, the *SBA* algorithm runs faster than the *Bit-Array* algorithm, because it scans only the index data corresponding to the characters which the pattern consists of. The experimental results for 10 million character Japanese text with $2 \leq m \leq 10$ and $k \leq m$ show that the *SBA* algorithm has the advantage of its matching speed $28 \sim 178$ times faster than the *Bit-Array* algorithm.

1. はじめに

文字列と文字列を，何文字かの不一致を許容して一致と見なす「あいまい照合」は，古くからテキスト処理の重要な技術として認知されている[1)〜4)]．特に，不一致を許容しない「完全照合」では検索洩れが発生してしまうような応用で，このあいまい照合の技術が求められている．たとえば，OCR 処理結果のテキストを全文検索可能な文書画像ファイリングシステム[5),6)]や，つづりミスや表記のゆれを自動検出する英文スペルチェッカ[7)]/日本語文校正支援システム[8)]など，誤りを含むテキストを扱う応用のために必須の技術である．また，検索対象テキストに誤りが含まれていなくとも，

† 日本電気株式会社ヒューマンメディア研究所
Human Media Research Laboratories, NEC Corporation

検索したい文字列の側にあいまい性や不確かさがあるならば☆1，通常の情報検索システムにおいても，あいまい照合の要求が生じてくる．一般に検索精度は，適合率（過剰検索の少なさ）と再現率（検索洩れの少なさ）とのトレードオフ関係のうえで議論される．あいまい照合は再現率を向上させるための技術であり，そのトレードオフとして，適合率は低下せざるをえない側面を持つ．このことから，あいまい照合が生きるのは，適合率を多少犠牲にしても再現率を重視する上記のような応用分野/利用場面だということができる．

あいまい照合は完全照合に比べて計算コストがかかる．しかしながら，最近の計算機の記憶容量・処理能力の飛躍的向上や，インターネット/イントラネットを介して流通・蓄積される文書量の増大を背景として，あいまい照合/検索の応用分野においても，大量テキストに対する高速処理の要求が特に高まってきている．

以下本稿では，文字列のあいまい検索/照合について論じるが，検索の対象となる文字列を「テキスト」，検索したい文字列を「検索文字列」と呼ぶ．また，文字列と文字列の一致を検出する処理を「照合」，テキスト中の検索文字列を検出するための処理全体を「検索」と呼ぶ．したがって,「照合」は「検索」の一部と考える．

文字列あいまい照合の優れたアルゴリズムとして，単純なビットシフト演算とビット論理演算を，照合状況を示すビット配列（bit-array）に漸化的に適用するアルゴリズムが提案されている[3]☆2．以下，本稿ではこれを「*Bit-Array* アルゴリズム」と呼ぶ☆3．*Bit-Array* アルゴリズムは，計算機の1ワード（演算を1命令で行えるビット数）を M ビットとしたとき，検索文字列の長さ m（文字数）が $m \leq M$ であるなら，長さ n のテキストに対する k 回以内の文字の挿入/削除/置換を許したあいまい照合を，$O(nk)$ の計算量で実行できるという優れた特長を持つ．

しかし，*Bit-Array* アルゴリズムは，テキストのすべての文字を1回走査することを前提としている．このため，テキストが大量となった場合に，照合処理のためのデータ参照量が非常に多くなるという問題がある．もちろん，データ参照量はテキストの量に比例するものであるが，テキストの絶対量が多ければ，結果としてアプリケーションから見て許容できない検索処理時間となることもある．また，テキストが大量になりそれを2次記憶に置いた場合には，照合処理のために，テキストをメインメモリに転送する時間も多く要してしまう．したがって，現実の応用を考えるとき，*Bit-Array* アルゴリズムは，そのままで大量テキストのあいまい検索に適した手法であるとはいえない．

一方，更新頻度の低い大量テキストを検索する場合の高速化技術の1つとして，転置ファイルを用いる方式がある[9)~11)]．テキストにおけるある文字の出現位置のリストを「文字インデクス」と呼ぶものとし，この文字インデクスの集合体として転置ファイルを構成する．検索処理は，転置ファイルの中から検索文字列に含まれる文字に対する文字インデクスだけを取得し，その取得した文字インデクス間の位置整合性をチェックすることで行う．この方式では，検索対象テキストの一部（検索文字列に含まれる文字に該当する部分）の情報だけしか処理する必要がないので，照合処理におけるデータ参照量が少なく，高速な検索を行えるという利点がある．

しかし，転置ファイルを使った照合処理において，一部文字の不一致を許したあいまい照合を，効率良く行える有効なアルゴリズムは提案されていない．従来技術を組み合わせて，転置ファイルを使ったあいまい検索を行うとすると，たとえば，検索文字列に対し，それと一致と見なせる文字列（つまり，何文字かが不一致である文字列）をすべて作成し，その文字列で完全照合による検索を繰返し行う方法が考えられる．しかし，不一致の状況には，文字の「挿入」「削除」「置換」など，いくつかのパターンがある．それらの状況とその発生位置の組合せで作られる文字列の数は，検索文字列長と許容する不一致の回数に従って爆発的に増加してしまうので，この方法は現実的でない☆4．また別の方法として，転置ファイルより得られた文字インデクスから元のテキストを擬似的に復元して（テキストで検索文字列中の文字が出現しない位置には，どの文字とも異なる特殊な文字を入れておいて），オリジナルの *Bit-Array* アルゴリズムを適用することも考えられる．しかし，この方式では，照合のためにテキストの全文字数分のデータを参照せねばならず，転置ファイルを使用するメリットが，ほとんどなくなってしまう．

そこで，*Bit-Array* アルゴリズムの考え方を，文字

☆1 送り仮名の付け方など単語表記法にゆれのあるケースをあいまい性，単語つづりをうろ覚えのようなケースを不確かさと，ここではいっている．

☆2 これは，完全照合アルゴリズムである Shift-OR アルゴリズム[4)]の拡張である．

☆3 文献3) ではアルゴリズムに特別な名称が付けられていない．計算方法の特徴から，本稿ではこのように呼ぶ．

☆4 任意の文字との一致を許容する正規表現を導入しても，作られる文字列の数は膨大なものとなる．また，正規表現を許容する照合処理にも計算量を要してしまう．

ベースの転置ファイルの手法と組み合わせることで，大量のテキストに対する効率の良い文字列あいまい照合の実現を検討した．提案するアルゴリズムは，転置ファイルから得られた文字インデクスにより，テキストを復元して走査する方法をベースとして，走査の際に不要な計算をスキップするというアイデアに基づく☆．以下，この提案アルゴリズムを「スキップ型 *Bit-Array* アルゴリズム（*SBA* アルゴリズム）」と呼ぶ．*SBA* アルゴリズムによれば，検索文字列に含まれる文字に対する文字インデクスを，整列してただ1回走査することにより，k 回以内の文字の挿入/削除/置換を許すあいまい照合を実施することができる．その結果，転置ファイルを使用するメリットを生かし，*Bit-Array* アルゴリズムよりも実用上高速な文字列あいまい照合を可能とする．

以下，2 章では，議論に使う記号の定義とともに，オリジナルの *Bit-Array* アルゴリズムを述べ，3 章では，転置ファイルを用いたテキスト検索について説明する．4 章では，*Bit-Array* アルゴリズムの考え方を転置ファイルと組み合わせるためのアイデアとその計算式，および導出された計算式に基づく *SBA* アルゴリズムの提案を行う．5 節では 1000 万文字（約 20 M バイト）の日本語テキストの検索を例に，従来の *Bit-Array* アルゴリズムに基づく全文走査型のあいまい検索と，*SBA* アルゴリズムに基づく転置ファイルを使ったあいまい検索の処理速度を比較した実験を報告する．

2. ビット配列ベースの文字列あいまい照合アルゴリズム——*Bit-Array* アルゴリズム

本章では，本稿での議論に用いる記号の定義と，オリジナルの *Bit-Array* アルゴリズム[3)]について述べる．

2.1 準　備

長さ m の検索文字列 P が与えられたとき，長さ n のテキスト T における P の出現位置を，k 回以内の文字の挿入/削除/置換☆☆を許容してすべて検索する問題を考える．以下に，議論に使う記号を準備する．

Σ：テキストあるいは検索文字列に使われる文字の集合．$|\Sigma|$ は集合 Σ に属する文字の数を示す．以下，文字および文字列は，'a'，'abc' のように表現する．

Ω_m：長さ m のビット配列の集合．以下，ビット配列は $\langle 10101 \rangle$ のように表現する（この例は $m=5$）．また，ビット配列は左側を先頭とする．

$T = t_1 t_2 t_3 \ldots t_n$ $(t_i \in \Sigma)$：長さ n の検索対象のテキスト．

$P = p_1 p_2 p_3 \ldots p_m$ $(p_i \in \Sigma)$：長さ m の検索文字列．

k：照合時に許容する挿入/削除/置換の回数．以下，「不一致文字数」と呼ぶ．

B_f $(\in \Omega_m, 0 \le f \le m)$：先頭の f ビットが 1 で残りが 0 であるような長さ m のビット配列．たとえば，$m=5, f=3$ なら $B_3 = \langle 11100 \rangle$ となる．

$Sft(\omega)$ $(\in \Omega_m, \omega \in \Omega_m)$：長さ m のビット配列 ω を右に 1 ビットシフトする演算．空いたビットには 1 を入れる．たとえば，$Sft(\langle 01110 \rangle) = \langle 10111 \rangle$ である．

S_c $(\in \Omega_m, c \in \Sigma)$：長さ m のビット配列で，検索文字列 P における文字 c が存在した位置に対応するビットが 1 で残りが 0 となる．たとえば，$P=$'ababc' に対して，$S_{\text{'a'}} = \langle 10100 \rangle$，$S_{\text{'b'}} = \langle 01010 \rangle$，$S_{\text{'c'}} = \langle 00001 \rangle$，$S_{\text{'d'}} = \langle 00000 \rangle$，である．

R_i^d $(\in \Omega_m, 0 \le i \le n, 0 \le d \le k)$：長さ m のビット配列で，T の i 文字目における T と P の照合状況を示す．以下,「一致状態ビット配列」と呼ぶ．R_i^d の先頭から j ビット目が 1 ならば，T の i 番目の位置で，P の 1 番目（左側）から j 番目までの部分文字列と，d 文字以内の不一致を許して照合が成功していることを示す．P の長さは m なので，R_i^d の m ビット目が 1 であれば，d 文字以内の不一致を許して検索文字列 P が見つかったことを意味する．たとえば，$m=5$ のとき，$R_{10}^1 = \langle 11001 \rangle$ であれば,「T の 10 文字目の位置で，P の 1 文字目，2 文字目，5 文字目までと，1 文字以内の不一致を許して照合が成功している」ことを示す．なお，$i=0$ に該当する R_0^d は，照合処理のためのダミーである．

AND：ビット配列の論理積演算子．

OR：ビット配列の論理和演算子．

2.2 アルゴリズムの形式的記述

Bit-Array アルゴリズムでの照合処理は，一致状態ビット配列 R_i^d に対する漸化式の計算を繰り返すことによって行われる．R_i^d の漸化式は次のとおりである．

☆ *Bit-Array* アルゴリズムで照合に関係ない箇所をスキップしたい，という考えと，転置ファイルを用いた検索において文字インデクスが得られない箇所を処理しない，という考えが非常に近い関係にあるとの直感から，両者の融合を試みた．

☆☆「挿入」はテキストに余分な文字が含まれること,「削除」はテキストから文字が欠落すること,「置換」はテキストの文字が別の文字に変わっていること（前後の文字の交換ではない）を意味する．

T='adeabcddffabefcaefddabaca'
P='abaca'

i	t_i	S_{t_i}	R_i^0	R_i^1	R_i^2
0	-	-	00000	10000	11000
1	a	10101	10000	11000	11100
2	d	00000	00000	11000	11100
3	e	00000	00000	10000	11100
4	a	10101	10000	11000	11100
5	b	01000	01000	11100	11110
6	c	00010	00000	11110	11111*
7	d	00000	00000	10000	11111*
8	d	00000	00000	10000	11000
9	f	00000	00000	10000	11000
10	f	00000	00000	10000	11000
11	a	10101	10000	11000	11100
12	b	01000	01000	11100	11110
13	e	00000	00000	11100	11110
14	f	00000	00000	10000	11110
15	c	00010	00000	10000	11010
16	a	10101	10000	11000	11101*
17	e	00000	00000	11000	11100
18	f	00000	00000	10000	11100
19	d	00000	00000	10000	11000
20	d	00000	00000	10000	11000
21	a	10101	10000	11000	11100
22	b	01000	01000	11100	11110
23	a	10101	10100	11110	11111*
24	c	00010	00010	11111*	11111*
25	a	10101	10001*	11111*	11111*

図 1 *Bit-Array* アルゴリズムによる照合処理例（$k=2$）
Fig. 1 An example of the matching process with *Bit-Array* algorithm ($k=2$).

$$R_i^d = \begin{cases} B_d & i=0 \\ Sft(R_{i-1}^0) \\ \quad AND\ S_{t_i} & i>0, d=0 \\ \left(\begin{array}{l} (Sft(R_{i-1}^d) \\ \quad AND\ S_{t_i}) \\ \quad OR\ R_{i-1}^{d-1} \\ \quad OR\ Sft(R_{i-1}^{d-1}) \\ \quad OR\ Sft(R_i^{d-1}) \end{array} \right) & i>0, d>0 \end{cases} \quad (1)$$

この式に従い，i および d の小さい順に R_i^d を逐次計算する．Σ に属するすべての文字 c に対する S_c は，漸化式計算の前に P から作成しておく．前にも述べたが，検索文字列 P の長さを m とすると，R_i^d の m ビット目が1になったことにより，テキスト T の i 番目の文字位置で d 文字以内の不一致を許した P を検出できる．

2.3 照合処理例

照合処理の例を図 **1** に示す．不一致文字数 k は 2 である．

R_i^d の値に '*' の付いた位置が，検索に成功したことを示している．$i=6$ の位置では，$P=$'abaca' に対して 'abc' が「3 番目の 'a' および 5 番目の 'a' の削除」で，$i=7$ の位置では，「3 番目の 'a' の削除および

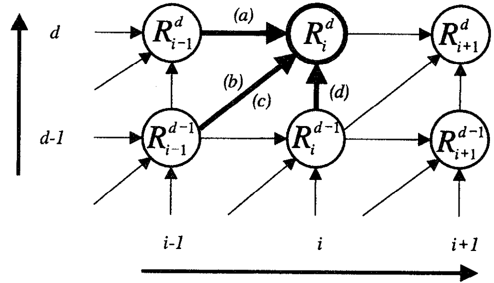


図 2 *Bit-Array* アルゴリズムおける照合状況の遷移
Fig. 2 Matching state transition in *Bit-Array* algorithm.

5 番目の 'a' の置換」という状況で検索されている☆．$i=16$ の位置では，'abaca' に対して 'abefca' が「1 文字の置換と 1 文字の挿入（3 文字目の 'a' に対して 'e' および 'f' が対応）」によって検索されている☆☆．この他では，$i=23$ で 2 文字以内の不一致，$i=24$ で 1 文字以内の不一致，$i=25$ で 0 文字以内の不一致（すなわち，完全一致）で文字列が検索されている．

2.4 アルゴリズムの直感的理解

アルゴリズムの詳細や論理的裏付けは，文献 3) に詳しい．ここでは，アルゴリズムを直感的に理解するために，図を参考に漸化式を説明する．

式 (1) において，$i=0$ のケースは初期設定である，また $d=0$ のケースは $d>0$ の特殊ケースと考えられるので，$d>0$ のケースを理解すれば十分である．この式は，隣接する 3 つの一致状態ビット配列から作られる，4 つの項の論理和となっている．模式的には，図 **2** のような関係となっている．この (a)〜(d) の項は，それぞれ前の状態から現在の状態に至る際の要因が，文字の一致/挿入/置換/削除の各々であることに対応している．式ではそれらの論理和をとることにより，少なくともどれか 1 つの状態が起こっていれば照合成功を示すビットに 1 が立ち，部分的なあいまい照合が成功している状態を表現する．次に，(a)〜(d) が示す内容を説明する．

今，T の i 文字目までと P の j 文字目までが d 文

☆ ただし，この両者は解釈こそ違え T 中の同じ文字列を検出したものであり，片方が検索されれば十分である．すなわち，$i=6$ の時点で，$i=7$ の状況は予測可能である．

☆☆ 挿入/削除/置換の発生の解釈は，一意ではない．この例でも，(1)「'a' が 'e' に置換され，かつ 'f' が挿入」，(2)「'e' が挿入され，かつ 'a' が 'f' に置換」のように 2 通りの解釈ができる．

字以内の不一致を許容して一致と見なせる状態，すなわち，R_i^d の j ビット目が1となる条件を考えてみる．前記の4つの項が，それぞれ次のような条件を意味している．

(a) $Sft(R_{i-1}^d)$ *AND* S_{t_i}：T の $i-1$ の位置までで，P の $j-1$ 文字目までと d 文字以内の不一致で，i の位置の文字が p_j と一致の場合

(b) R_{i-1}^{d-1}：T の $i-1$ の位置までで，P の j 文字目までと $d-1$ 文字以内の不一致で，i の位置に挿入が起こった場合

(c) $Sft(R_{i-1}^{d-1})$：T の $i-1$ の位置までで，P の $j-1$ 文字目までと $d-1$ 文字以内の不一致で，i の位置で置換が起こった場合

(d) $Sft(R_i^{d-1})$：T の i の位置で，P の $j-1$ 文字目までと $d-1$ 文字以内の不一致で，さらに i の位置で削除が起こった場合

2.5 *Bit-Array* アルゴリズムの特長

Bit-Array アルゴリズムは，式(1)を計算しながら長さ n のテキストを1回走査することにより，k 文字以内の不一致（挿入/削除/置換）を許した照合が可能である．計算機の1ワード（演算を1命令で行えるビット数）を M ビットとしたとき検索文字列の長さ m（文字数）が $m \leq M$ であるなら，m 文字分のビット演算は1命令で実行できるので，計算量は m によらない☆．このことから照合処理の計算量は，ビット配列 R_i^d の計算量 $O(nk)$ に P から S_c を作成する計算量 $O(|\Sigma|)$ を加えたものとなるが，n が大きければ通常後者の時間は無視できる．したがって，あいまい照合を許しながら，処理時間がテキストの長さおよび不一致文字数にほぼ比例した時間に抑えられるという優れた特徴を持つ．また，漸化式の演算量が非常に小さく，高速な処理が可能である．

3. 転置ファイルを用いたテキスト検索

本章では，転置ファイルを用いた，テキスト検索の高速化について述べる．

転置ファイルを広義にとらえるなら，「照合の処理単位となるものが元のテキスト中のどの位置にあったかを示すリストの集合」ということができる．英語では単語単位の検索が主流であるため，ある単語がテキスト中のどの位置にあったかというリストを転置ファイルにすることが多い[11),12)]．一方，日本語の場合は単語区切りが難しいため，文字単独，あるいは複数の文字を組み合わせて照合処理単位を構成し，転置ファイルを作成するのが一般的である[9),10)]．圧縮や不要語除去等の工夫を特にしなければ，転置ファイルの全体容量は，一般に元のテキストよりも大きくなる．また，転置ファイルが大量になった場合，ディスクなどの2次記憶に格納されるのが普通である．

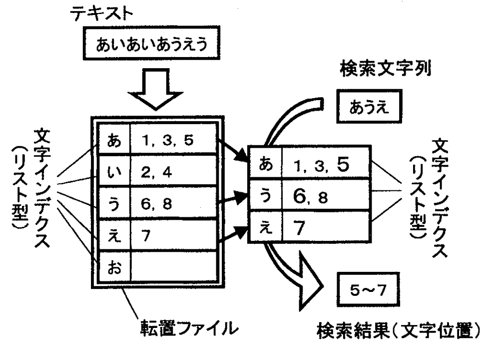


図3 転置ファイルを使った検索処理
Fig. 3 An example of the search process using an inverted file.

このように転置ファイルにはその目的に応じて様々な形態が考えられるが，以下本稿では文字単位の照合を議論するので，文字単位の転置ファイルを考える．このとき，ある文字の出現位置を表す情報を「文字インデクス」と呼ぶ．すなわち，ここでの転置ファイルは文字インデクスの集合体として構成されることになる．図3に例を示す．

文字インデクスの形式には「リスト型」と「ペア型」がありうる．図3の例では，文字に対応付けてその出現位置のリストを管理する「リスト型」の文字インデクスを示している．ある文字の出現位置の一覧がすぐに得られる構造になっていることから，通常はこのリスト型で転置ファイルを実装するケースが多い．それに対して，(文字，位置）のように，文字と位置の対の形式で表現したものが「ペア型」である．リスト型とペア型は，相互に変換可能であり，行おうとする処理に適した形式を選択すればよい☆☆．たとえば，図3のリスト型をペア型に直すと，(あ,1)(い,2)(あ,3)(い,4)$\cdots$ のようになる．

テキストに対する転置ファイルは検索前にあらかじめ作っておき，検索時には，転置ファイルから検索文字列に含まれる文字に該当する文字インデクスだけを取得し，その位置関係の整合をチェックすることで照合を実施する．

☆ 市販の一般的な計算機では1ワード32ビット，64ビットなどである．

☆☆ たとえば，本稿で提案するアルゴリズムでは，ペア型に対して処理を行う．

転置ファイルを使った検索処理では，テキスト中の文字のうち，照合処理に意味のある，すなわち検索文字列に現れた文字の情報だけしか処理しないので，テキスト全体を参照するのに比べて，高速になるという利点がある．この利点は，特に，検索文字列に対して得られる文字インデクスの量が，元のテキストに比べて少ないケース，たとえば，日本語のように字種が多いテキストの検索の場合に，顕著である．反面，事前に転置ファイルを作成しておく必要があるという制約も生じるが，データの更新頻度に比べて検索頻度の方が圧倒的に多い場合には，特に大きな問題とはならない．

4. 転置ファイルと組み合わせた *Bit-Array* アルゴリズムの拡張——*SBA* アルゴリズム

本章では，*Bit-Array* アルゴリズムの考え方を文字ベースの転置ファイルと組み合わせた *SBA* アルゴリズムについて，そのアイデア，*Bit-Array* アルゴリズムからの漸化式の展開，転置ファイルへの適用方法を提案する．

4.1 拡張のアイデア

転置ファイルを使った検索において文字列照合をする場合，文字インデクスは元のテキストに対して飛び飛び，つまり不連続に得られる．もちろん，この特徴が照合に不要なデータ参照を減らすのであるが，この不連続性のために *Bit-Array* アルゴリズムをそのまま適用することができない．

強引に *Bit-Array* アルゴリズムを適用することを考えると，図4のような方式を比較的容易に思い付く．すなわち，元のテキストで文字インデクスが獲得されなかった位置には集合 Σ に属さない（したがって照合処理でどの文字とも一致しない）適当な文字を割り当て，元のテキストを擬似的に復元して *Bit-Array* アルゴリズムを適用するものである．テキストや転置ファイルを2次記憶に置くと想定した場合，メインメモリへのデータ転送量を削減できるので，これでもある程度の検索速度の向上が期待できる．しかし，照合時間は従来の *Bit-Array* アルゴリズムと変わらず，転置ファイルを使うことによる照合処理自身の高速化は行えていない．

そこで図5のように，文字インデクスが得られない部分の計算をできるだけスキップし，得られた文字インデクスだけを1回走査することで照合処理ができないかと考えた．

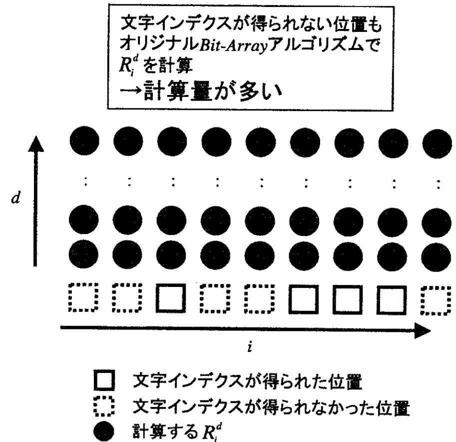


図4 転置ファイルと *Bit-Array* アルゴリズムによる照合処理
Fig. 4 The matching process using a character-based inverted file and *Bit-Array* algorithm.

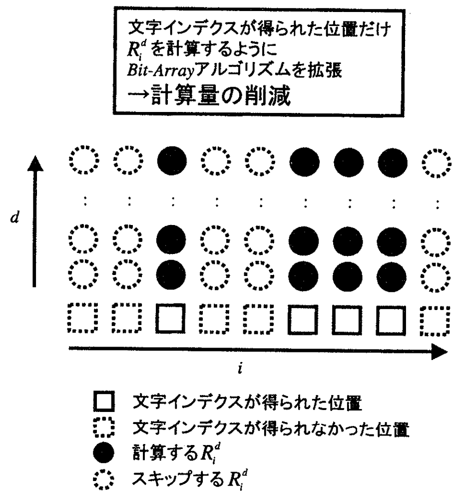


図5 一致状態ビット配列の計算省略のアイデア
Fig. 5 The main idea for reducing calculation on bit arrays.

4.2 漸化式の展開

今，テキストの位置 x で文字インデクスが得られ，$(x+1)$〜$(x+w)$ の区間 $(w>0)$ で得られず，$(x+w+1)$ でまた得られたという状況を仮定する．元のテキストで，文字インデクスが得られなかった位置 $(x+1 \le i \le x+w)$ では，$S_{t_i}=B_0$ と考えてよい．このとき R_{x+w+1}^d を R_x^d から，途中の計算をできるだけ省略して直接的に計算できるアルゴリズムを導こうというのが，前節に述べたアイデアの基本である．

そこでまず，オリジナルの *Bit-Array* アルゴリズムの漸化式である式(1)について，上記条件に該当するときに区間 $(x+1)$〜$(x+w-1)$ の一致状態ビット配列の計算をできるだけ省略して，R_{x+w}^d を計算する

方法を検討した．R^d_{x+w+1} の直前の一致状態ビット配列である R^d_{x+w} が求められれば，そこから R^d_{x+w+1} は，式 (1) によって容易に計算できる．導出過程は付録に示すが，次式が導かれた．

$$R^d_{x+w} = \begin{cases} F_w(R^{d-w}_x) & (d \geq w) \\ B_d & (d < w) \end{cases}$$
$$F(\omega) = \omega \ OR \ Sft(\omega) \quad (2)$$
$$F_y(\omega) = \underbrace{F(F(F(\ldots F(\omega)\ldots)))}_{y\ times}$$

この式に従えば，オリジナルの *Bit-Array* アルゴリズムに比べ，シフトと論理和演算の回数を大幅に減少させ，R^d_{x+w} を計算することができる．

4.3 計算省略の直感的理解

図に従って，式 (2) の直感的な理解を試みる．**図 6** は，式 (2) による計算省略のイメージ（不一致文字数 $k = 4, w = 3$ の例）である．R^d_{x+4} を求めるための R^d_{x+3} を計算する際に，計算を大幅に省略している．

図から，R^d_{x+3} の計算において，左斜め下の情報が演算 F_y で伝えられていることが分かる．また，情報の起点が R^0_{x+3-d} $(1 \leq d < 3)$ である場合には，単に B_d となる．このような簡略化が可能となるのは，計算の省略を考える区間で $S_{t_i} = B_0$ であること，そして，*Bit-Array* アルゴリズムが，不一致の発生の状況（挿入/置換/削除）の区別をしないことによる．

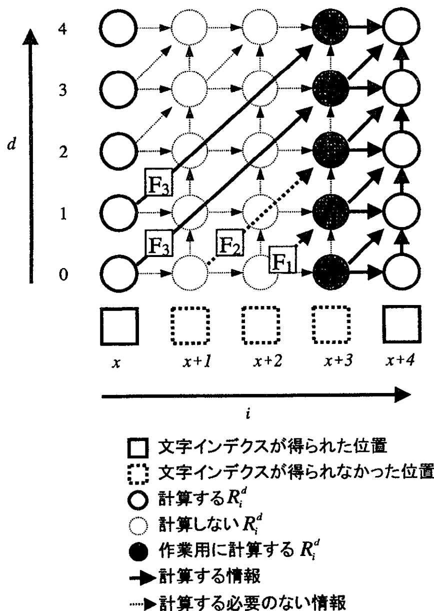


図 6 式 (2) に基づく計算省略の例（$k = 4, w = 3$ の場合）
Fig. 6 An example of reducing calculation by the formula (2) (in case of $k = 4, w = 3$).

まず，$S_{t_i} = B_0$ から，この区間で文字の一致はないので，文字の一致に対応する項，つまり図中の右向きの矢印を考える必要がなくなる．次に，やはり $S_{t_i} = B_0$ から $R^0_i = B_0$ $(x + 1 \leq i \leq x + 3)$ となる．これらを起点とする情報は，文字の削除が起こったケースとして図中の上向き矢印で Sft 演算を通して伝播するが，結局 R^d_i に対しては B_d が論理和演算により加わるだけである．しかし，それと同じ値は，R^d_i に論理和演算で加えられる $Sft(R^{d-1}_{i-1})$ にも含まれており，上向き矢印を無視しても結果は変わらない．これは，*Bit-Array* アルゴリズムが不一致の状況として削除なのか置換なのかを区別せず，その回数だけをカウントするアルゴリズムだからである．結局，左斜め下からの情報の伝達だけを考えればよく，それを順次に適用すれば，図のようになる．

その結果，図 6 に示した点線に対応する演算を行う必要がなくなるので，計算量を大幅に削減できる．また，伝播の元が $d = 0$ の行である場合は実際にビット配列の演算を行う必要はなく，先頭の何ビットかが 1 の値を埋め込むだけでよいので，さらに処理量が削減できる．

4.4 文字インデクスへの適用

次に，上記の計算式をペア型文字インデクスに適用する．その準備として，記号を次のように定義する（**図 7** 参照）．

$idx_i = (pos_i, chr_i)$ $(0 \leq i \leq n')$：ペア型文字インデクスの配列．以下本節では，特に説明がなければ，「文字インデクス」はペア型を指すものとする．n' は得られた文字インデクスの個数とする．1 つの文字インデクスは，テキスト中の該当文字の出現位置 pos_i と，該当文字 chr_i の対から成り，pos_i の小さい順にソートされているとする．したがって，$pos_i - pos_{i-1} > 0$ である．なお idx_0 は，処理で使うダミーである．

R'^d_i $(0 \leq i \leq n', 0 \leq d \leq k)$：各文字インデクスに対する一致状態ビット配列．$idx_i = (pos_i, chr_i)$ のとき，$R'^d_i = R^d_{pos_i}$ を意味するものとする．なお，R'^d_0 は，処理に使うダミーである．

整列した n' 個の文字インデクスのすべてについて，R'^d_i $(= R^d_{pos_i})$ を計算することが照合処理となるので，以下にその手順を示す．

今，idx_{i-1} に対する R'^d_{i-1} $(= R^d_{pos_{(i-1)}})$ が既知であるとして，R'^d_i $(= R^d_{pos_i})$ を求めることを考える．そのためにまず，$R^d_{(pos_i)-1}$ に相当する値を求める．4.2 節で議論したように，これが求まれば式 (1) によって $R^d_{pos_i}$ が容易に求まるからである．

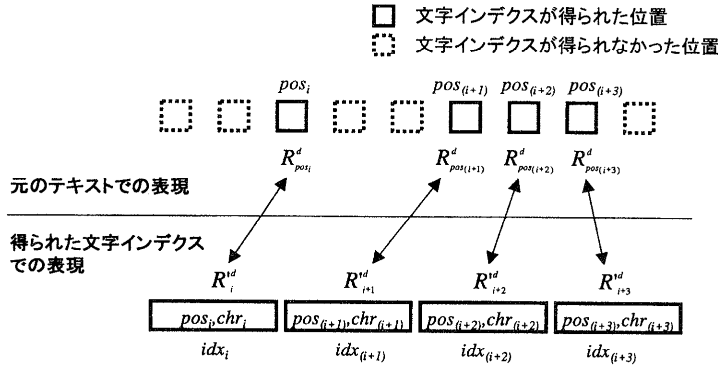


図7 文字インデクスでの R' とテキストでの R との対応
Fig. 7 Correspondence between R' for the character index and R for the text.

$w = pos_i - pos_{(i-1)} - 1$ とおけば，$(pos_i) - 1 = pos_{(i-1)} + w$ である．また $(pos_{(i-1)} + 1)$〜$(pos_{(i-1)} + w)$ の区間では，文字インデクスが得られていないことから，$R^d_{(pos_i)-1}$ の計算には，式 (2) がそのまま適用できる．すなわち，次式となる．

$$R^d_{(pos_i)-1} = R^d_{pos_{(i-1)}+w} = \begin{cases} F_w(R^{d-w}_{pos_{(i-1)}}) = F_w(R'^{d-w}_{i-1}) & (d \geq w) \\ B_d & (d < w) \end{cases}$$
$$w = pos_i - pos_{(i-1)} - 1 \quad (3)$$

後は式 (1) に従って，$R^d_{(pos_i)-1}$ の値と S_{chr_i} から $R^d_{pos_i}$ を計算すれば，それがすなわち R'^d_i を計算したことになる．式 (3) は，式 (2) を導出する際の前提から，$w > 0$ のケースにしか適用できない．しかし，$w = 0$，すなわち文字インデクスがテキストの位置で連続して得られたケースは，$R^d_{(pos_i)-1} = R'^d_{i-1}$ であることを利用して計算を続ければよい．

以上の議論から，*Bit-Array* アルゴリズムの計算式を変形してペア型文字インデクスに適用することで，新しいあいまい照合アルゴリズムが導かれた．これを，「スキップ型 *Bit-Array* アルゴリズム（*SBA* アルゴリズム）」と名付ける．また，その処理を，言語 C の制御構造を使って示す（図 8）．

```
/* SBAアルゴリズム */
/*
文字インデクスはpos_i, chr_iに取得整列済みとする
R^d_tmp : R^d_(pos_i)-1を入れる作業用変数
pos_0 = 0, S_chr_0 = B_0とする
Msk : mビット目が1である長さmのビット配列
*/

/* 準備 */
for (d = 0 ; d <= k ; d++)  R'^d_0 = B_d
/* ビット配列の漸化計算 */
for (i = 1 ; i <= n' ; i++) {
  /* R^d_tmpの計算 */
  w = pos_i - pos_i-1 - 1
  for (d = 0 ; d <= k ; d++) {
    if (w == 0)  R^d_tmp = R'^d_i-1
    else if (d < w)  R^d_tmp = B_d
    else  R^d_tmp = F_w(R'^d-w_i-1)
  }
  /* R^d_tmpを使ったR'^d_iの計算 */
  for (d = 0 ; d <= k ; d++) {
    if (d == 0)  R'^0_i = Sft(R^0_tmp) & S_chr_i
    else  R'^d_i = (Sft(R^d_tmp) & S_chr_i)
                 | R^d-1_tmp | Sft(R^d-1_tmp) | Sft(R'^d-1_i)
    if (R'^d_i & Msk) {/* 一致報告 */}
  }
}
```

図8 *SBA* アルゴリズム
Fig. 8 *SBA* algorithm.

4.5 照合処理例

図 9 に，*SBA* アルゴリズムにおける照合処理過程を示す．図 1 の例と同じテキストを同じ不一致文字数（$k = 2$）で検索した場合を示す．図 1 ではテキスト中のすべての位置 25 箇所でビット配列の計算がなされているのに対して，図 9 では 13 箇所に削減されている．

この例では，$i = 4$，$i = 8$，$i = 11$〜13 で，それぞれ照合に成功している．$i = 4$ は元のテキストの位置で 6，$i = 8$ は 16 に対応しており，図 1 で述べた例と同じ位置で文字列が検出されていることが分かる．$i = 11$〜13 についても同様である．なお，図 1 ではテキストの文字位置 7 でも文字列が検出されているが，これは文字位置 6，つまり $i = 4$ の状況から容易に推測できるものであり，特に検出する必然性はない．提案手法では文字インデクスが取得されない箇所での処理を行わないため，このように一部の冗長な検索結果

T='adeabcddffabefcaefddabaca'
P='abaca'

i	pos_i	chr_i	S_{chr_i}	$R^0_{(pos_i)-1}$	$R^1_{(pos_i)-1}$	$R^2_{(pos_i)-1}$	R'^0_i	R'^1_i	R'^2_i
0	-	-	00000	-----	-----	-----	00000	10000	11000
1	1	a	10101	00000	10000	11000	10000	11000	11100
2	4	a	10101	00000	10000	11100	10000	11000	11100
3	5	b	01000	10000	11000	11100	01000	11100	11110
4	6	c	00010	01000	11100	11110	00000	11110	11111*
5	11	a	10101	00000	10000	11000	10000	11000	11100
6	12	b	01000	10000	11000	11100	01000	11100	11110
7	15	c	00010	00000	10000	11010	00000	10000	11010
8	16	a	10101	00000	10000	11010	10000	11000	11101*
9	21	a	10101	00000	10000	11000	10000	11000	11100
10	22	b	01000	10000	11000	11100	01000	11100	11110
11	23	a	10101	01000	11100	11110	10100	11110	11111*
12	24	c	00010	10100	11110	11111	00010	11111*	11111*
13	25	a	10101	00010	11111	11111	10001*	11111*	11111*

図 9 *SBA* アルゴリズムによる照合処理例
Fig. 9 An example of the matching process using *SBA* algorithm.

が出力されない．

4.6 計 算 量

まず，検索文字列中の 1 文字から取得される文字インデクスの平均個数は，$n/|\Sigma|$ となる．したがって，m 文字の検索文字列では，平均すると $mn/|\Sigma|$ 個となる．不一致文字数 k のときのビット配列のトータルサイズは，$k+1$ に比例するので，$O(kmn/|\Sigma|)$ となる．*Bit-Array* アルゴリズムの場合と同じ前提として，計算機の 1 ワード（演算を 1 命令で行えるビット数）を M ビットとしたとき検索文字列の長さ m（文字数）が $m \leq M$ であると考えるならば，m ビットの演算は m に依存しない時間で実行可能であるから，*SBA* アルゴリズムの照合処理は $O(kmn/|\Sigma|)$ となる．

これを *Bit-Array* アルゴリズムの $O(nk)$ と比較すると，m に依存しない点では *Bit-Array* アルゴリズムの方が優位である．しかし，$1/|\Sigma|$ のファクタがあるため，$|\Sigma|$ に比べて m が十分に小さければ，*SBA* アルゴリズムの方が高速になる．日本語テキストの場合は $|\Sigma| \geq 10^3$ であり，現実の応用ではほとんど $m \leq 10$ と考えられるから，*SBA* アルゴリズムは十分優位性を持つ．

また，検索を行うためには，照合処理の他に，2 次記憶からメインメモリへのデータ転送，リスト型文字インデクスからペア型文字インデクスへの変換，整列が必要である．データ転送とデータ変換の処理量は得られる文字インデクスの数に比例する，すなわち，$O(mn/|\Sigma|)$ である．また，整列の計算量は，文字インデクスの数に $\log m$ をかけた $O(mn \log m/|\Sigma|)$ となる．したがって，m が非常に大きくなれば，整列の計算量が全体の計算量に対して支配的となる可能性はあるが，通常の文字列検索では m は小さいので，大きな問題とはならない．

5. 実験と考察

5.1 実　　験

提案した *SBA* アルゴリズムの効果を調べるために，日本語の約 20M バイトのテキストに対して，検索の実験を行った．プログラムは言語 C で実装し，NEC 製の UNIX ワークステーション EWS4800/470（CPU：R10000，クロック 200 MHz，主記憶 1 GB）で実行した．

実験では，次の 2 つの方式について，検索速度を測定した．

従来方式 べた書きテキストに対するオリジナル *Bit-Array* アルゴリズムによる検索．テキストは 2 次記憶のファイルに保持し，検索時にメモリへ転送する．

提案方式 転置ファイルに対する *SBA* アルゴリズムによる検索．転置ファイル（リスト型）は検索前にあらかじめ作成して 2 次記憶のファイルに保持しておき，検索時に必要な文字インデクスをメモリへ転送してペア型に変換する．

検索対象のテキストは，日本語の特許 1000 万文字（1 文字あたり 2 バイトで約 20 M バイト）である．これに対する転置ファイル容量は約 40 M バイトとなった☆．照合で参照するデータ（従来方式の場合はテキスト，提案方式の場合は検索文字列に対応する文字イ

☆ 転置ファイルは，文字インデクスの検索キーとなる文字テーブルの領域と位置情報群の領域とから成る．前者は 97,008 バイト，後者はテキスト 1000 万文字の各々に対する位置情報を 4 バイトで表現して格納した．なお，この転置ファイルの作成に要した時間は約 140 分である．

表 1 実験に使用した検索文字列の例
Table 1 An example of patterns used in the string search experiment.

長さ	検索文字列の例
2	パン，トス，管理，同時，お茶，苦み
3	デルタ，ホイル，急発進，検出値，小えび，ケイ酸
4	エンジン，ロータリ，正規分布，固定方法，あんパン，陽イオン
5	キーワード，フィンガー，特許明細書，防水防湿性，排水ポンプ，食器洗い機
6	ヒストグラム，プラスチック，音声認識処理，酵素反応温度，1／ｆゆらぎ，冷凍シリンダ
7	シリンダケース，ベルトコンベア，鑑賞魚飼育装置，蛋白質含有食品，エンジン回転数，海苔巻おにぎり
8	ソースプログラム，エンジニアリング，不飽和高級脂肪酸，高圧高温発生装置，アルミニウム合金，動物性混合エキス
9	ベンジルアルコール，ポリアクリルアミド，重金属溶出防止装置，点火系回路遮断装置，反転出力タイミング，ストライクめっき層
10	フェネチルアルコール，エキスパートシステム，食品用防水防湿処理剤，制振材料用粘弾性樹脂，アルカリ土類金属元素，グラファイト製つるぼ

ンデクス）は，すべてファイルからメモリにロードした後に，照合処理を行うようにしている．

検索文字列は，2〜10 文字の各長さについて 15 個ずつを選んだ（各長さとも漢字語 5 個，片仮名語 5 個，交ぜ書き語 5 個という内訳を合わせた）．合計 135 個の検索文字列となる．実験に使用した検索文字列の例を**表 1** に示す．なお，不一致文字数 k は，検索文字列長 m に対して意味を持つ $k < m$ のすべてのケースを測定した．

実験結果を，**表 2** に示す．表の項目は次のとおりである．

m：検索文字列の長さ．

T-size：照合処理のためのファイルからの転送データ量（K バイト）．従来方式の場合はテキスト，提案方式の場合は検索文字列に含まれる文字に対応する文字インデクスの量になる．

T-time：T-size で示したデータを転送するのに要した時間．提案方式では，読み込んだリスト型文字インデクスをペア型文字インデクスに変換する時間も含む．

S-time：データの整列時間．提案方式にのみ必要な時間．

T+S：T-time と S-time の合計．

M-time：照合処理の時間．不一致文字数 k ごとに異なる．

T+S+M：T-time+S-time+M-time の値で検索処理全体の合計時間．

処理時間の単位は ms である．なお，従来方式はアルゴリズムの性質上，検索文字列の長さ m によらず一定の処理時間となる．実験ごとに数%の測定誤差は見られたが，k ごとの全実験の平均値を示す．提案方式の実験結果は，検索文字列の長さごと，すなわち漢字語 5 個，片仮名語 5 個，交ぜ書き語 5 個，合計 15 個の検索文字列による検索での平均値を示す．表の下部には，提案方式を 1 としたときの，従来方式の転送データ量，処理時間の比を示す．転送データ量や処理速度が，従来方式に比べ何倍改善されたかを意味している．

5.2 考　　察

5.1 節の実験の結果から，次のことが分かる．

(1) データ転送量

提案方式のデータ転送量（T-size）は，従来方式と比較して，約 76 分の 1（$m = 2$ のとき）〜約 20 分の 1（$m = 10$ のとき）となった．提案方式で，検索文字列長 m の増加におよそ比例してデータ転送量が増加しているのは，m に比例して文字インデクスを取得する対象の文字数が増加するためである．転置ファイルを用いることで，$m \leq 10$ において，データ転送量（照合時に参照するデータ量）を大きく削減できていることが分かる．

(2) 処理時間

提案方式の処理時間は，従来方式に比較して，以下のように短縮された．

- データ転送時間（T-time）は，約 42 分の 1（$m = 2$ のとき）〜約 8 分の 1（$m = 10$ のとき）に短縮された．
- データ転送から整列までの合計時間（T+S），すなわち照合の直前までの処理時間は，約 23 分の 1（$m = 2$ のとき）〜約 2 分の 1（$m = 10$ のとき）に短縮された．
- 照合処理時間（M-time）は，約 178 分の 1（$m = 2$, $k = 0$ のとき）〜約 28 分の 1（$m = 10$, $k = 9$ のとき）に短縮された．
- データ転送から照合までの全体処理時間（T+S+M）は，約 112 分の 1（$m = 2$, $k = 1$ のとき）〜約 9 分の 1（$m = 10$, $k = 0$ のとき）に短縮

表 2 実験結果
Table 2 Experimental results.

	m	T-size (KB)	T-time (ms)	S-time (ms)	T+S (ms)	k			
						0		1	
						M-time (ms)	T+S+M (ms)	M-time (ms)	T+S+M (ms)
従来	2〜10	19531	543	0	543	3389	3932	6091	6634
提案	2	256	13	11	24	19	43	35	59
	3	358	20	27	47	30	77	59	106
	4	510	32	51	83	48	131	94	177
	5	621	41	76	117	61	178	118	235
	6	654	43	102	145	64	209	127	272
	7	845	60	143	203	92	295	175	378
	8	792	55	157	212	85	297	162	374
	9	918	64	195	259	102	361	193	452
	10	992	72	233	305	112	417	210	515
改善比	2	76.29	41.77	0.00	22.63	178.37	91.44	174.03	112.44
	3	54.56	27.15	0.00	11.55	112.97	51.06	103.24	62.58
	4	38.30	16.97	0.00	6.54	70.60	30.02	64.80	37.48
	5	31.45	13.24	0.00	4.64	55.56	22.09	51.62	28.23
	6	29.86	12.63	0.00	3.74	52.95	18.81	47.96	24.39
	7	23.11	9.05	0.00	2.67	36.84	13.33	34.81	17.55
	8	24.66	9.87	0.00	2.56	39.87	13.24	37.60	17.74
	9	21.28	8.48	0.00	2.10	33.23	10.89	31.56	14.68
	10	19.69	7.54	0.00	1.78	30.26	9.43	29.00	12.88

	m	k							
		2		3		4		5	
		M-time (ms)	T+S+M (ms)	M-time (ms)	T+S+M (ms)	M-time (ms)	T+S+M (ms)	M-time (ms)	T+S+M (ms)
従来	2〜10	9055	9598	11895	12438	14225	14768	16637	17180
提案	2	***	***	***	***	***	***	***	***
	3	89	136	***	***	***	***	***	***
	4	133	216	177	260	***	***	***	***
	5	169	286	222	339	276	393	***	***
	6	179	324	234	379	287	432	344	489
	7	248	451	330	533	407	610	484	687
	8	230	442	306	518	377	589	447	659
	9	273	532	364	623	448	707	532	791
	10	299	604	397	702	489	794	580	885
改善比	2	***	***	***	***	***	***	***	***
	3	101.74	70.57	***	***	***	***	***	***
	4	68.08	44.44	67.20	47.84	***	***	***	***
	5	53.58	33.56	53.58	36.69	51.54	37.58	***	***
	6	50.59	29.62	50.83	32.82	49.56	34.19	48.36	35.13
	7	36.51	21.28	36.05	23.34	34.95	24.21	34.37	25.01
	8	39.37	21.71	38.87	24.01	37.73	25.07	37.22	26.07
	9	33.17	18.04	32.68	19.96	31.75	20.89	31.27	21.72
	10	30.28	15.89	29.96	17.72	29.09	18.60	28.68	19.41

	m	k							
		6		7		8		9	
		M-time (ms)	T+S+M (ms)	M-time (ms)	T+S+M (ms)	M-time (ms)	T+S+M (ms)	M-time (ms)	T+S+M (ms)
従来	2〜10	19148	19691	21656	22199	24072	24615	26585	27128
提案	2	***	***	***	***	***	***	***	***
	3	***	***	***	***	***	***	***	***
	4	***	***	***	***	***	***	***	***
	5	***	***	***	***	***	***	***	***
	6	***	***	***	***	***	***	***	***
	7	567	770	***	***	***	***	***	***
	8	520	732	597	809	***	***	***	***
	9	619	878	706	965	797	1056	***	***
	10	674	979	769	1074	865	1170	962	1267
改善比	2	***	***	***	***	***	***	***	***
	3	***	***	***	***	***	***	***	***
	4	***	***	***	***	***	***	***	***
	5	***	***	***	***	***	***	***	***
	6	***	***	***	***	***	***	***	***
	7	33.77	25.57	***	***	***	***	***	***
	8	36.82	26.90	36.27	27.44	***	***	***	***
	9	30.93	22.43	30.67	23.00	30.20	23.31	***	***
	10	28.41	20.11	28.16	20.67	27.83	21.04	27.64	21.41

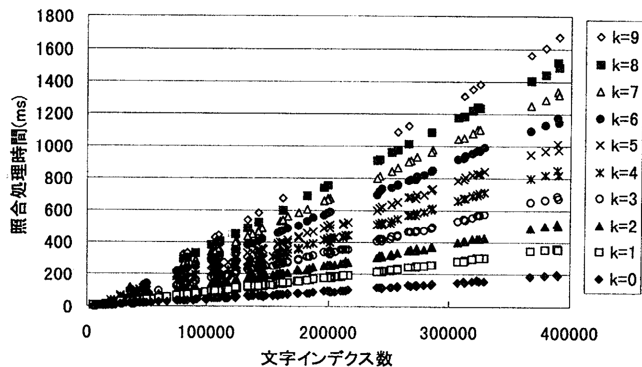


図10 文字インデクス数と照合処理時間の関係
Fig. 10 Relation between the number of character indexes and *SBA* matching time.

された．

蓄積された大量のテキストデータを検索する応用では，検索文字列長 m が 10 を超えることはまれと考えられるから，上記の比較結果より，検索速度における提案方式（*SBA* アルゴリズムの）の優位性が実証されたものと考える．

(3) 文字インデクス数との関係

(1) のデータ転送量は検索文字列長 m の増加におよそ比例して増加する傾向を示したが，詳細に見ると，$m = 7,8$ で逆転が生じていることに気づく．これは，各 m で検索文字列を構成する文字種の傾向を合わせるようにしたものの，実際には，m が同じでも個々の検索文字列ごとに参照される文字インデクス数☆に大きなばらつきがあったことに起因する．同じような傾向・現象が (2) の処理時間についても見られる．

4.6 節では，テキスト長 n，検索文字列長 m，不一致文字数 k に対する平均的な計算量を論じたが，個々の測定結果を説明し，*SBA* アルゴリズムの特性について踏み込んだ考察をするためには，文字インデクス数との関係に着目するのがよい．そこで，4.6 節において示した計算量を，文字インデクス数を基準に見直してみると，データ転送量とデータ転送時間は文字インデクス数と線形関係，整列処理時間は $\log m$ のファクタ分だけ m が大きくなるほど悪化，照合処理時間は文字インデクス数と不一致文字数 k の両ファクタに対して線形関係となる．

データ転送や整列の処理時間がこの関係に従うことは，一般によく知られたことなので，ここでは採りあげない．ここでは，照合処理時間との関係を確認するために，図 **10** を示す．図 10 は，実験で測定した 135 種類の検索文字列に対するすべての k の結果（810 ケース）について，各々のケースで参照された文字インデクス数と *SBA* アルゴリズムの照合処理時間（M-time）との関係をプロットしたグラフである．このグラフから，*SBA* アルゴリズムの照合処理時間が，文字インデクス数と不一致文字数 k の両ファクタに対して線形関係にあることがよく分かる．

☆ ここでいう文字インデクス数は，ペア型文字インデクスの数を意味している．図 3 の例でいえば，検索文字列「あうえ」に対して読み出された文字位置数の 6 個が，文字インデクス数である．

6. おわりに

本稿では，文字ベースの転置ファイルとビット配列を用いた，新しい文字列あいまい照合アルゴリズムとして「スキップ型 *Bit-Array* アルゴリズム」（*SBA* アルゴリズム）を提案した．*SBA* アルゴリズムは，Wu と Manber の *Bit-Array* アルゴリズムの考え方をベースとして，その計算式を変形することで，文字ベースの転置ファイル方式に組み合わせられるようにしたものである．テキスト長を n，検索文字列長を m，不一致文字数を k，文字セット規模を $|\Sigma|$ とすると，照合処理の計算量は，*Bit-Array* アルゴリズムが $O(nk)$ であるのに対して，*SBA* アルゴリズムは $O(kmn/|\Sigma|)$ であるが，$|\Sigma|$ に比べて m が十分小さければ *SBA* アルゴリズムの方が高速な検索速度が得られる．1000 万文字の日本語テキストに対する検索実験（$2 \leq m \leq 10$, $k \leq m$）では，*Bit-Array* アルゴリズムと比べて，*SBA* アルゴリズムの照合処理時間が約 1/28〜約 1/178 に短縮され，上記の優位性を確認できた．

大量テキストを検索する場合に事前に転置ファイルを作成しておく手法は，現在の主流となっている高速化技術である．本稿で提案した *SBA* アルゴリズムによれば，転置ファイルを使用することによる検索の高速性を生かした形で，文字列あいまい照合が可能になる．OCR 処理後のテキストなど，誤りを含む大量のテ

キストへの応用をはじめとする，実用的なアプリケーションへの適用が期待できる．

SBA アルゴリズム自身に関する今後の課題および展望としては，次のことがあげられる．

(1) より高速なインデクシング方式との組合せ

今回用いた転置ファイルは，1文字を検索キーとしたシンプルな形式のものである．しかし，完全照合の高速化を目指す研究では，検索キー長を長くとる方向で改良が進められている（N文字インデクス[10]やパトリシア[13]など）．*Bit-Array* アルゴリズムの考え方（本稿で導いた計算式）を，そのようなより高速なインデクシング方式と組み合わせていくことも考えてみたい．

(2) 更新頻度の高いテキストへの適用

転置ファイルを利用すると一般に，テキスト更新への対応が問題となるが，*SBA* アルゴリズムにおいても，これは同様である．検索キーの領域にB-tree構造を用いたり，位置情報の領域をブロック化して管理するなど，転置ファイルで一般に用いられているような更新対応の実装技法を取り込んでいく必要がある．

また，本稿で導いたビット配列のスキップ計算の式は，転置ファイルに適用することが必須とは限らない．すなわち，べた書きテキストの検索においても，検索文字列に現れない文字が連続している位置では，必ずしもすべての一致状態ビット配列を計算しなくてもよいことが4.1節，4.2節の議論から分かる．したがって，*SBA* アルゴリズムの発想は，べた書きテキストの走査型検索の形でも適用できる可能性がある．この場合は，転置ファイルを用いないので，*SBA* アルゴリズムほどの高速性は得られないであろうが，テキスト更新への対応の問題は生じない．

(3) *Bit-Array* アルゴリズムと同様の改良・発展

オリジナルの *Bit-Array* アルゴリズムについては，長い検索文字列に対する実装マシンのワード数を意識した改良，正規表現への対応などの議論がなされている[3]．本稿で述べたアルゴリズムは，*Bit-Array* アルゴリズムの精密な拡張なので，これらの議論はほぼそのまま適用できると予測している．

(4) 検索対象データの拡大

SBA アルゴリズムの特質から，文字セットが大きく，検索文字列が短い場合ほど，検索時間の改善は大きくなる．今後は，日本語以外のテキスト，さらに文字列ではなく一般の記号列に対する適用など，より広い問題に対する効果も評価することが望ましい．

参考文献

1) Hall, P.A.V. and Dowling, G.R.: Approximate String Matching, *ACM Computing Surveys*, Vol.12, No.4, pp.381–402 (1980).
2) Shang, H. and Merrettal, T.H.: Tries for Approximate String Matching, *IEEE Trans. Knowledge and Data Engineering*, Vol.8, No.4, pp.540–547 (1996).
3) Wu, S. and Manber, U.: Fast String Matching Allowing Errors, *Comm. ACM*, Vol.35, No.10, pp.83–91 (1992).
4) Baeza-Yates, R. and Gonnet, G.H.: A New Approach to Text Searching, *Comm. ACM*, Vol.35, No.10, pp.74–82 (1992).
5) 太田　学，片山紀生，高須淳宏，安達　淳：統計的手法による文字誤りテキスト検索，第52回情報処理学会全国大会論文集，5P-10 (1996).
6) 丸川勝美，藤澤浩道，嶋　好博：認識機能の出力のあいまい性を許容した情報検索手法の一検討，電子情報通信学会論文誌，Vol.J79-D-II, No.5, pp.785–794 (1996).
7) Computer Aids for Authors and Editors, *The Seybold Report on Publishing Systems*, Vol.13, No.10 (1984).
8) 池原　悟，小原　永，高木伸一郎：文書校正支援システムにおける自然言語処理，情報処理，Vol.34, No.10, pp.1249–1258 (1993).
9) 菊池忠一：日本語文書用高速全文検索の一手法，電子情報通信学会論文誌，Vol.J75-D-I, No.9, pp.836–846 (1992).
10) 赤峯　享，福島俊一：高速全文検索のためのフレキシブル文字列インバージョン法，情報処理学会 ADBS '96, pp.35–42 (1996).
11) Harman, D., Fox, E. Baeza-Yates, R. and Lee, W.: Inverted Files, *Information Retrieval – Data Structure and Algorithms*, Frakes, W.B. and Baeza-Yates, R., (Eds.), pp.28–43, Prentice Hall (1992).
12) 長尾　真（編）：自然言語処理，岩波講座ソフトウェア科学 15，岩波書店 (1996).
13) Gonnet, G.H., Baeza-Yates, R.A. and Snider, T.: New Indices for Text-PAT trees and PAT arrays, *Information Retrieval – Data Structure and Algorithms*, Frakes, W.B. and Baeza-Yates, R. (Eds.), pp.66–82, Prentice Hall (1992).

付録　本文式 (2) の導出

本文，式 (1) から式 (2) を導出する．

今，$S_{t_i} = B_0\ (x+1 \leq i \leq x+w)$ の条件下で，R_x^d から R_{x+w}^d を計算する式の導出を考える．まず，$S_{t_i} = B_0$，$i > 0$ では，本文式 (1) は次のように簡略

化される．

$$R_i^d = \begin{cases} Sft(R_{i-1}^0)\ AND\ S_{t_i} = B_0 & d=0 \\ \left. \begin{array}{l} Sft(R_{i-1}^d)\ AND\ S_{t_i} \\ \quad OR\ R_{i-1}^{d-1} \\ \quad OR\ Sft(R_{i-1}^{d-1}) \\ \quad OR\ Sft(R_i^{d-1}) \\ = R_{i-1}^{d-1}\ OR\ Sft(R_{i-1}^{d-1}) \\ \quad OR\ Sft(R_i^{d-1}) \end{array} \right) & d>0 \end{cases} \quad (A.1)$$

次に，式 (1) と式 (A.1) から，次式を数学的帰納法によって証明する．

$$R_i^d = R_{i-1}^{d-1}\ OR\ Sft(R_{i-1}^{d-1}) \quad (d>0) \quad (A.2)$$

まず，$d=1$ のときには，$Sft(\omega)$ が先頭ビットに 1 を埋めるビットシフト演算であることから，式 (A.1) の $d>0$ のケースは，次式となる．

$$\begin{aligned} R_i^1 &= R_{i-1}^0\ OR\ Sft(R_{i-1}^0)\ OR\ Sft(R_i^0) \\ &= R_{i-1}^0\ OR\ Sft(R_{i-1}^0)\ OR\ Sft(B_0) \\ &= R_{i-1}^0\ OR\ Sft(R_{i-1}^0)\ OR\ B_1 \\ &= R_{i-1}^0\ OR\ Sft(R_{i-1}^0) \end{aligned} \quad (A.3)$$

次に，ある $z(\geq 1)$ について，

$$R_i^z = R_{i-1}^{z-1}\ OR\ Sft(R_{i-1}^{z-1}) \quad (A.4)$$

が成立していると仮定し，式 (A.1) の $d>0$ のケースの計算式から，R_i^{z+1} を求める．まず前提として，Sft 演算の性質から，

$$Sft(\omega_1)\ OR\ Sft(\omega_2) = Sft(\omega_1\ OR\ \omega_2) \quad (A.5)$$

が成立する．また，式 (A.1) 中の $d>0$ のケース，

$$\begin{aligned} R_i^d = (&Sft(R_{i-1}^d)\ AND\ S_{t_i}) \\ &OR\ R_{i-1}^{d-1}\ OR\ Sft(R_{i-1}^{d-1}) \\ &OR\ Sft(R_i^{d-1}) \end{aligned} \quad (A.6)$$

の右辺に $Sft(R_i^{d-1})$ が論理和で加えられていることから，同じ $Sft(R_i^{d-1})$ を再度論理和で加えても結果に変化はなく，

$$R_i^d = R_i^d\ OR\ Sft(R_i^{d-1}) \quad (A.7)$$

も一般に成立することが分かる．したがって，式 (A.1) の $d>0$ のケースは，まず式 (A.4) 式 (A.5) の適用により次のようになる．

$$\begin{aligned} R_i^{z+1} &= R_{i-1}^z\ OR\ Sft(R_{i-1}^z)\ OR\ Sft(R_i^z) \\ &= R_{i-1}^z\ OR\ Sft(R_{i-1}^z)\ OR \\ &\qquad Sft(R_{i-1}^{z-1}\ OR\ Sft(R_{i-1}^{z-1})) \\ &= R_{i-1}^z\ OR \\ &\qquad Sft(R_{i-1}^z\ OR\ R_{i-1}^{z-1}\ OR\ Sft(R_{i-1}^{z-1})) \end{aligned} \quad (A.8)$$

さらにこれに式 (A.7) を適用すると，次のようになる．

$$\begin{aligned} R_i^{z+1} &= R_{i-1}^z\ OR\ Sft(R_{i-1}^z\ OR \\ &\qquad R_{i-1}^{z-1}\ OR\ Sft(R_{i-1}^{z-1})) \\ &= R_{i-1}^z\ OR\ Sft(R_{i-1}^z\ OR\ R_{i-1}^{z-1}) \\ &= R_{i-1}^z\ OR\ Sft(R_{i-1}^z)\ OR\ Sft(R_{i-1}^{z-1}) \\ &= R_{i-1}^z\ OR\ Sft(R_{i-1}^z) \end{aligned} \quad (A.9)$$

以上，式 (A.2) は，$d=1$ で成立し，かつ $d=z$ で成立するなら，$d=z+1$ でも成立することが示されたので，数学的帰納法より $d \geq 1$ で成立する．この式の前提は $S_{t_i} = B_0$ だけなので，その条件を満たす $(x+1)$〜$(x+w)$ の区間でも成立することは明らかである．

そこで，

$$F(\omega) = \omega\ OR\ Sft(\omega) \quad (A.10)$$

$$F_y(\omega) = \underbrace{F(F(F(\ldots F(\omega)\ldots)))}_{y\ times} \quad (A.11)$$

という演算を定義し，式 (A.2) に従って R_{x+w}^d を計算する．

$$\begin{aligned} R_{x+w}^d &= F(R_{x+w-1}^{d-1}) \\ &= F(F(R_{x+w-2}^{d-2})) = F_2(R_{x+w-2}^{d-2}) \\ &= \cdots \\ &= F_y(R_{x+w-y}^{d-y}) \end{aligned} \quad (A.12)$$

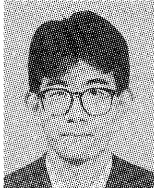
すなわち，R_{x+w-y}^{d-y} が既知であれば，その値に F という演算を y 回繰り返すことで，R_{x+w}^d が計算できる．

$d \geq w$ の範囲では，$y=w$ のときに該当する R_x^{d-w} から，$F_w(R_x^{d-w})$ によって R_{x+w}^d が計算できる．また，$d<w$ の範囲では，$y=d$ に該当する R_{x+w-d}^0 に F という演算を d 回繰り返すことで計算できる．このとき式 (A.1) より $R_{x+w-d}^0 = B_0$ であり，$F_d(B_0) = B_d$ なので，$R_{x+w}^d = B_d$ となる．さらに，式 (A.1) より $R_{x+w}^0 = B_0$ なので，$d=0$ においてもこの式は成立している．以上を整理すると次のようになり，本文式 (2) が導出される．

$$R_{x+w}^d = \begin{cases} F_w(R_x^{d-w}) & (d \geq w) \\ B_d & (d < w) \end{cases} \quad (2)$$

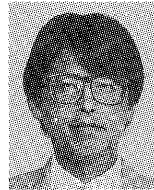
（導出終）

(平成 10 年 4 月 1 日受付)
(平成 11 年 1 月 8 日採録)



下村　秀樹（正会員）

1965年生．1993年東京農工大学大学院工学研究科博士後期課程（電子情報工学）修了．同年，日本電気（株）入社．1998年7月よりソニー（株）．1990年本学会プログラミングシンポジウムにて山内奨励賞受賞．情報検索，文字認識知識処理，文書作成環境に興味を持つ．工学博士．



福島　俊一（正会員）

1958年生．1982年東京大学理学部物理学科卒業．同年，日本電気（株）入社．現在，同社ヒューマンメディア研究所主任研究員．高速で頑健な探索アルゴリズムの切り口から自然言語処理・情報検索分野の研究開発に取り組んでいる．言語処理学会，人工知能学会，情報知識学会，情報科学技術協会，ACM各会員．工学博士．情報処理学会平成4年度論文賞，第45回全国大会奨励賞，第53回全国大会優秀賞，平成9年度坂井記念特別賞受賞．