記　号　処　理　36－1
プログラミング言語　5－1
（１９８６．３．１０）

日本語テキストにおける パターンマッチング手法の比較と改善

尹志熙　　高木利久　　牛島和夫
（九州大学　工学部）

1. まえがき

日本語の字種の多い性質から日本語文字を表現するには１文字当たり 2バイト以上を必要とする．一般に日本語テキストには，2バイトの日本語文字と 1バイトの従来の英数文字が混在しており，そのコーディング方法や混在方法は計算機の日本語処理システムによって異なる．このような日本語テキスト上でパターンマッチングを行うためには，英数文字と日本語文字との区別（シフトコードの存在）や日本語文字のずれ読みなどを常に意識する必要があり，その操作は英数文字だけのテキストに比べて複雑になる．

本稿では，日本語テキスト上での文字列のパターンマッチングについて検討を行う．英文テキスト用として一般に知られているいくつかのパターンマッチングアルゴリズムを日本語テキストに適用し，その効率を調べた．しかし，上記のように日本語テキストは英文テキストとは異なる特性を持っており，アルゴリズムの適用はそれほど簡単ではない．そこで，我々はまずテキストを正規化テキスト[1] と呼ぶ形式に変換し，そのテキスト上でパターンマッチングを行うことを試みた．

1個のパターンを対象とする場合と複数個のパターンを対象とする場合それぞれについて英文テキスト用アルゴリズムを正規化日本語テキストに適用し，その効率の比較，改善を行った．また，この結果と正規化しないテキスト上でのアルゴリズムの効率との比較も合わせて行った．

2. 日本語テキスト

2.1 日本語テキストの性質

日本語テキストの形式は各々の計算機メーカの日本語処理システムに依存しているが，一般に日本語テキストには，2バイトの日本語文字と1 バイトの従来の英数文字が混在しており，その混在を可能とする方法には大きく分けてシフトコードを使う方法と使わない方法がある[2].

どちらの方法を使うにしても日本語テキストのテキスト処理プログラムを作成する際，次のようなことが問題になる．1バイトコードからなる英文テキスト上での文字と文字の間のポインタの移動は簡単であり，次の文字へポインタを移動するにはポインタを１バイト末尾方向に，一つ前の文字へポインタを移動するにはポインタを１バイト先頭方向に，移動するだけですむ．ところが，日本語テキスト上の文字と文字の間のポインタの移動のためには英数文字と日本語文字との区別を常に意識する必要があり，その操作にはかなり手間がかかる．また，2バイトの日本語文字コードの第 1バイト目の文字コードの集合と第 2バイト目の文字コードの集合とは共通要素を持つことがあり，日本語文字列の上で 1バイト単位でポインタを移動させる場合に図1 に示すような日本語文字のずれ読みの起こる可能性がある．

2.2 正規化日本語テキスト

我々の研究室では 2バイトの日本語文字と 1バイトの英数文字の混在方法として正規化日本語文字コードを提案している[1]．正規化日本語文字コードは英数文字の前に１バイトのパッディングを加え，すべての文字を2 バイトで表すコード系である．この文字コードを用いた日本語テキストを正規化日本語テキストという．このテキストは，正規化されていない元のテキストに比べて，テキスト長が長くなる，正規化のための手間がかかるという不利はあるが，テキスト処理に必須のポインタの移動などが容易に行えるため日本語テキスト処理プログラム開発の生産性を高めるという特徴を持っている．

3. 実験方法

3.1 パターンマッチングアルゴリズム

文字列のパターンマッチングとはテキスト上でパターンと呼ばれる文字列（ 1個又は複数個）が

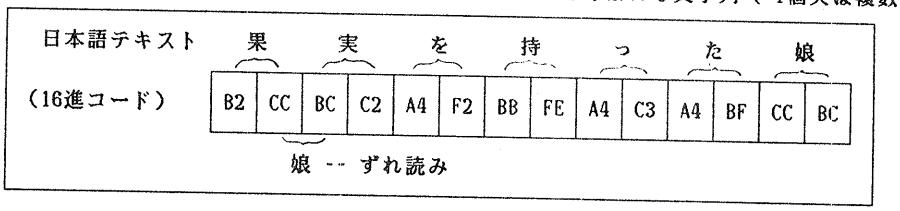

図 1．日本語文字のずれ読みの例

存在するかどうかを調べ，もし存在すれば，その最初の位置またはすべての位置を求め，存在しなければその旨を示す問題である．

対象とするパターンの個数が 1個である場合，Knuth-Morris-Pratt[3] やBoyer-Moore[4]によるアルゴリズム（各々KMP,BMと略す）などがすでに考案されており，英文テキスト上ではBMがこれらのアルゴリズムの中で最もよい効率を示す[5] ことが知られている．一方，パターンの個数が複数である場合，英文テキスト上では有限オートマトンを用いるAho-Corasick[6] アルゴリズム（ACと略す）の有効性が知られているが[7] ，最近，BMの拡張型であるExpanded Boyer-Moore[8] アルゴリズム（EBM と略す）が発表され，その実用性が報告されている．ここでは， 1個のパターンを対象とする場合，KMP，BM および従来からある二次パターンマッチングアルゴリズム[5](QDと略す)の三つのパターンマッチングアルゴリズムを日本語テキストに適用し，その効率を比較した．複数個のパターンを対象とする場合は，ACとEBM についてその効率を調べた．

3.2 日本語テキストへの適用

今回，実験には九州大学大型計算機センターのFACOM M380S OS IV/F4 JEF のFDMS（和文エディタ）で作成した日本語文章テキスト（FDMSテキストと呼ぶ）とそれを正規化したものを用いた．FDMSテキストには，日本語文字（2 バイト）と英数文字（1 バイト）がシフトコードを境界として混在している．図2 にFDMSテキストと正規化日本語テキストの形式を示す．

文字数 5000,20000,40000,80000 のテキスト上でパターンマッチング（全出現を見つける）を行い，各々のアルゴリズムのパターンマッチング時間を測った．その結果，各々のテキスト上でほぼ同じ結果を得ており，以下では5000字のテキスト上での実験結果だけを示すことにする．

パターンはランダムに実験テキスト上から選んだものを使用した． 1個のパターンを対象とする場合は 1から14までのパターン長に関して，それぞれ20個のパターンを選び，実験を行いその平均値をそのパターン長におけるパターンマッチング時間とした．

また，複数個のパターンを対象とする場合は 1から14までのパターン長のパターンをそれぞれ 1個から100 個まで選び実験を行った．これらの実験を10回繰り返し，その平均値を複数個のパターンを対象とする場合のパターンマッチング時間とした．

4. 1 個のパターンを対象とする場合

FDMSテキスト
AS: アルファベットシフトコード
KS: 漢字シフトコード

A	B	情	報	C	D	E
C1	C2	KS	bef0	caf3	AS	C3	C4	C5

正規化日本語テキスト

A	B	情	報	C	D	E
00C1	00C2	bef0	caf3	00C3	00C4	00C5

図 2. 日本語テキストの形式

4.1 正規化テキスト上でのパターンマッチング

4.1.1 2バイト単位のパターンマッチング

KMP やBMなどのパターンマッチングアルゴリズムは暗黙のうちに英文テキストをその処理対象としており，日本語文字と英数文字が混在している日本語テキストにこれらのアルゴリズムをそのまま適用することは困難である．最も簡単なQDとしてもFDMSテキストでは常にシフトコードを意識しながらパターンマッチングを行う必要がありそのアルゴリズムはかなり複雑になる．

しかし，正規化日本語テキストを使えば，KMP，BM，QDなどの英文テキスト用のアルゴリズムにおける1バイト単位（英文テキストの文字単位）のマッチングを， 2バイト単位のマッチングに置き換えるだけで，アルゴリズムの適用が可能になる．正規化日本語テキスト上での各々のアルゴリズムのパターンマッチング時間を図 3の KMP-2，BM-2 QD-2に示す．ここで KMPやBMはテキストの上でパターンマッチングを行う前に予めパターンを解析しテーブルなどを作成しておく処理（これを前処理と呼ぶ）を必要とするため，そのパターンマッチング時間としては前処理時間を含む全体の CPU 時間を測った．

QD-2や KMP-2はほぼ同じ実行時間を示している．理論上，KMP は字種の少ないテキストを対象とするような場合，QDに比べて優れた効率を持つが，字種が多い日本語の場合，QDとKMP はほぼ同じ効率を示すことが図3 からわかる．

一方，英文テキスト上で最も効率のよいアルゴリズムとして知られているBMが，日本語テキスト上ではQDやKMP よりも低い効率を示している．BMではマッチング中ミスマッチが起こったとき，パターンをいくつ右にずらすかを表わすのに字種の大きさのテーブル（delta1と呼ぶ）を用いる．と

ころが，このテーブルを作るためには，まず，文字コードに対応するテーブル要素を初期化する必要があり，字種の多い日本語テキスト上では効率低下の原因となる．すなわち，字種の多い日本語テキスト上では，このdeltaIテーブルを作るための前処理の手間が増え，BMの場合，三つのアルゴリズムの中で最も効率が落ちている．

我々は，BMの前処理の効率の改善のため，このテーブルを階層的に表現することにより，その大きさを圧縮し，その初期化のための手間を減らす改善方法（NEW-BMと呼ぶ）を考案した[9]．図 4の(A) に従来の方法によるテーブル形式を，(B) にNEW-BMで実現したテーブルの形式を示す．このアルゴリズムによる実行時間を図 3のNEW-BM-2に示す．この図からパターンの長さM が 4以上の場合，QD-2やKMP-2 に比べてNEW-BM-2が高速であることがわかる．

4.1.2 1バイト単位のパターンマッチング

前項では文字単位のパターンマッチングを前提としていたが，この項では，日本語テキストを 1バイトコードの列として見做し，英文テキストの場合と同じく，パターンマッチングを 1バイト単位で行う手法について考察する．しかしこのように2 バイトで表現されている文字を 1バイト単位でマッチングする場合，図 1で示したようなずれ読みの起こる可能性があり，これを防ぐ方法を考えなければならない．ところが，テキストが正規化されている場合，この判別は非常に簡単である．すなわち，すべての文字が2 バイトで表現されていることから，パターンを見つけたとき，そのパターンの見つかった位置をチェックし，それが奇数バイト目であるものを選択すればよい．

この方式によりアルゴリズムを実現し，各々のパターンマッチング時間（前処理時間を含む）を

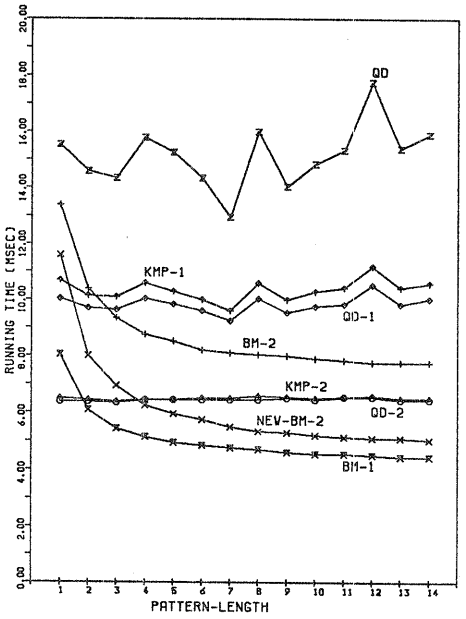

図 3．パターンマッチング時間（単位 ： msec）
（前処理時間を含む）

測定した．その結果を図 3のQD-1,KMP-1，BM-1に示す．QD-1,KMP-1のパターンマッチング時間がQD-2やKMP-2 に比べて約 2倍程度かかっている．すなわち，2 バイト単位でマッチングを行う場合，N をテキスト長とすると，QD-2,KMP-2の実行時間がO(N) に比例するに対して， 1バイト単位でマッチングを行う場合,QD-1,KMP-1 の実行時間はO(2N)に比例している．

一方，BM-1は優れた特性を示し，NEW-BM-2よりもよい効率を示している．これは，BM-1はNEW-BM-2よりマッチング回数が増えるにも拘らず，字種

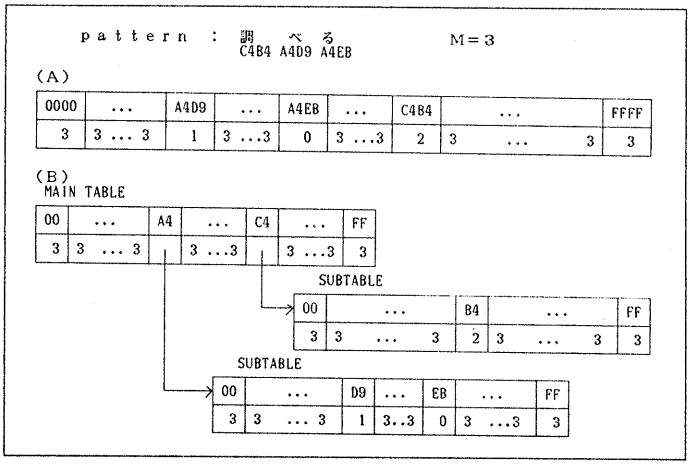

図 4．deltaIテーブルの圧縮表現

の大きさのテーブルをつくるための手間とそれを検索する時間が減ることによる効果の方が大きいことによるものである．

なお，1バイト単位でパターンマッチングを行い，パターンの見つかった位置をチェックした結果，間違えて発見したパターン（偶数バイト目のもの）の比率を表 1に示す．パターン長M が 5以上である場合，ずれ読みによるパターンマッチングは生じなかった．

表 1．1バイト単位のパターンマッチングによるずれ読みパターンの出現比率

M	mis-detection（％）	M	mis-detection（％）
1	11.30	8	0.00
2	0.00	9	0.00
3	18.30	10	0.00
4	21.99	11	0.00
5	0.00	12	0.00
6	0.00	13	0.00
7	0.00	14	0.00

4.2 非正規化テキスト上のパターンマッチング

正規化されていないFDMSテキスト上でパターンマッチングを行う場合，2.1 で示したように日本語テキストの性質のためBMのようなアルゴリズムの適用は殆ど不可能であり[9]，QDのような簡単なものでもアルゴリズムにかなりの修正を加えることになる．FDMSテキスト上でのQDによるパターンマッチング時間を図 3のQDに示す．

4.3 比較

QD，KMP，BM を正規化日本語テキストに適用しその効率の比較を行った．しかし，この場合，まず，テキストを正規化する必要があり，テキストの正規化時間をパターンマッチング時間に加えなければならない．正規化テキスト上でパターンマッチングを行う図 3の各々のアルゴリズムのパターンマッチング時間はテキストの正規化のための時間を含んでいる．

表 2にテキスト長の異なる 4つの日本語テキストを正規化するための時間を示す．

表 2．テキストの正規化のための時間
（テキスト長：文字数，正規化時間：msec）

テキスト長	5000	20000	40000	80000
正規化時間	4.00	16.59	33.39	67.00

ここで，テキストの正規化やアルゴリズムの実現にはFORTRAN77 を用いた．図3 の結果は5000字のテキストを使用しているので，QD-2,KMP-2，BM-2，QD-1，KMP-1，BM-1 には正規化時間の 4msecが加えられている．

以上の実験結果によると，パターン長 1の場合を除くと，正規化日本語テキスト上でのBMによる1 バイト単位のパターンマッチングが最もよい効率を示すと結論できる．パターン長 1の場合は正規化日本語テキスト上でのQDによる 2バイト単位のパターンマッチングが最も高速である．

5．複数個のパターンを対象とする場合

5.1 正規化テキスト上でのパターンマッチング

5.1.1 Aho-Corasickアルゴリズム

対象とするパターンの個数が複数である場合，英文テキスト上ではAho,Corasickによるアルゴリズムが有効である[7]．このアルゴリズムでは，ある定まったパターンに関して有限オートマトン（パターンマッチングマシン又はマシンと呼ぶ）を作成することにより，テキストを 1回検査するあいだに，テキスト中に存在するすべてのパターンの出現位置を検出できる．このため，パターンマッチングマシンの作成時間を除けばACのパターンマッチング時間はパターンの個数に無関係な一定の値を持つ．図 5に{ HE，SHE，HIS，HERS} のパターン集合を対象とするACパターンマッチングマシンを示す[6]．マシンは，goto関数，failure 関数，output関数から構成され，ここでの円は状態を表わし，実線は状態遷移を示すgoto関数を，破線はfailure 関数を，二重円の状態は出力を持つ最終状態（output関数）を表わす．

正規化日本語テキスト上で 2バイト単位のパターンマッチングを行うことにより，ACを日本語テキストに適用(AC-2 と呼ぶ）することができる．一般にACマシンの状態遷移表は処理速度の高速化のため，行列形式を用いて文字コードとマシンの状態を引数とする2 次元配列で表現され，その配列の大きさは（テキストの字種＊状態数）に達する．この場合日本語テキストを処理対象とするAC-2の状態遷移表は，（ 2^{16} ＊状態数）の大きさになり，表の実現のための記憶容量や表の初期化の

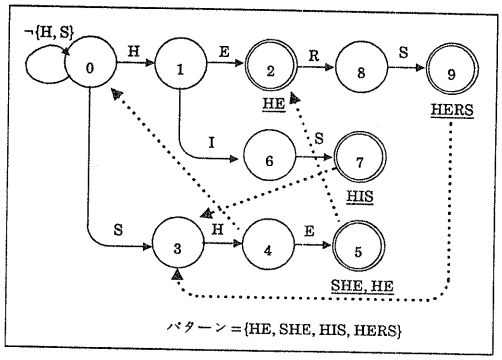

図 5　ACマシン

ための時間を考えると実用性があるとはいい難い．

正規化日本語テキスト上で 1バイト単位のパターンマッチングを行うことにより，状態遷移表の大きさの問題は解決できる[7] が， 1バイト単位のパターンマッチングを行うACマシン（AC-1と呼ぶ）を作成する際，4.1.2 の場合と同じく，文字列のずれ読みによるずれ読みパターンを取り除く方法を考えなければならない．図 6に{ 娘，息子} のパターン集合を対象とするAC-1パターンマッチングマシンを示す．

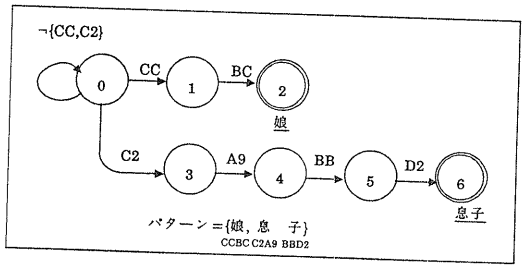

図 6　AC-1マシン

英文テキストの場合，テキスト文字が 1 文字ずつ入力文字として与えられたとき，図 5のACマシンはそのテキスト文字と現在マシンの状態を用いて次の状態への状態遷移を行い，その次の状態が最終状態（出力を持つ状態）であるのかを確認することによってパターンの出現を検出する．しかし，この方法では，図 6のAC-1マシンの場合，図 1 で示したような入力文字列 "果実を持った娘" が与えられたときずれ読みを起こすことが明かである．ところが，この場合もテキストが正規化されていることからそのずれ読みパターンを取り除く方法を簡単に見つけることができる．すなわち，テキストが 2バイトに正規化されていることから，2 バイトおきにパターンの出現を表わす最終状態を調べることにより，ずれ読みによるパターンを取り除くことが可能になる．

この方法によるACアルゴリズムのパターンマッチング時間を図 7（パターンの個数が 1から10までの場合），図 8（パターンの個数が10から100までの場合）のAC-1に示す．3.2 で述べたように実際にはパターン長 Mが 1から14までに関して実験を行ったが，ここではM が 1と 3の場合ついてのみ示した．ここで，AC-1は前処理のための時間（マシンの作成時間）を含んでいるため，パターンの個数が増えると共にパターンマッチング時間も増加している．

5.1.2 Expanded Boyer-Moore アルゴリズム

EBM[8]はBMの拡張型であり，BMではテキストを 1 回検査するあいだに 1個のパターンしか検出できないことに対して，EBM を用いることによりテキストを 1回検査するあいだにテキスト中に存在する複数個のパターンの出現位置を検出することができる．

EBM を日本語テキストに適用する際，BMの場合と全く同じことが問題になる．まず，正規化テキスト上で 2バイト単位のパターンマッチングを行う場合，EBM はdeltalテーブル（BMのdeltalテーブルの拡張型）を必要とするため，字種の多い日本語テキスト上では，BMの場合と同じく，テーブル作成のための時間がかかる．次に，テキスト上で 1バイト単位のパターンマッチングを行う場合，ずれ読みの起こる可能性があるが，BMの場合と同

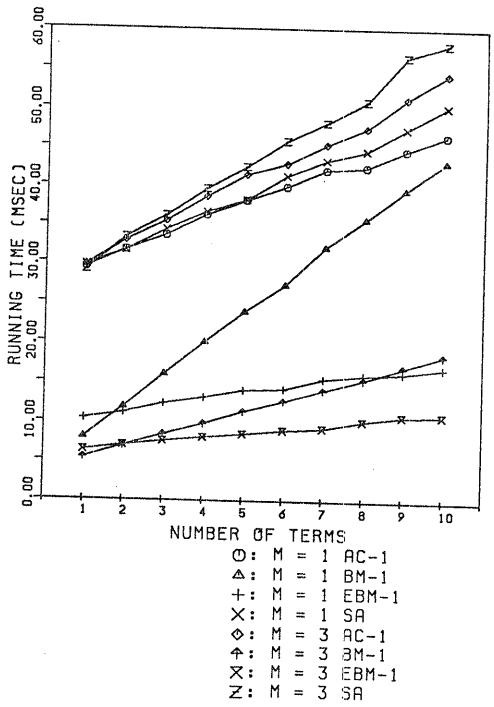

図 7．パターンマッチング時間（単位 ： msec）
（前処理時間を含む）

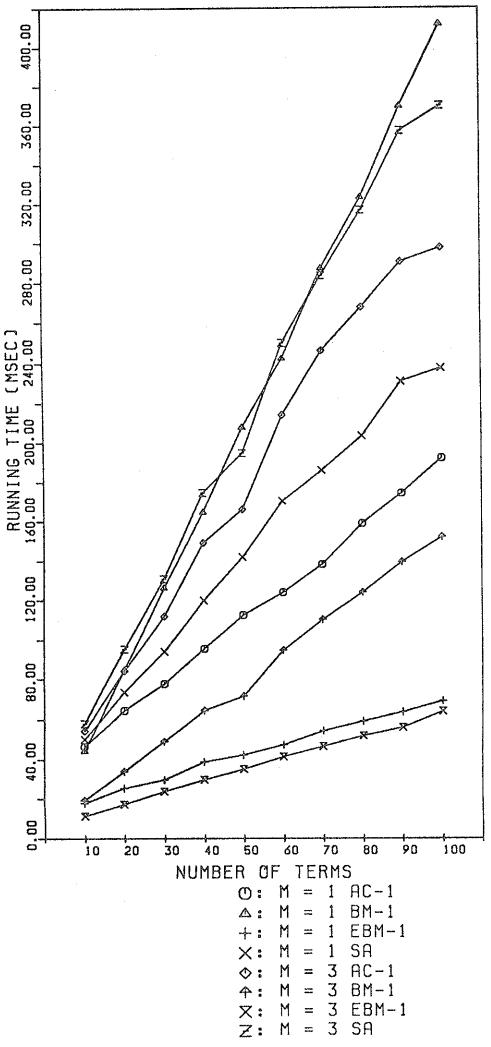

図 8. パターンマッチング時間（単位 ： msec）
（前処理時間を含む）

じく，パターンの出現位置を確認するだけでずれ読みのパターンを取り除くことができる．

正規化日本語テキスト上での 1バイト単位でパターンマッチングを行うEBM （EBM-1 と呼ぶ）の実行時間を図 7，8 のEBM-1 に示す．この実験結果により，M = 1 の場合，パターンの個数が 1個増えるたびにその実行時間は3%から12% まで増加している．また，BMではパターン長が長くなると共に実行時間が減少するが，EBM もこの性質をひきついでいる．比較のため，BM-1の実行時間を図 7，8に示す．BMでは一度にパターンを 1個ずつ検出するため,K個のパターンを検出する場合，その実行時間は1 個のパターンマッチング時間の約 K 倍になっている．この図から，パターンの個数が多くなると（2 以上）EBM-1 がBM-1に比べて，よい効率を示すことがわかる．

5.2 非正規化テキスト上のパターンマッチング

テキストの正規化を行わず，日本語テキスト上でコード系の特性を反映させた有限オートマトンを作成することによりパターンマッチングを行う方法[10]（SAと略す）が篠原らにより提案されている．この方法によるFDMSテキスト上でのパターンマッチング時間（有限オートマトンの作成時間を含む）を図 7，8 のSAに示す．

5.3 比較

図 7,8の結果により，日本語テキスト上で複数個のパターンを対象としパターンマッチングを行う場合， EBM-1が有効であることがわかった．ここで，AC-1，EBM-1，BM-1 は正規化日本語テキスト上でパターンマッチングを行っており，図 7,8 の正規化テキスト上でパターンマッチングを行う各々のアルゴリズムのパターンマッチング時間は正規化のための手間を含んでいる．

AC-1の場合SAに比べて，テキストの正規化の時間が多少かかるが，パターンマッチングのとき最終状態を 2バイトおきに調べるため，SAとAC-1はほぼ同じ効率を示している．

図 7,8のすべてのアルゴリズムの実行時間は前処理時間を含んでおり，特に，AC-1やSAの場合，全体のパターンマッチング時間の中で前処理の占める時間は大きい．ところが，テキスト長が長くなると，全体のパターンマッチング時間に比べて，前処理時間の占める比率は少なくなる．図 9にパターンの個数 Kが10から100 である場合について各々のアルゴリズムの前処理時間を除いたパターンマッチング時間を測定しその結果を示した．この図によると前処理時間を除く場合，パターン長 M = 1，パターンの個数 K = 50以上のとき，または，M = 3,K = 70以上のとき， AC-1 と SA は EBM-1 より高速である．

6.まとめ

正規化テキストを用いることより，英文テキスト用として開発された有効なアルゴリズムを日本語テキストに適用することができ，またその適用のためのアルゴリズムの修正も簡単であった．すなわち，すべての文字が 2バイトに正規化されているため，英文テキスト用のアルゴリズムにおける 1バイト単位のマッチングを，2 バイト単位のマッチングに置き換えるだけでアルゴリズムの適用が可能になり，また，日本語テキスト上で 1バイト単位のパターンマッチングを行う場合もそのとき生じる問題の解決方法が簡単に見つかった．

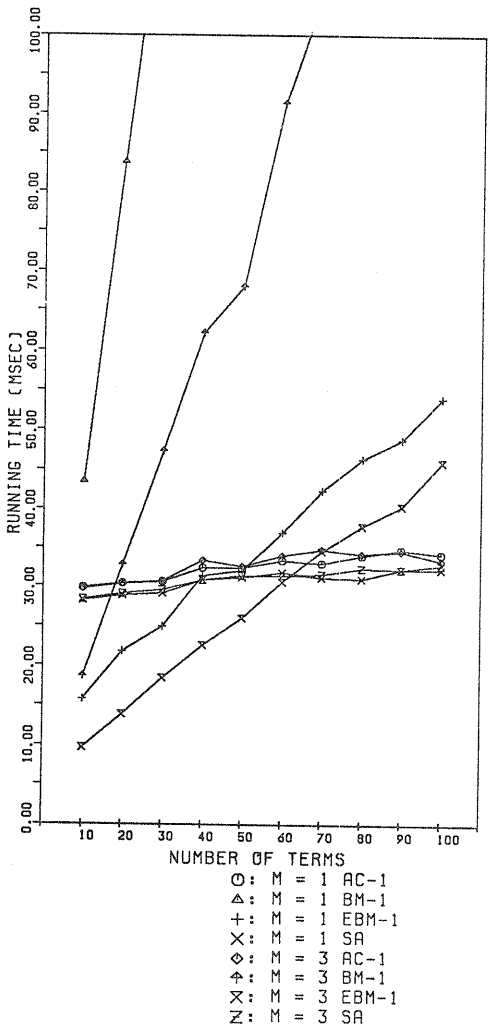

図 9. パターンマッチング時間（単位 ： msec）
（前処理時間を含まない）

現在までの結果を次にまとめる．

(1) 対象とするパターンが 1個である場合 ：
パターン長 1の場合を除くと，正規化日本語テキスト上でのBMによる1 バイト単位のパターンマッチングが最もよい効率を示す．パターン長 1の場合は正規化日本語テキスト上でのQDによる 2バイト単位のパターンマッチングが最も高速である．

(2) 対象とするパターンが複数個である場合 ：
EBM-1 が有効である．但し，テキスト長がかなり長い場合（前処理時間を無視できる場合），パターンの個数が約50個以下であるときはEBM-1 が，パターンの個数が約50個以上であるときはAC-1または SA が有効である．

すなわち，正規化テキスト上でパターンマッチングを行うときテキストの正規化のための時間が問題になるが，テキストの正規化のため多少時間がかかるとしても，これらの方法によるパターンマッチングが高速であったといえる．

7. おわりに

日本語テキスト上に QD, KMP, BM, AC, EBM アルゴリズムを適用，それらの効率比較を行った．

ここでは，FDMSテキストを対象として実験を行ったが，今後様々な形式のテキストについて検討を加える．

なお，この研究の一部は昭和60年度科学研究費補助金一般研究(B) 60460228の援助を受けた．

参考文献

[1] Ushijima, K., Kurosaka, T. and Yoshida, K., "SNOBOL4 with Japanese Text Processing Facility," Proc. ICTP'83, pp. 235-240 (1983)

[2] 川副，吉田，牛島，" 日本語テキスト処理プログラムの流通性を上げるためのプリミティブ，" 電子通信学会技術研究報告，EC84-59, pp. 1-12, (1985)

[3] Knuth, D.E., Morris, J.H.Jr. and Pratt V. R.,"Fast Pattern Matching in Strings," SIAM J. Computing, Vol. 6, No. 2, pp. 323-350 (1977)

[4] Boyer, R. S. and Moore,J. S.,"A Fast String Searcingh Algorithm," Comm. ACM, Vol. 20, No.10, pp. 762-772 (1977)

[5] Smit, G., "A Comparison of Three String Matching Algorithms," Software-Practice and Experience, Vol. 12, pp. 57-66 (1982)

[6] 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)

[7] 有川，篠原， "パターンマッチングマシンの実現における Time-Space Trade off について，" 情報処理学会第27回全国大会，pp. 11-12 (1983)

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

[9] 尹，高木，牛島，" 日本語テキストのためのパターンマッチング手法について，" 電子通信学会技術研究報告，EC84-58, pp. 25-33 (1985)

[10]篠原，有川， "日本語テキスト用のAho-Corasick型パターン照合アルゴリズム，" 情報処理学会研究報告，85-NL-52 (1985)