キーワードの遅延抽出を考慮した文書検索構造の効率的構成法

岡田 真[†] 安藤一秋[††]
森田和宏[†] 青江順一[†]

文書から抽出されたキーワードを索引表のキー（見出し）とする文書検索技術は非常によく利用されているが，複合語キーワードの的確な抽出技術は，依然として重要な課題となっている．しかし，抽出条件は利用目的に依存するので，キーワード候補の決定を検索段階まで遅延できれば，目的に応じたキーワード抽出と検索が実現できる．この課題に対して，本論文では，複数キーワードの文字列照合マシン AC（Aho and Corasick）を拡張することで，複合語キーワードのすべての候補をコンパクトに格納できる検索構造を提案し，検索構造上で目的に合ったキーワードを決定できる手法を提案する．9万ファイル（キーワード 496,837 語）による実験により，提案手法によるキーワード抽出遅延の有効性，検索構造の高速性が実証される．

An Efficient Construction of Text Retrieval Structures Considering Delayed Keyword Extraction

Makoto Okada,[†] Kazuaki Ando,[††] Kazuhiro Morita[†] and Jun-ichi Aoe[†]

Although extracting keywords efficiently is an important task in text retrieval systems, it is very difficult to determine suitable keywords for arbitrary purposes because there are many compound words. This paper presents a retrieval structure that can delay keyword extraction until a retrieval stage. It needs to integrate the keyword extraction stage into the keyword retrieval stage. A string pattern matching machine by Aho and Corasick (AC) is extended to the usage of a delayed extraction and retrieval structure. The approach is evaluated by the experimental estimation that is supported by the simulation results for 90,000 Japanese text files.

1. はじめに

文書などの一次情報から二次情報となるキーワードを抽出し，そのキーワードを索引として目的とする一次情報を検索する手法は非常によく利用されている技術である[6),8)〜10),14),16),17)]．しかし，キーワードとなる複合語を的確に抽出する技術は，まだ重要な課題として残されている．極論すれば，キーワードには，長単位語（複数形態素で構成）と短単位語（単一自立語の形態素）とが考えられ[11),12)]，長単位語優先処理では，キーワードとならない短単位語を除去できる反面，索引の完全一致検索により，短単位語の検索漏れが生じる．また，短単位語優先処理では，この検索漏れは生じないが，複合語の構成情報が失われるので，キーワード自体の精度に問題が残る．

以上の理由により，従来の研究では，文書解析や複合語構造の解析技術を駆使して，目的に応じた複合語キーワードを抽出段階で効率的に選択する手法がさかんに提案されてきた[10),12)〜15)]．特に，この選択処理をできるだけ遅延させるために，林ら[6)]は複数キーワードに対する照合マシン AC（Aho and Corasick）[1)]を利用して，複合語キーワード候補の選択処理（短単位語と長単位語優先処理など）を抽出の最終段階で行う手法を提案した．しかし，この手法でも，抽出段階でキーワードを決定しなくてはならないので，検索段階ではキーワードが固定されたものとなる．

これらの問題を解決するには，キーワード候補の絞り込みを検索段階まで遅延し，短単位語と長単位語の両方からなるキーワード候補を可能な限り多く抽出しておき，検索段階で目的に応じたキーワードを取捨選択できる手法を確立することが必要である．しかし，このためには，多くのキーワードに対する検索表と，

† 徳島大学工学部
Faculty of Engineering, Tokushima University
†† 香川大学工学部
Faculty of Engineering, Kagawa University

それに対する膨大なレコード情報〔一次情報を検索するためのファイル番号列や用例文番号列などのポスティング（postings）情報〕をコンパクトな構造に格納し，さらにその構造上でキーワードの選択検索が実現できなければならない．

本来，キー検索技法は完全一致を実現するための手法であり，一般的には，前方部分一致検索の可能なトライ法[2),3)]が利用される．しかし，長単位語に含まれる短単位語を検索するためには，完全一致以外に入力キーワードを部分的に含む別のキーワードの検索（部分一致検索と呼ぶ）が必要となるため，トライ法だけでは部分一致検索には不十分である．これに対して，望月ら[13)]は，特徴ベクトルを検索構造に組み込むことで，部分一致検索を実現したが，この手法では索引表自体でキーワードを選択することは不可能である．本論文では，マシン AC を拡張することで，この課題を実現する．

以下，2 章では，マシン AC について説明し，3 章で，マシン AC を検索構造に拡張する方法を提案する．4 章では，高速化のために検索構造を圧縮する手法を提案する．5 章で，理論的評価と実験による評価で提案手法の有効性を示し，6 章で，まとめと今後の課題を述べる．

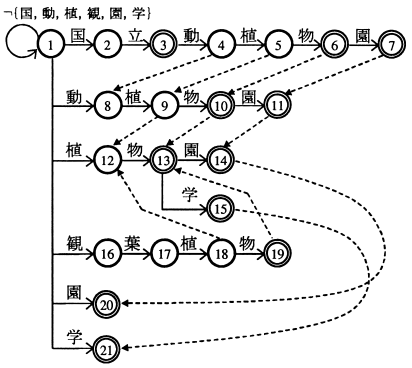
2. 複合語キーワードとマシン AC

文字列 $y_1, y_2, \ldots, y_h; h \geq 1$ を要素とする有限集合を K で表すとき，マシン AC は K に属するすべての部分文字列 y およびその位置を任意のテキスト（文字列）t の中から検出する．状態の集合を S，入力記号の集合を H とするとき，マシン AC の動作は次の 3 つの関数で制御される[1),2),15)]．

goto 関数　g: $S \times H \rightarrow S \cup \{fail\}$
failure 関数　f: $S \rightarrow S$
output 関数　$output$: $S \rightarrow K$ の部分集合

本論文では，複合語キーワードをマシン AC に格納する．複合語キーワード集合 K={“国立動植物園”, “動植物”, “動植物園”, “植物”, “植物園”, “植物学”, “観葉植物”, “園”, “学” } に対するマシン AC を**図 1** に示す．ただし，図 1 (a) の ¬{ 国, 動, 植, 観, 園, 学 } は，‘国’, ‘動’, ‘植’, ‘観’, ‘園’, ‘学’ 以外の記号を表す．

図 1 (a) の実線による状態遷移図は goto 関数であり，初期状態 1 から状態 2 へのラベル ‘国’ の遷移を g(1,‘国’)=2 で表す．また，状態 1 からラベル ‘立’ の遷移は定義されていないので，g(1,‘立’)=$fail$ と表し，このとき「状態遷移に失敗した」という．図 1 (a) の破線で示す failure 関数は，goto 関数による状態遷移に失敗したときの状態遷移先を定義する☆．図 1 (b) の output 関数は，状態 s に到達したとき，$output(s)$ に含まれるキーワードが検出されたことを示す．



(a) goto 関数と failure 関数

状態番号 s	$output(s)$
3	{ 国立 }
6	{ 国立動植物, 動植物, 植物 }
7	{ 国立動植物園, 動植物園, 植物園, 園 }
10	{ 動植物, 植物 }
11	{ 動植物園, 植物園, 園 }
13	{ 植物 }
14	{ 植物園, 園 }
15	{ 植物学, 学 }
19	{ 観葉植物, 植物 }
20	{ 園 }
21	{ 学 }

(b) output 関数

図 1　キーワード集合 K に対する AC マシン
Fig. 1　An AC machine for keyword set K.

マシン AC による照合アルゴリズム[1)]は，以下で示される．

【マシン AC の照合アルゴリズム】

照合を行う文字列を $X = a_1 a_2 \cdots a_i \cdots a_n$ $(1 \leq i \leq n)$，goto 関数を g，failure 関数を f，output 関数を $output$ とする．また，マシン AC 上の現在の状態を $state$ で示す．

手順 (A-1)｛初期化｝

$state$ を初期状態 1 に，i を 1 に初期化する．

手順 (A-2)｛照合処理｝

$state$ に $g(state, a_i)$ を代入する．$state$=$fail$ならば，$state$ に $f(state)$ を代入し，手順 (A-2) の先頭へ戻る．さもなければ，手順 (A-3) へ進む．

☆ 説明の便宜上，初期状態 1 への failure 関数は省略する．

手順 (A-3)｛output 関数の出力｝

$output(state) \neq \phi$ のとき，i と $output(state)$ を出力する．

手順 (A-4)｛文字位置の変更と終了判定｝

i をインクリメントする．$i \leq n$ ならば，手順 (A-2) へ戻る．$i > n$ ならば，終了． **(アルゴリズム終)**

図 1 の例で，入力文字列 “新動植物学” に対する照合動作を簡単に説明する．まず，初期状態 1 より goto 関数を検索すると，g(1,‘新’)=1，g(1,‘動’)=8，g(8,‘植’)=9，g(9,‘物’)=10 より状態 10 まで遷移し，$output$(10)={“動植物”，“植物”} を出力する．さらに，g(10,‘学’)=$fail$なので，f(10)=13 により状態 13 へ遷移し，g(13,‘学’)=15 より $output$(15)={“植物学”，“学”} を出力し終了する．

このように，マシン AC は，入力文字列の 1 回の走査で，部分文字列となるキーワードをすべて検出できる．

3. マシン AC による検索構造

マシン AC をキー検索構造に適用するために，キーワード x のポスティング情報（レコード情報）を参照するポインタを P(s) で表し，以後，参照情報と呼ぶ．ここで，s はマシン AC 上の状態である．この参照情報を要素として持つ関数 output を関数 output1 に改める．たとえば，図 1 (a) において，状態 13 の output 関数 $output$(13)={“植物”} を output1 関数に改めると，$output1$(13)={P(13)} となる．

また，H 上の全文字列の集合を H^*，空記号列（empty string）を ε で表すとき，goto 関数は，以下のように拡張される[4]．

goto 関数 g: $S \times H^* \rightarrow S \cup \{fail\}$

これにより，任意の文字列 x に対して，$g(s,x) = t$，$t \in S$ なる状態遷移が可能となる．

図 1 (a) で，短単位語のキーワード x=“植物” を検索すると，g(1,x)=13 なる状態 13 で，$output1$(13)={P(13)} しか得られず，短単位語で構成される複合語キーワード（“植物園”，“動植物”，“動植物園” など）に対する参照情報が検索できない．また，ワイルドカード ‘$*$’ による検索キーワード指定（たとえば，“$*$園”，“国立 $*$” による指定）も不可能である．

このような短単位語を構成要素に含む複合語キーワード検索をマシン AC 上で実現する．

まず，状態 s の逆 failure 関数を $f^{-1}(s)$ で表し，状態集合を出力する関数として定義する．ただし，初期状態からの逆 failure 関数は考慮しない．次に，空記号列（empty string）でない文字列 y，z に対して，キー x の完全一致；x を接頭辞に含む xy；x を接尾辞に含む zx；x を真（proper）部分文字列に含む zxy の検索をそれぞれ完全，接頭，接尾，中間検索と呼び，マシン AC に対して，以下のように定義する．

(1) 完全検索 EXACT(x)

EXACT(x) = {P(s)|$g(1,x) = s$ なる状態 s に対して，P(s) $\in$ $output1(s)$}

たとえば，x=“植物” に対して，EXACT(x)={P(13)} が定義される．完全検索の出力はたかだか 1 個である．

(2) 接頭検索 PREFIX(x)

PREFIX(x) = {P(t)|$g(1,x) = s$，P(s) $\in$ $output1(s)$ なる状態 s に対して，$g(s,y) = t$ が存在して，P(t) $\in$ $output1(t)$}

x=“植物” に対して，PREFIX(x)={P(14),P(15)} が得られ，それぞれ “植物” を接頭辞に含む “植物園” と “植物学” の参照情報を表す．

(3) 接尾検索 SUFFIX(x)

SUFFIX(x) = {P(t)|$g(1,x) = s$，P(s) $\in$ $output1(s)$ なる状態 s に対して，$t \in f^{-1}(s)$ なる t と $g(1,zx) = t$ なる z が存在して，P(t) $\in$ $output1(t)$}

x=“植物” に対して，SUFFIX(x)={P(10),P(6),P(19)} が得られ，それぞれ “植物” を接尾辞に含む “動植物”，“国立動植物” と “観葉植物” の参照情報を表す．

(4) 中間検索 PROPER(x)

PROPER(x) = {P(r)|$g(1,x) = s$，P(s) $\in$ $output1(s)$ なる状態 s に対して，$t \in f^{-1}(s)$ なる t と $g(t,y) = r$ なる y と r が存在して，P(r) $\in$ $output1(r)$}

x=“植物” に対して，PROPER(x)={P(11),P(7)} が得られ，それぞれ “植物” を真部分列に含む “動植物園” と “国立動植物園” の参照情報を表す．

以上のように，接頭や接尾，中間検索では，各 output1 関数が持つ完全一致の情報しか利用しないため，failure 関数を構築[1]する際に output1 関数を更新する必要はない．

図 1 (a) に対するそれぞれの検索結果を**図 2** に示す．図 2 のハイフン ‘-’ は，参照情報がないことを示す．

4. 検索の高速化と圧縮法

3 章で提案した検索手法は，最悪の場合，goto 関数と逆 failure 関数の遷移をすべて走査する必要があるので，検索アルゴリズムの最悪の時間計算量は，総ノード数 n に対して $O(n)$ となる．しかし，この最悪の計算量は，特異なマシン AC で生じるので，実用

状態番号 s	EXACT(x)	PREFIX(x)	SUFFIX(x)	PROPER(x)
3	{ P(3) }	{ P(6), P(7) }	–	–
6	{ P(6) }	{ P(7) }	–	–
7	{ P(7) }	–	–	–
10	{ P(10) }	{ P(11) }	{ P(6) }	{ P(7) }
11	{ P(11) }	–	{ P(7) }	–
13	{ P(13) }	{ P(14), P(15) }	{ P(10), P(6), P(19) }	{ P(11), P(7) }
14	{ P(14) }	–	{ P(11), P(7) }	–
15	{ P(15) }	–	–	–
19	{ P(19) }	–	–	–
20	{ P(20) }	–	{ P(14), P(11), P(7) }	–
21	{ P(21) }	–	{ P(15) }	–

図 2 各検索法の結果情報
Fig. 2 Result information for each retrieval method.

上は大きな問題とはならない[1),15)]．より現実的な問題は，格納するキーワードが多くなり，マシン AC が主記憶上に在中できない場合に生じる．この場合，遷移の走査回数は，二次記憶からのアクセス回数に比例するので，検索時間が大きく低下する．本章では，この解決法を提案する．

4.1 高速検索のための情報圧縮法

高速化を実現するために，構成されたマシン AC に対して，完全，接頭，接尾，中間検索の参照情報をあらかじめ計算しておき，その結果情報から二次記憶へアクセスする手法を用いる．この手法により，マシン AC 上の走査は，failure 関数を考慮しない，goto 関数上の縦型探索による完全一致検索のみとなるので，トライ法と同様に深さ優先で一定のブロックに分割して二次記憶に格納することで，検索のディスクアクセス回数を一定にできる☆．

しかしながら，結果情報をすべて保存しようとした場合，その記憶容量の増大が問題となる．本節では，結果情報の圧縮方法を提案する．本手法の着眼点は，結果情報の包含関係であり，これを図 2，**図 3** で説明する．

図 2 と図 3 (a)，(b) の goto 関数の木構造（goto 木と呼ぶ）から分かるように，接頭検索結果は，PREFIX(“国立”)=EXACT(“国立動植物”) ∪ PREFIX(“国立動植物”)={P(6),P(7)} ⊃ EXACT(“国立動植物園”) ∪ PREFIX(“国立動植物園”)={P(7)} なる包含関係を満たす．また，接尾検索結果では，SUFFIX(“園”)=EXACT(“植物園”) ∪ SUFFIX(“植物園”)={P(14),P(11),P(7)} ⊃ EXACT(“動植物園”) ∪ SUFFIX(“動植物園”)={P(11),P(7)} ⊃ EXACT(“国立動植物園”) ∪ SUFFIX(“国立動植物”)={P(7)} なる包含関係が見られるが，これらは図 2 と図 3 (c)，(d) で示すように $f^{-1}(s)$ による木構造（f^{-1} 木と呼ぶ）から得られる．同様に中間検索結果も PROPER(“植物”)=EXACT(“動植物園”) ∪ PROPER(“動植物”)={P(11), P(7)} ⊃ EXACT(“国立動植物園”) ∪ PROPER(“国立動植物”)={P(7)} なる包含関係が見られる．

この包含関係を利用して，P(s) を一次元配列 ARY に重ね合わせて格納し，ARY の探索開始位置を示す配列 START と探索数を示す配列 OFFSET，マシン AC の状態番号と START，OFFSET のインデックスを管理する配列 IND から成る検索表を作成

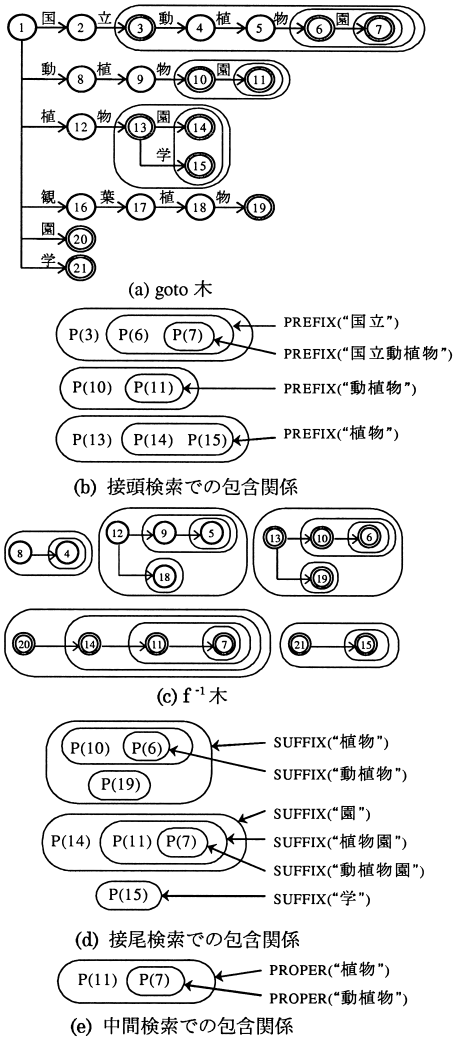


図 3 包含関係の例
Fig. 3 Examples of inclusive relationships.

☆ メモリには，深さ 1 の状態のみを格納し，深さ 2 以降の状態は，深さ優先で任意サイズのブロックに分割し，二次記憶に格納することでアクセス回数の一定化を実現できる．

状態番号s	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
EP_IND	0	0	1	0	0	2	3	0	0	4	5	0	6	7	8	0	0	0	0	9	10	11
S_IND	0	0	0	0	0	0	0	0	0	2	5	0	1	4	0	0	0	0	0	3	6	
PS_IND	0	0	0	0	0	0	0	0	0	2	0	0	1	0	0	0	0	0	0	0	0	

(a) 各検索表へのインデックス

	1	2	3	4	5	6	7	8	9	10	11
EP_START	1	2	3	4	5	6	7	8	9	10	11
EP_OFFSET	3	2	1	2	1	3	1	1	1	1	1

	1	2	3	4	5	6	7	8	9	10	11
EP_ARY	P(3)	P(6)	P(7)	P(10)	P(11)	P(13)	P(14)	P(15)	P(19)	P(20)	P(21)

(b) 完全接頭検索表

	1	2	3	4	5	6
S_START	1	2	4	5	6	7
S_OFFSET	3	1	3	2	1	1

	1	2	3	4	5	6	7
S_ARY	P(10)	P(6)	P(19)	P(14)	P(11)	P(7)	P(15)

(c) 接尾検索表

	1	2
PS_START	1	2
PS_OFFSET	2	1

	1	2
PS_ARY	P(11)	P(7)

(d) 中間検索表

図 4 結果情報の検索表
Fig. 4 Tables of retrieval results.

する．したがって，この圧縮構造における検索結果は，状態 t に対するインデックス J(=IND[t]) が与えられたとき，START[J]=I，OFFSET[J]=K より ARY[I], ARY[$I+1$], $\cdots$, ARY[$I+K-1$] となる．IND，ARY，START，OFFSET に対しては，完全接頭，接尾，中間検索表を区別するために，EP_，S_，PS_を先頭に付加する．図 2 に対する圧縮例を**図 4** に示す．

ここで，完全検索と接頭検索の結果情報は，goto 木の構造上，同一処理で得られるので，互いに圧縮統合して格納される．したがって，状態 t に対する完全接頭検索情報が EP_ARY[I], EP_ARY[$I+1$], $\ldots$, EP_ARY[$I+K-1$] と得られた場合，EP_ARY[I] が完全検索の結果情報になり，残りの EP_ARY[$I+1$] から EP_ARY[$I+K-1$] までの (EP_OFFSET[EP_IND[t]]-1) 個が接頭検索の結果情報となる．

図 4 でキーワード x=“植物” の接頭検索を考えてみる．$g(1,x)$=13 となるので，図 4 (a) から EP_IND[13]=6 なので，図 4 (b) の完全接頭検索表より EP_START[6]=6，EP_OFFSET[6]=3 が得られる．その結果 EP_ARY[6]=P(13)，EP_ARY[7]=P(14)，EP_ARY[8]=P(15) となる．ここで，P(13)=EXACT(x) なので，接頭検索 PREFIX(x) の結果は，{P(14), P(15)} となる．

4.2 結果情報の圧縮アルゴリズム

前節で述べた包含関係は，マシン AC の goto 木と f^{-1} 木より得られるので，これらの木構造を入力として，配列 IND，ARY，START，OFFSET を決定する圧縮アルゴリズムを以下に提案する．ただし，状態 t の子供の集合を CHILD(t) で表す．

【結果情報の圧縮アルゴリズム】

手順 (B-1)｛初期化｝

処理中の状態番号 t を木構造の根 root に初期設定し，配列 ARY のインデックス IA，START と OFFSET のインデックス IO をそれぞれ 1 に初期化する．また，状態番号に対して START の格納位置を管理する配列 IND，ARY，START，OFFSET の全要素を 0 に初期化する．

手順 (B-2)｛枝の遷移と IND，START への入力｝

t をスタックにプッシュする．ただし，$output1(t) \neq \phi$ ならば，ARY[IA] に検索情報 P(t) を，IND[t] に IO；START[IO] に IA を格納し，IA と IO をインクリメントする．

手順 (B-3)｛遷移済みの枝の削除｝

CHILD(t) $\neq \phi$ ならば，t を $r \in$CHILD(t) なる r に改め，CHILD(t) から r を除き，手順 (B-2) へ戻る．CHILD(t) $= \phi$ ならば，手順 (B-4) へ進む．

手順 (B-4)｛OFFSET への入力｝

H = IND[t] $\neq 0$ ならば，START[H] はすでにセットされているので，OFFSET[H] に IA$-$START[H] を格納する．

手順 (B-5)｛終了判定｝

スタックからポップし，スタックが空でなければ，スタックのトップの状態番号を t として，手順 (B-3) へ戻る．スタックが空であれば終了する．

(アルゴリズム終)

上記のアルゴリズムを使って，まず，完全と接頭検索の結果を圧縮するために goto 木を処理する．次に，接尾検索結果や中間検索結果を圧縮するために，上記のアルゴリズムで複数の f^{-1} 木を処理する．そこで，完全や接頭検索の情報が重複して格納されるのを避けるために，手順 (B-2)，手順 (B-4) を変更する必要がある．

ある状態 s における接尾検索の結果情報は，逆 failure 関数で遷移した状態 t の完全検索の結果情報となる．そこで，接尾検索表を作成するには，f^{-1} 木を深さ優先探索で走査し，遷移した先の状態が持つ完全検索情報を接尾検索情報として検索表に順次追加していけばよい．

以下に，変更後の手順を示す．

手順 (B'-2)

t をスタックにプッシュする．まず，t が根 root ならば，S_IND[t] に IO；S_START[IO] に IA を格納し，IO をインクリメントした後，手順 (B-3) へ進む．t が root でなく，かつ $output1(t) \neq \phi$ ならば，S_ARY[IA] に $output1(t)$ を格納し，IA をインクリメントする．ここで CHILD(t) $\neq \phi$ であれば，S_IND[t] に IO；S_START[IO] に IA を格納し，IO をインクリメントする． **（手順 (B'-2) 終）**

手順 (B'-4)

H = S_IND[t] $\neq$ 0 ならば，S_START[H] はすでにセットされているので，S_OFFSET[H] に IA−S_START [H] を格納する．

ここで，S_OFFSET[H]=0 ならば，S_START[H]，S_OFFSET[H] に 0 を格納し，IO をデクリメントして，S_IND[t] を 0 にする． **（手順 (B'-4) 終）**

また，中間検索の結果情報も，接尾検索と同様に f^{-1} 木を走査して検索表を作成するが，逆 failure 関数で遷移した状態の接頭検索の結果情報を利用する点が異なる．

具体的には，次の手順と手順 (B'-4) を用いる☆．

手順 (B"-2)

t をスタックにプッシュする．まず，t が根 root ならば，PS_IND[t] に IO；PS_START[IO] に IA を格納し，IO をインクリメントした後，手順 (B-3) へ進む．t が root でなく，かつ EP_IND[t](=H)≠0，EP_OFFSET[H]≥2 ならば，PS_ARY[IA] に EP_ARY[$H+1$] から (EP_OFFSET[H]−1) 個の結果情報を格納し，IA に EP_OFFSET[H]−1 を加える．そこで CHILD(t) $\neq \phi$ であれば，PS_IND[t] に IO；PS_START[IO] に IA を格納し，IO をインクリメントする． **（手順 (B"-2) 終）**

接頭と中間検索表は，複数の f^{-1} 木から構成されるので，新しい f^{-1} 木を処理するたびに状態 t 以外の変数を初期化する必要はない．

以上のアルゴリズムによって完全接頭検索表ができる流れを，図 3 (a)，(b) を用いて説明する．

まず，状態 t を木構造の根である状態 1 に初期設定し，変数 IA，IO，配列 EP_IND，EP_ARY，EP_START，EP_OFFSET を初期化する．次に，手順 (B-2) で t をスタックにプッシュし，手順 (B-3) へ進む．CHILD(1) $\neq \phi$ なので，t を状態 2 に改め，CHILD(1) から状態 2 を除き，手順 (B-2) へ戻る．状態 2 をスタックにプッシュし，$output1(2)=\phi$ なので手順 (B-3) へ進む．t を状態 3 に変えて CHILD(2) から状態 3 を除いた後，手順 (B-2) へ戻る．状態 3 をスタックにプッシュして，t が root でなく，$output1(3) = P(3) \neq \phi$ なので，EP_ARY[1(=IA)]=P(3)，EP_IND[3]=1(=IO)，EP_START[1(=IO)]=IA=1 と入力し，IA と IO をそれぞれインクリメントして 2 とする．以下同様に状態 4，状態 5，状態 6，状態 7 に対して手順 (B-2) と (B-3) が繰り返される．また，$output1(6) \neq \phi$，$output1(7) \neq \phi$ なので，状態 6 では EP_ARY [2(=IA)]=P(6)，EP_IND[6]=2(=IO)，EP_START[2(=IO)]=2(=IA) と入力して，IA と IO をインクリメントして，それぞれ 3 にする．同様に，状態 7 では，EP_ARY[3]=P(7)，EP_IND [7]=3，EP_START[3]=3 と入力されて，IA と IO をインクリメントして，それぞれ 4 とする．状態 7 で，CHILD(7)=ϕ より，手順 (B-4) へ進み，H=EP_IND[7]=3≠0 より，EP_OFFSET[3]=IO−EP_START[3]=4−3=1 となり，手順 (B-5) へ進む．スタックから状態 7 をポップして，スタックのトップの状態 6 を t とする．スタックは空でないので，手順 (B-3) へ戻る．ここで，状態 6 から状態 2 までの CHILD はすべて ϕ なので，手順 (B-3) から (B-5) までの処理が繰り返されるが，EP_IND[t]≠0 の状態 3 では EP_IND[3]=1 より EP_OFFSET[1]=4−1=3，状態 6 では，EP_IND[6]=2 より EP_OFFSET[2]=4−2=2 がそれぞれ入力される．

以上の操作をすべての状態に対して，つまり CHILD(1) = ϕ となるまで繰り返す．その結果，図 4 (a)，(b) にある検索表 EP_IND，EP_START，EP_ OFFSET が得られる．

また，接尾検索表，中間検索表については同様の処理を f^{-1} 木に対して行い，図 4 (c)，(d) の検索表を得る．

結果情報の圧縮後は，入力キーワードと完全一致する状態の output 関数と検索表を参照するだけで，完全，接頭，接尾，中間検索結果が取得できるため，failure 関数を利用する必要がない．したがって，提案手法の最終構造は，goto 関数と output 関数，検索表により構成される．また，検索の効率化のためにマシン AC を決定性オートマトンに変換して利用する場合が多いが，本論文では，f^{-1} 木の取得を容易にするために，変換を行わない．

5. 評　　価

5.1 理論的解析

4.2 節で圧縮された検索表における参照情報の検索

☆ 手順 (B'-4) の S_を PS_に変更する．

表 1 実験結果
Table 1 Experimental results.

	$K1$	$K2$	$K3$	$K4$	$K5$
キーワードと索引数の情報					
ファイル数	10,000	30,000	50,000	70,000	90,000
キーワード数	32,520	78,788	116,904	150,100	181,043
索引数	92,063	225,867	333,679	423,332	509,962
キーワードの総出現数	55,048	165,079	275,712	386,280	496,837
提案手法の情報					
検索表の要素数					
IND	276,189	677,601	1,001,037	1,269,996	1,529,886
ARY	269,064	762,051	1,178,866	1,488,228	1,809,055
START,OFFSET	317,724	794,452	1,175,558	1,483,498	1,787,248
ARY の圧縮率（%）	65.8	71.7	74.1	73.3	73.4
索引辞書の容量					
goto 関数（MB）	0.70	1.72	2.55	3.23	3.89
検索表（MB）	3.3	8.5	12.8	16.2	19.6
総容量（MB）	4.0	10.2	15.3	19.4	23.4
ポスティング情報					
容量（MB）	0.21	0.63	1.05	1.47	1.90
情報数	55,048	165,079	275,712	386,280	496,837
平均情報数	0.60	0.73	0.83	0.91	0.97
最大情報数	144	304	583	849	1,101
トライ法の情報					
索引辞書の容量					
トライ（MB）	0.7	1.7	2.5	3.2	3.9
参照情報（MB）	0.4	0.9	1.3	1.6	1.9
総容量（MB）	1.1	2.6	3.8	4.8	5.8
ポスティング情報					
容量（MB）	0.91	2.80	4.64	6.42	8.26
情報数	238,033	734,598	1,217,400	1,683,563	2,166,548
平均情報数	2.59	3.25	3.65	3.98	4.25
最大情報数	797	2,315	4,099	5,824	7,611
比較情報					
索引辞書容量の比率	3.79	3.96	4.02	4.01	4.02
ポスティング容量の比率	0.23	0.22	0.23	0.23	0.23
総容量の比率	2.14	2.02	1.94	1.85	1.80

は，すべて配列参照で行われるので，最悪の時間計算量は $O(1)$ となる．圧縮アルゴリズムの時間計算量のうち，手順 (B-2)，手順 (B-3)，手順 (B-5) でスタックと関数 CHILD(t) を使って goto 関数をたどる時間計算量は，深さ優先探索アルゴリズムに準拠しているので，goto 関数がリスト構造で表現されていれば，状態数 n，遷移数 m に対して，$O(m+n)$ となる[7]．

また，手順 (B-2) と (B-4) で，配列 IND，START，OFFSET を構築するための計算量は，キーワード数 k に比例するので，圧縮アルゴリズム全体の時間計算量は $O(m+n+k)$ となる．

次に，検索表の記憶容量について考える．

完全一致および接頭検索の各配列の要素数は，goto 木上に存在する $output1(s) \neq \phi$ なる状態 s の総数 p と等しくなるので，その要素の総数は $4p$ 個となる．また，すべての f^{-1} 木上において，$output1(s) \neq \phi$ なる状態 s の総数が q，終端状態の総数が r，それらのうち，$output1(s) \neq \phi$ である初期状態の総数を u とした場合，接尾検索表に追加される要素数は $(4q-3r-u)$ 個となり，同様の f^{-1} 木上で，PREFIX(x) $\neq \phi$ かつ初期状態でない状態 s の総数が v であるとき，中間検索表に追加される要素数は $3(u+v-r)$ と PREFIX(x) の総出力数の和となる．以上により，全検索表配列の記憶容量は，各検索表の要素数の総和となる．

5.2 実験による評価

実験データとして，文部省学術情報センター（NACSIS）の情報検索システム評価用テストコレクション（NTCIR）[5]☆を利用した．まず，NTCIR から 5 個以上のキーワードが定義されているデータファイルを抽出し，1 万ファイル分のキーワードを持つ集合を $K1$，3 万，5 万，7 万，9 万ファイル分のキーワードを持つ集合をそれぞれ $K2$，$K3$，$K4$，$K5$ として定義した．また，ポスティング情報は，各キーワードを含むファイルの ID 番号とした．実験に使用したマシンの CPU は Celeron 300 MHz，メモリは 320 MB で，開発言語は C++である．これらのキーワード集合に対して，提案手法とトライ法の比較実験を行った結果を**表 1** に示す．

表 1 の索引数は，各キーワードに対して，その構成

☆ テストコレクション中のファイルには，論文表題，著者名，所属学会，出典，発表年次，論文概要，著者の定めたキーワードなどが記述されている．

要素の形態素，隣接形態素のすべての組合せを索引とした数である☆．本実験では，状態遷移ラベルを 1 形態素と定義したので，goto 関数の遷移数は索引数と等しくなる．提案手法の検索表の要素数は，完全，接頭，接尾，中間検索の各配列の要素数を示す．また，索引辞書の容量は，goto 関数と検索表の容量の和であり☆☆，goto 関数の容量は goto 関数の遷移数（索引数），検索表の容量は各配列の要素数から決まる．一方，トライ法の索引辞書の容量は，トライと参照情報の容量の和である．ここで，goto 関数とトライは同一の索引で構築されるため，同じ構造になる．また，ポスティング情報には容量，情報数，1 索引に対する平均と最大情報数を示す．

表 1 より，索引数が 2.42 倍，3.59 倍，4.62 倍，5.57 倍（$K1$ から $K5$）と増加するとき，検索表配列の容量も 2.59 倍，3.89 倍，4.92 倍，5.94 倍とほぼ同じ比率で増加する．これは，理論的解析で述べたように，検索表配列が登録キーワード数（索引数）に依存することを実証している．

索引辞書容量の比率（提案手法の値をトライ法の値で割った値で示す．以下同様）より，提案手法の索引辞書の容量は，検索表を含むのでトライ法より大きくなるが，その比率は，索引数が多くなると一定値（約 4 倍）に飽和することが分かった．これはファイル数（文書量）の増加率に対して，キーワード数の増加率が低くなっているからである．

次に，ポスティング容量の比率より，提案手法はトライ法の約 1/4 に圧縮されていることが分かった．これは，提案手法では，キーワードの完全一致の状態にのみポスティング情報を登録するだけで，接尾，中間などの検索が実現できるのに対して，トライ法で提案手法に近い検索を実現するためには，キーワードを構成する形態素や隣接形態素のすべての組合せに対して，ポスティング情報を登録する必要があるためである．この結果は，ポスティング情報の平均と最大情報数から確認できる．また，ARY の圧縮率が，約 72%となることより，4.2 節の包含関係を利用した圧縮アルゴリズムの有効性が分かる．

ポスティング情報も含めた総容量の比率では，提案手法の容量はトライ法の容量の約 2 倍となっているが，この比率はキーワードが増加するにつれて減少していくことが分かった．これは，トライ法でのポスティング容量の増加率が提案手法の増加率より高く，総容量に対するポスティング容量の占有率も高いからである．

以上により，包含関係を利用した圧縮アルゴリズムの有効性が確認できた．

次に，検索時間の比較・評価を行う．検索対象には $K5$ を使用し，ポスティング情報は二次記憶上に置くものとする．

$K5$ の全キーワードに対して完全，接頭，接尾，中間検索を行い，ポスティング情報を取得するまでの平均検索時間を取得した．検索時間の内訳は，goto 関数と検索表の検索時間である．ここで，提案手法の goto 関数とトライは同じ構造であるため，両者の検索時間は等しい．

検索表をメモリ上に置いた場合，1 キーワードあたりの平均検索時間は約 4.21 ミリ秒であり，その内訳は goto 関数が平均 4.21 ミリ秒，検索表が平均 0.01 ミリ秒以下であった．また，検索表を二次記憶に移した場合でも，検索時間は平均 4.31 ミリ秒と十分高速で，両者とも検索表の検索時間は，全検索時間にほとんど影響しないことが確認できた．したがって，提案手法とトライ法の検索時間は同等であるといえる．

以上により，提案手法は，複合語の構成関係を指定した検索機能，完全一致検索と遜色がない検索速度と記憶容量を実現できる点で，きわめて有用な手法であるといえる．

6. むすび

マシン AC を拡張して，キーワード候補の決定を検索段階まで遅延可能な検索手法を提案した．また，包含関係を利用して，参照情報を圧縮して格納する構造を提案した．実験の結果，提案手法は従来手法と比較して，記憶容量の面でも検索性能の面でも十分に有効であることを確認した．

今回利用した論文抄録の文書データでは，1 ファイルあたりのキーワード数が 5 個と非常に少ないものであった．しかし，一般的にファイルが大きくなれば，1 ファイルのキーワード数は多くなる．したがって，数ギガバイトを超えるような大規模文書データベースへの応用と評価が今後の課題となる．また，キーワードの追加登録・削除が可能である動的アルゴリズムの考案も行う．

☆ “自然/言語/処理” なるキーワードに対しては，完全一致の “自然/言語/処理” と，形態素の “自然”，“言語”，“処理”，隣接形態素の組合せ “自然/言語”，“言語/処理” が登録される．

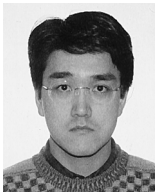
☆☆ failure 関数は検索表の作成時にだけ必要となるので，容量に含める必要はない．

参考文献

1) 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).
2) Aoe, J.: An Efficient Implementation of Static String Pattern Matching Machines, *IEEE Trans. Softw. Eng.*, SE-15, No.8, pp.1010–1016 (1989).
3) Aoe, J.: An Efficient Digital Search Algorithm by Using a Double-Array Structure, *IEEE Trans. Softw. Eng.*, SE-15, No.9, pp.1066–1077 (1989).
4) 青江順一，森本勝士，長谷美紀：トライ構造における接尾辞の圧縮アルゴリズム，電子情報通信学会論文誌，J75-D-II, No.4, pp.770–779 (1992).
5) 情報検索システム評価用テストコレクション（NTCIR）（予備版），文部省学術情報センター（NACSIS）.
6) 林淑 隆，中野英雄，獅々堀正幹，青江順一：文字列照合マシンを利用した複合語キーワードの効率的抽出法，情報処理学会論文誌，Vol.38, No.4, pp.815–825 (1997).
7) 石畑 清：アルゴリズムとデータ構造，p.236, 岩波書店 (1989).
8) 伊藤 哲，丹羽寿男，萱嶋一弘，丸野 進，〆木泰治：利用目的に応じて最適化可能なキーワード抽出手法，電子情報通信学会技術研究報告，NLC93-53, pp.41–46 (1993).
9) 木本晴夫：日本語新聞記事からのキーワード自動抽出と重要度評価，電子情報通信学会論文誌，J74-D-I, No.8, pp.556–566 (1991).
10) 宮崎正弘：係り受け解析を用いた複合語の自動分割法，情報処理学会論文誌，Vol.25, No.6, pp.970–979 (1984).
11) 宮崎正弘，大山芳史：階層的単語属性を用いた同形語の自動読み分け法，電子情報通信学会論文誌，J68-D, No.3, pp.392–399 (1985).
12) 宮崎正弘，池原 悟，横尾昭男：複合語の構造化に基づく対訳辞書の単語結合型辞書引き，情報処理学会論文誌，Vol.34, No.4, pp.743–754 (1993).
13) 望月久稔，森田和宏，獅々堀正幹，青江順一：拡張ハッシュ法における部分文字列検索の設計と実現，情報処理学会論文誌，Vol.38, No.2, pp.310–320 (1997).
14) 小川泰嗣，望主雅子，別所礼子：複合語キーワードの自動抽出法，情報処理学会自然言語処理研究会，97-15, pp.103–110 (1993).
15) 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，電子情報通信学会論文誌，J77-D-I, No.4, pp.282–289 (1994).
16) 吉村賢治，日高 達，吉田 将：日本語科学技術文における専門用語の自動抽出システム，情報処理学会論文誌，Vol.27, No.1, pp.33–40 (1986).
17) 梅田茂樹，諸橋正幸，細野公男，原田隆史，後藤智範：漢字クラスターによる日本語文献の重要語抽出，情報処理学会自然言語処理研究会，58-5, pp.1–8 (1986).

(平成 11 年 4 月 26 日受付)
(平成 12 年 2 月 4 日採録)



岡田 真（学生会員）

昭和 49 年生．平成 8 年徳島大学工学部知能情報工学科卒業．平成 10 年同大学大学院博士前期課程修了．現在同大学院博士後期課程在学中．情報検索，自然言語処理の研究に従事

安藤 一秋（正会員）

昭和 46 年生．平成 6 年徳島大学工学部知能情報工学科卒業．平成 8 年同大学大学院博士前期課程修了．平成 11 年同大学院博士後期課程修了．平成 10 年日本学術振興会特別研究員（DC），平成 11 年同会特別研究員（PD）．平成 12 年香川大学工学部信頼性情報システム工学科助手，現在に至る．自然言語処理，情報検索の研究に従事．工学博士．電子情報通信学会会員．

森田 和宏（学生会員）

昭和 47 年生．平成 7 年徳島大学工学部知能情報工学科卒業．平成 9 年同大学大学院博士前期課程修了．現在同大学院博士後期課程在学中．情報検索，自然言語処理の研究に従事．

青江 順一（正会員）

昭和 26 年生．昭和 49 年徳島大学工学部電子工学科卒業．昭和 51 年同大学大学院修士課程修了．同年同大学工学部情報工学科助手．現在同大学工学部知能情報工学科教授．この間コンパイラ生成系，パターンマッチングアルゴリズムの効率化の研究に従事．最近，自然言語処理，特に情報検索システムの開発に興味を持つ．著書「Computer Algorithms – Key Search Strategies」，「Computer Algorithms – String Matching Strategies」IEEE CS press．平成 4 年度情報処理学会「Best Author 賞」受賞．工学博士．電子情報通信学会，人工知能学会，日本認知科学会，日本機械翻訳協会，IEEE，ACM，AAAI，ACL 各会員．