トライ構造を用いた共起情報の効率的検索アルゴリズム

森田和宏[†] 望月久稔[††]
山川善弘[†] 青江順一[†]

自然言語辞書に構築される基本語彙は有限であるが，それら基本語の関係を定義することで，膨大な数の関係情報が作り出される．複合語，慣用表現，格関係などもこの関係情報の範疇に属し，これらを基本単語の共起情報と呼ぶ．共起情報を基本単語の並びとして格納すると，記憶効率が非常に悪くなるので，これら関係情報の効率的な記憶検索技法は重要な課題である．本論文では，基本単語からなる共起情報をトライ構造で効率的に記憶検索する手法を提案する．本手法では共起情報を構成する2つの基本単語を1つのトライに登録し，関係情報をトライの葉ノード間のリンク関数で定義する．共起情報の登録による記憶量の増加はこのリンク情報のみとなり，リンク情報もトライに格納する．本手法では，トライのアークを高速にたどる必要があるので，これを $O(1)$ の計算量で実現するダブル配列法を適用する．この結果，共起情報の検索時間は，基本単語数や葉ノード間のリンク数に依存しない一定の計算量となった．約10万語の基本単語に対して，複合語，同音語判定の共起語，格構造辞書などの約100万の関係情報を構築した実験結果より，検索時間は1.2msと一定となること，また記憶量は従来法より1/3に圧縮できることが分かった．

An Efficient Retrieval Algorithm of Collocational Information Using Trie Structures

KAZUHIRO MORITA,[†] HISATOSHI MOCHIZUKI,[††]
YOSHIHIRO YAMAKAWA[†] and JUN-ICHI AOE[†]

Collocational information is very useful for natural language processing systems and it includes compound words, cooccurrency words, verbs and the role of nouns in the case slot, and so on. Collocational information can be constructed by combining basic words infinitely, so it is important to propose a fast and compact structure representing them. This paper presents an efficient data structure by introducing a trie that can define the linkage among leaves. It enables us to decrease the amount of memory required for the same basic words. Theoretical observations show that the worst-case time complexity of retrieving collocational information is a constant, independent of the number of words and linkages. From the simulation results for collocational information, it is shown that the presented method is about 1/3 smaller than that of the competitive methods.

1. はじめに

自然言語処理システムにおける単語の共起情報は有用であり，自然言語解析における曖昧性の解決手段[1),2)]，機械翻訳での訳語の選択[3),4)]，かな漢字変換の同音異義語の決定[5)~7)]，音声認識における候補文字の制約[8)]などに利用されており，それぞれの応用辞書だけでも膨大な共起情報が必要となる．特に，形態素解析の精度向上でも必要となる基本単語☆から造語される長単位語（複合語と呼ぶ）は，文書検索システムの索引語における重要語，機械翻訳の訳語解釈などのように，正確な意味解釈をするうえでも必要不可欠なものであり，その数は膨大なものとなる．この観点より，共起情報を格納する辞書の効率的記憶検索は重要な課題であるといえる．この課題に関連する研究として，複合語の接頭辞と接尾辞を圧縮して格納する手法[9)]が提案されているが，この手法は一般的な共起情報の格納方式を提案したものではない．

以上のように，自然言語処理システムには様々な共起情報による知識辞書が必要であるが，これらの知識辞書の検索の切り口（キー，見出し語）は，やはり形態素表記である．すなわち，抽象的な概念化が行われ

† 徳島大学工学部
Faculty of Engineering, Tokushima University
†† 大阪府立工業高等専門学校
Osaka Prefectural College of Technology

☆ 文章を構成する最小単位の単語，すなわち 形態素表記を示す．

たとしても，人間が認知するうえで概念名も形態素表記に合致する場合が多い．たとえば，“タクシー”や“バス”は，“乗り物”と概念化されるが，この概念名も形態素表記となる．したがって，共起情報を格納する知識辞書は，自然言語処理システムの基本辞書である形態素辞書の見出し語と融合して構築するのが効率的である．さらに，異なる属性の共起情報によるすべての知識辞書を形態素辞書の中に統合化して格納できれば，その利便性は非常に高いものとなる．

本論文では，上記のような知識辞書と形態素辞書の統合化などに必要な様々な付加情報を格納できるよう拡張性を持たせた，効率的な共起情報の記憶検索手法を提案する．本手法では，形態素表記が格納されたトライ上の葉ノード間のリンク関数により共起情報を定義するので，登録による記憶の増加はこのリンク情報のみとなる．また，このリンク情報に複数の関係情報を格納する方式を提案することで，種々の自然言語処理システムへの応用性を高める．

以下，2 章で共起情報の記憶の概要と従来の手法について述べる．3 章では，本手法の検索と更新の形式的アルゴリズムを提案し，4 章では実装するための効率的なデータ構造を提案する．5 章では，提案手法の理論的評価と実験による具体的評価を与え，考察を加える．6 章では，本論文のまとめと今後の課題について触れる．

2. 共起情報と従来の格納手法

2.1 共起情報の概要

本論文で取り扱う共起情報とは，基本単語 X, Y に対する関係情報 α を定義するものであり，(X,Y,α) で表す．この関係情報 α は様々な形態での定義と検索要求があるが，本論文では特に言及せず記号 α_1, α_2, ... 等で表す．また，形態素表記は“ ”で囲む．

（例 1） 以下に共起情報 (X,Y,α) の例を示す．以後，この例をもとに議論する．

(“アメリカ”,“合衆国”,α_4)
(“アメリカ”,“合衆国”,α_5)
(“アメリカ”,“国名”,α_1)
(“カナダ”,“国名”,α_1)
(“カッターシャツ”,“衣類”,α_1)
(“カッター”,“カッターシャツ”,α_6)
(“衣類”,“生地”,α_3)
(“合う”,“衣類”,α_2)
(“合う”,“気候”,α_2)
(“国名”,“気候”,α_3)
(“国名”,“国籍”,α_4) （例終）

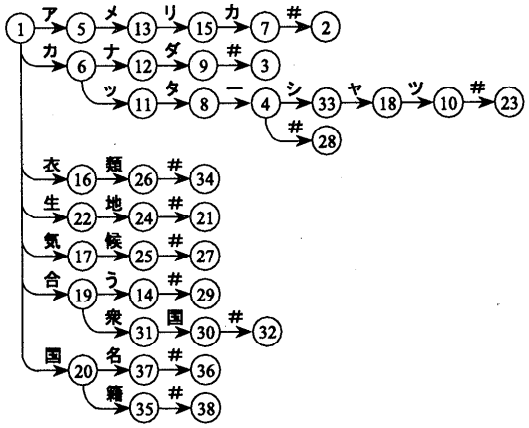


図 1 トライの例
Fig. 1 An example of trie structures.

2.2 トライ構造と従来の格納構造

2.2.1 トライ構造

トライ構造 [10)~12)] はキーの共通接頭辞を併合圧縮した木構造であり，検索はトライのアーク文字単位にたどるので，任意の入力文字列に対して，キーの最長一致検索や接頭辞のみが一致する検索が容易であり，アークを $O(1)$ で検索できれば，キーの数に無関係な高速検索が実現できる．

トライの初期ノードを 1 とし，トライの葉とキーを 1 対 1 に対応させるために，キー自身には含まれない終端記号‘#’をキーの最後につけてトライを構成する．レコード情報は各キーに対応するトライの葉から格納する．ノード s から t へアークラベル a が定義されていることを関数 g で $g(s,a)=t$ と定義し，アークが定義されていない場合は，$g(s,a)=fail$ と記述する．関数 g は文字列 X に対しても使用され，$g(s,X)=t$ と記述する．以後，断りのない限り，文字（記号）は a, b, c, ... を使用し，文字列（記号列）は X, Y, Z を使用する．また，終端記号付きの文字列を $X\#$, $Y\#$, $Z\#$ と表す．

（例 2） 例 1 に対する共起情報 (X,Y,α) の X, Y をキー集合の要素とした場合のトライの構成例を図 1 に示す．図 1 において，“カナダ#”の検索は，$g(1,\text{‘カ’})=6$, $g(6,\text{‘ナ’})=12$, $g(12,\text{‘ダ’})=9$, $g(9,\text{‘\#’})=3$ とたどることで検索でき，$g(1,\text{“カナダ\#”})=3$ となる． （例終）

2.2.2 従来の格納構造

一般に，共起情報の格納方式は基本単語 X と Y に対する連鎖語彙 XY を構成し，この連鎖語彙をキーとして，関係情報をレコード情報としてトライに格納する．しかし，連鎖語彙の作成にともなう平均語彙

長の増加は，共通接頭辞を圧縮するだけでは効率的に記憶されるとはいえない．この問題に関する研究として，ダブルトライ[9]が提案されている．ダブルトライは，2つのトライを使用して前半と後半のキーの部分的な共有化を実現する．この方法では登録キーを他のキーと区別できる接頭辞で区切り，これを最初のトライ（左トライと呼ぶ）に格納する．残りの接尾辞は逆向きに，すなわち，キーの最後の文字がトライの根となるようにもう1つのトライ（右トライと呼ぶ）へ格納する．そして，接尾辞と接頭辞を関係づけるアーク（リンクアークと呼ぶ）を構成する．また，レコード情報は左トライの葉から格納する．

（例3） 例2であげたトライをダブルトライで表現したものを**図2**に示す．左トライのノード4から右トライのノード6への破線アークが，“カナダ#”を左右のトライに対して定義し，左トライのノード4から関係情報のレコードが格納される[9]．このキー“カナダ#”の検索は，$g(1,\text{‘カ’}) = 3$，$g(3,\text{‘ナ’}) = 4$と左トライをたどり，破線アークをたどって右トライを$g^{-1}(6,\text{‘ダ’}) = 2$，$g^{-1}(2,\text{‘\#’}) = 1$とたどることで検索できる．ここで，関数$g^{-1}$は関数$g$の逆関数で$g(s,a) = t$のとき$g^{-1}(t,a) = s$である．（例終）

例3で示したように，ダブルトライはレコード情報を一意に検索できるトライの特性を生かし，かつ接尾辞も可能な限り併合圧縮する手法であり，一般的なキーを対象としているため，膨大な共起情報の格納には通常のトライ同様，効率的ではない．また，ダブルトライの更新は，たとえば，例3において“アフリカ#”を追加すると，右トライのノード4から5へのアークを左トライに移動しなければならないなど，左右トライのアークの部分的な移動，追加，削除が生じるので，取り扱いは複雑になる．ただし，キーの判定とそのレコード情報の検索は，トライのアークの探索時間計算量を$O(1)$で行えるならば，トライに格納される語彙数に関係なく，キーの長さに比例した計算量で実行できる特徴を有する．

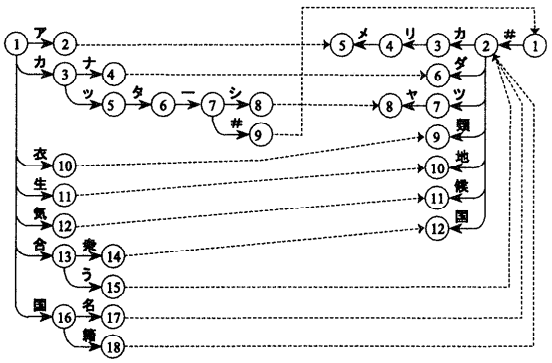


図2 ダブルトライの例
Fig. 2 An example of double-trie structures.

本論文では，ダブルトライのように共起情報をXYの文字列としてとらえるのではなく，X，Yの独立した単語間の関係を，X，Yを完全に共有化した構造で検索できる手法を提案する．

3. 共起情報の格納と検索アルゴリズム

3.1 トライによる新しい格納手法

本手法では，(X,Y,α)に対して，$X\#$，$Y\#$を1つのトライへ格納し，2項関係を定義するために，トライの葉ノード間にリンクを作成する．したがって，本手法でのトライをリンクトライと呼ぶ．葉ノードsから出るリンク情報は関数$f(s)$で定義され，$f(s)$が要素tを含むとき，葉ノードsから葉ノードtへのリンクが存在し，2項関係が定義されていることを意味する．

トライにアーク$g(1,X\#) = s$なるsが存在するとき，葉ノードsはXと1対1に対応するので，キーXに関するレコード情報は葉ノード番号sに対応したレコードに格納できる．そこでリンクトライでは，$f(s) \ni t$となり，ノードsから$g(1,Y\#) = t$なる葉ノードtへのリンクが存在するとき，X，Yの関係情報αをレコード情報の集合$REC(s,t)$の要素として格納するものとする．

（例4） 例1の共起情報をリンクトライに登録した場合のリンク情報を**表1**に示す．トライ部は図1と同様である．（例終）

関数$f(s)$によって定義されるリンク情報は，表1で示したようにXを基準にそのすべてを取得可能である．言い換えれば，(X,Y,α)の検索はもとより，X，Yに定義されるすべてのαの検索や，Xとの関係が定義されるすべてのYの検索が可能となる．これは，基本単語と共起情報を共有化した構造を持つリンクト

表1 共起情報におけるリンク情報の例
Table 1 An example of link information for collocational informations.

X	s	$f(s)$	REC(s,t)
アメリカ	2	{36,32}	$REC(2,36) = \{\alpha_1\}$, $REC(2,32) = \{\alpha_4,\alpha_5\}$
カッター	28	{23}	$REC(28,23) = \{\alpha_6\}$
カッターシャツ	23	{34}	$REC(23,34) = \{\alpha_1\}$
カナダ	3	{36}	$REC(3,36) = \{\alpha_1\}$
合う	29	{34,27}	$REC(29,34) = \{\alpha_2\}$, $REC(29,27) = \{\alpha_2\}$
衣類	34	{21}	$REC(34,21) = \{\alpha_3\}$
国名	36	{27,38}	$REC(36,27) = \{\alpha_3\}$, $REC(36,38) = \{\alpha_4\}$

ライの特徴であり，共起情報をより効率的に記憶する手段でもある．

3.2 共起情報の検索アルゴリズム

共起情報 (X,Y,α) を検索するアルゴリズムを示す．

まず，リンクトライ T からキー X を探索し，$g(1,X\#)=s$ なるノード s を返す関数 Trie_Search (T,X) を先に与える．ただし，X が探索できなかった場合は，$fail$ を返す．

[関数 Trie_Search(T,X)]

$X\#=a_1a_2\ldots a_na_{n+1}$，$a_{n+1}=$'#' と表す．

手順（T-1）：{トライ検索の初期化}

トライ T のノード番号を表す変数 $node$ を初期ノード 1 に，文字位置を表す変数 pos を 1 にセットする．

手順（T-2）：{トライ検索}

$next=g(node,a_{pos})$ なる $next$ が $fail$ ならば，X はトライに格納されていないので，$fail$ を返す．$fail$ でなければ，$node=next$ とし，pos をインクリメントする．

手順（T-3）：{探索の終了と出力の判定}

$pos<n+2$ ならば手順（T-2）へ戻り，$pos=n+2$ ならば X の検索は終了しているので，$node$ を出力する．（関数終）

（**例 5**）図 1 のトライに対するキー "カナダ" を検索する例を示す．まず，手順（T-1）で，$node$，pos を 1 にセットし，手順（T-2）で，$next=g(node,a_{pos})=g(1,$'カ'$)=6$ を得る．$next$ は $fail$ ではないので，$node=6$，$pos=2$ となる．手順（T-3）で，pos は $n+2=5$ ではないので，手順（T-2）に戻る．以後同様に $g(6,$'ナ'$)=12$，$g(12,$'ダ'$)=9$，$g(9,$'#'$)=3$ となり，$pos=5$ のとき，$node=3$ を得る．（例終）

【共起情報の検索アルゴリズム 1】

【入力】：X，Y，α．

【出力】：X，Y に α が定義されていれば，その α を含むレコード $REC(p,q)$ に対応するノード番号 p，q．すなわち，$g(1,X\#)=p$，$g(1,Y\#)=q$．定義されていなければ，$p=0$，$q=0$ が返される．

【方法】

手順（S1-1）：{X，Y のトライ上の検索}

X，Y に対してトライ T を検索し，$s=$ Trie_Search (T,X) と $t=$ Trie_Search(T,Y) を得る．s，t のいずれかが $fail$ ならば，X，Y のいずれかがトライに格納されていないので，$p=0$，$q=0$ を出力し，アルゴリズムを終了する．そうでなければ，次へ進む．

手順（S1-2）：{関係定義の探索と出力処理}

$f(s)\ni t$ かつ $REC(s,t)\ni\alpha$ ならば $p=s$，$q=t$ を出力し，そうでなければ $p=0$，$q=0$ を出力し，アルゴリズムを終了する．（アルゴリズム終）

（**例 6**）共起情報 ("アメリカ","合衆国",α_5) を検索する例を示す．まず，手順（S1-1）で，Trie_Search(T, "アメリカ") = 2，Trie_Search(T,"合衆国") = 32 を得る（図 1 参照）．手順（S1-2）で，$f(2)\ni 32$ かつ $REC(2,32)\ni\alpha_5$ であるので（表 1 参照），$p=2$，$q=32$ を出力する．（例終）

3.3 更新アルゴリズム

リンクトライに対するキーの追加と削除アルゴリズムを示す．

【共起情報の追加アルゴリズム】

【入力】：リンクトライに登録されていない (X,Y,α)．

【出力】：なし．

【方法】

手順（A-1）：{キー X の登録の確認}

$s=$ Trie_Search(T,X) なる s が $fail$ でなければ，次へ進む．$fail$ ならば，X はトライに登録されていないので，$g(1,X\#)=h$ なるアーク列をトライに追加し，$s=h$ とする．

手順（A-2）：{キー Y の登録の確認}

$t=$ Trie_Search(T,Y) なる t が $fail$ でなければ，次へ進む．$fail$ ならば，Y はトライに登録されていないので，$g(1,Y\#)=k$ なるアーク列をトライに追加し，$t=k$ とする．

手順（A-3）：{関係の定義}

$f(s)\ni t$ ならば，$REC(s,t)$ に α を追加し，$f(s)\not\ni t$ ならば，関係は未定義なので，$f(s)$ に t を加えてから，$REC(s,t)$ に α を追加する．（アルゴリズム終）

（**例 7**）共起情報 ("カッター","衣類",α_1) の追加を考える．まず，手順（A-1）で Trie_Search(T, "カッター") = 28，手順（A-2）で Trie_Search (T, "衣類") = 34，手順（A-3）で $f(28)$ には 34 が含まれていないので，$f(28)$ に 34 を加え，$REC(28,34)$ に α_1 を加える．（例終）

【共起情報の削除アルゴリズム】

【入力】：リンクトライに登録されている (X,Y,α)．

【出力】：なし．

【方法】：X，Y に対してトライ T を検索し，$s=$ Trie_Search(T,X) と $t=$ Trie_Search(T,Y) を得る．s と t のどちらかが $fail$ ならば，共起情報は未定義なので終了する．どちらも $fail$ でなければ $REC(s,t)$ から α を削除する．$REC(s,t)$ が空になれば，$f(s)$ から t を削除する．（アルゴリズム終）

（**例 8**）共起情報 ("アメリカ","国名",α_1) の削除を考える．まず，$s=$ Trie_Search(T,"アメリカ") = 2，$t=$ Trie_Search(T,"国名") = 36 を得る．s，t とも

に *fail* ではないので，$REC(2,36)$ から α_1 を削除する．$REC(2,36)$ が空になるので，$f(2)$ から 36 を削除し，$f(2)=\{32\}$ となる． （例終）

4. リンクトライのデータ構造

リンクトライは，キー集合 K を格納するトライ部，葉ノード s に対するリンク情報を格納する関数 $f(s)$，およびキー情報と共起情報を格納するレコード情報 $REC(s,t)$ から構成されるので，これらの個々のデータ構造の効率化を満足したうえで，リンクトライの全体効率が低下しない相互構造の連結手法が必要である．

4.1 トライ部のデータ構造

Trie_Search を高速に行うために，アークの最悪の検索時間計算量を $O(1)$ にでき，コンパクト性も満足するダブル配列構造[10),13)]を適用する．

ダブル配列法では，2 つの配列 BASE，CHECK でトライのアークを表現する．ダブル配列上のインデックス番号はトライのノード番号に 1 対 1 に対応しており，以下簡単のため，両者の値を一致させて説明する．

ダブル配列は $g(s,a)=t$ に対して，次を満足する．

$$t=\mathrm{BASE}[s]+N(a),\quad \mathrm{CHECK}[t]=s$$

すなわち，$g(s,a)$ のノード番号 t は，ノード番号 s に対する BASE[s] とアークラベル a の内部コード値（numerical code）$N(a)$ の和で計算され，CHECK[t] にはノード番号 s から引かれたことを定義する s を格納する．したがって，$g(s,a)$ を確認するための関数 Forward(s,a) は，ダブル配列で使用されている最大のインデックス番号を MAX とすると，次で示される．

```
Forward(s,a)
begin
    t =BASE[s]+N(a);
    if (0 < t < MAX + 1) and (CHECK[t]= s)
      then return(t) else return(0);
end;
```

また，キー集合 K に対するダブル配列 D(K) を使用した関数 Trie_Search(D(K),X) は，次で与えられる．ただし，検索が成功したときの関数の返値は，後の議論のために検索が成功したときのインデックス番号 *index* に対する $-$BASE[*index*]☆とする．

[関数 Trie_Scarch(D(K),X)]

$X\# = a_1a_2\ldots a_na_{n+1}$，$a_{n+1}=$ '#' と表す．

手順（D-1）：{ダブル配列 D(K) による検索の初期化}

ダブル配列 D(K) のインデックス番号を表す変数 *index* を初期インデックス 1 にセットし，X の文字位置を表す変数 *pos* を 1 にセットする．

手順（D-2）：{トライ検索}

next = Forward(*index*,a_{pos}) なる *next* が 0 ならば，X はトライに格納されていないので，0 を返す．0 でなければ，*index* = *next* とし，*pos* をインクリメントする．

手順（D-3）：{探索の終了と出力の判定}

pos $< n+2$ ならば手順（D-2）へ戻り，*pos* $= n+2$ ならば X の検索は終了しているので，$-$BASE[*index*] を出力する． （関数終）

（例 **9**）図 1 に対するダブル配列の例を図 **3** に示す．ここで，キー "合衆国" の検索について考える．まず，*index* と *pos* を 1 にセットする．次に，BASE[1]$+18=19$，CHECK[19] $=1$ より Forward(1,'合') = *index* $=19$，*pos* $=2$ となる．同様に Forward(19,'衆') $=31$，Forward(31,'国') $=30$，Forward(30,'#') $=32$ となり，*pos* $=5$ のとき，$-$BASE[32] $=9$ となる． （例終）

4.2 リンク関数 $f(s)$ のデータ構造

$f(s)$ の検索は，ノード番号集合から指定されたキー番号を探索する問題であるので，種々のキー検索技法が適用できる．1 つの方法として，番号集合をソートして配列に格納し，2 分探索を適用するならば，配列の長さ n に対して検索時間計算量は $O(\log_2 n)$ となる．しかし，本手法では高速検索の実現を目標にしているので，ここにも $f(s)$ に対する個別のダブル配列 D($f(s)$) を次のように導入する．

$f(s)$ の要素ノード番号 t の内部コード列 w と $REC(s,t)\ni\alpha$ なる α の内部コード列 v に対し，vw なるコード列をトライのアークとするダブル配列 D($f(s)$) を構成する．ただし，vw から v，w を一意に決定するために，v，w のそれぞれの列長は固定する．

（例 **10**）表 1 において，キーが "国名" の場合の $f(36)$ に対するダブル配列 D($f(36)$) の例を図 **4** に示す．ここでは簡単のために，$f(36)$ の要素を数値文字列として表現し，各関係情報 α_i に対する内部表現値は添字 i を使用する．コード列長はそれぞれ 2，1 とする．また，D(K) と区別するために F_BASE，F_CHECK としている．たとえば $f(36)$ の要素 38 は，$REC(36,38)\ni\alpha_4$ より，$v=$ "4"，$w=$ "38"，$vw=$ "438" として D($f(36)$) に格納される．このダブル配列の検索は，Trie_Search(D($f(36)$),"438") $=-$F_BASE[9] $=2$ のように，関数 Trie_Search で検索できる．ただし，vw の列長は固定のため，トライの葉とキーはつねに 1 対 1 となるので，終端記号

☆ 通常のノードと葉ノードを区別するために葉ノードの BASE 値を負数とするので，符号を変える．

文字	#	ー	う	ア	カ	シ	タ	ダ	ツ	ッ	ナ	メ	ャ	リ	衣	気	候	合	国	衆	生	籍	地	名	類
内部表現値	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
BASE	1	-1	-2	27	1	1	1	2	2	22	1	1	1	28	2	1	8	1	11	13
CHECK	38	7	9	8	1	1	15	11	12	18	6	6	5	19	13	1	1	33	1	1

	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
BASE	-6	1	-3	20	26	33	-7	-4	-8	31	11	-9	5	-5	37	-10	35	-11
CHECK	24	1	10	22	17	16	25	4	14	31	19	30	4	26	20	37	20	35

図 3 図 1 に対するダブル配列の例
Fig. 3 An example of the double-array for Fig. 1.

	1	2	3	4	5	6	7	8	9
F_BASE	1	0	1	1	3	1	0	-1	-2
F_CHECK	9	0	4	1	1	5	0	3	6

図 4 ダブル配列 D($f(36)$) の例
Fig. 4 An example of the double-array D($f(36)$).

'#' は存在しない. (例終)

4.3 トライ部とリンク関数 $f(s)$, $REC(s,t)$ の連結と更新処理

静的な D(K) では, D(K) のインデックス番号 s を手掛かりに D($f(s)$) がアクセスできるが, 動的な D(K) では s は更新される可能性があるので, インデックス s は使用できない. 解決法として, D($f(s)$) を格納する領域へのポインタ p を利用して, s からポインタ p を得るためのポインタ表 PTR[s] $= p$ を構成すれば, トライ部の D(K) とリンク関数の D($f(s)$) の連結は成功する. しかし, トライの葉ノード s が変更される場合, s の値は D(K) の最大インデックス番号までの任意の値をとるので, 最大インデックス番号に対するポインタ表が必要となり, 余分な記憶領域や管理処理が必要となる.

さらに, D(K) のインデックス番号の変更は $f(s)$ の要素 t にも関係するので, s, t に関係するポインタ表の更新だけでなく, t を格納するすべての D($f(s)$) の更新を余儀なくされ, リンクトライ全体の更新効率を低下させる深刻な問題となる.

しかし, $f(s)$ を定義するノード s とその要素 t は, すべてトライの葉ノードであり, しかも各葉ノードはキーと 1 対 1 に対応するので, 葉ノード番号の代わりにキーに対するユニークな識別番号を $f(s)$ の構成に利用すれば, 上記の問題は解決する. 不変な識別番号として, 最も適切なものはキー X のレコード情報の実体へのポインタ値 r であり, これは必然的にトライの葉ノード s から $f(s)$ の実体へのポインタ値 r と考えても同じ意味である. しかし, 実体のデータ領域をアクセスするポインタ値 r も更新されるので, キー X の識別番号 p とこのポインタ値 r を格納する表 KEYID[p] $= r$ を構成し, 表 KEYID のインデックス番号 p をキーの識別子として定義する. 表 KEYID は, 上記の表 PTR のように最大のインデックス番号に対応した大きさにならず, キーの数に比例した大きさで管理できる.

この識別番号を格納するにあたって, ダブル配列 D(K) は次のような有利な特徴を有する.

- トライの葉ノードに対応するインデックス番号 s において, ダブル配列の BASE[s] は値を定義しないので, BASE[s] に識別番号 p を格納できる.
- ダブル配列の更新では, インデックス番号 s が s' に変更されても BASE[s] $= p$ は BASE[s'] $= p$ として連動して変更されるので, 識別番号 p はインデックス番号の変更の影響を受けない.

説明を簡単にするために, BASE[s] $= p$ なる識別番号 p でのリンク関数 $f(s)$ のダブル配列を KEYID[p] $= r$ なるポインタ値 r に対して, DF[r] で参照できるものとし, DF[r] での F_BASE[t] $= u$ なるポインタ値 u に対するレコード情報を $REC(r,u)$ と定義する. また, DF[r] に対する検索関数 Trie_Search(DF[r],X) に限り, 手順 (D-1) での変数 $index$ を r で初期化するものとする.

(**例 11**) **図 5** に KEYID と DF[r] の例を示す. 図 5 では簡単のために, 図 4 と同様のコード列を用いている. また, r の値は KEYID に対応している. ここで, KEYID[10] $= 44$, DF[44] に対するコード列 "411" の検索を考える. まず, $index$ は 44 に, pos を 1 にセットする. 次に, F_BASE[44] $+ 4 = 48$, F_CHECK[48] $= 44$ より, Forward(44,'4') $= 48$ となる. 同様に Forward(48,'1') $= 46$, Forward(46,'1') $= 49$ となり, $pos = 4$ のとき, $-$F_BASE[49] $= 2$ となる. (例終)

以上のデータ構造による共起情報の検索アルゴリズムを次に示す.

【共起情報の検索アルゴリズム 2】
【入力】: X, Y, α.
【出力】: X, Y に α が定義されていれば, その α を含

	1	2	3	4	5	6	7	8	9	10	11
KEYID	1	12	16	22	29	0	0	36	0	44	0

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F_BASE	1	2	4	-1	7	8	1	2	0	-2	-2	12	13	15	-1
F_CHECK	11	1	2	3	1	1	5	6	0	6	8	15	12	13	14

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
F_BASE	16	18	16	0	0	-1	22	22	0	-1	0	0	23	29	29	0	30	0	0	-1
F_CHECK	21	16	17	0	0	18	28	28	0	23	0	0	22	35	32	0	29	0	0	30

	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
F_BASE	36	36	37	0	0	-1	0	-2	44	44	48	45	45	-2	0	-1
F_CHECK	43	38	36	0	0	37	0	37	51	47	48	44	44	46	0	45

図 5 ダブル配列 DF[r] と表 KEYID の例
Fig. 5 An example of the double-array DF[r] and table KEYID.

むレコード $REC(r,u)$ に対応するポインタ値 r, u. 定義されていなければ, $r=0$, $u=0$ を返す.

【方法】

手順 (S2-1):¦トライ部のダブル配列 D(K) における X, Y の検索¦

X, Y に対してトライを検索し, キー X に対する識別番号 p = Trie_Search(D(K),X) と Y に対する識別番号 q = Trie_Search(D(K),Y) を得る. p, q のいずれかが 0 ならば, $r=0$, $u=0$ を出力し, アルゴリズムを終了する. そうでなければ, 次へ進む.

手順 (S2-2):¦リンク関数とレコード情報の検索¦

X の識別番号 p で指定されるリンク関数の KEYID[p] = r なるダブル配列表現 DF[r], キー Y における識別番号 q のバイトコード列 w, および α のバイトコード列 v に対して u = Trie_Search(DF[r],vw) を検索し, r, u を出力する. (アルゴリズム終)

(例 12) 図 5 において, 共起情報 ("アメリカ", "合衆国",α_4) の検索を考える. まず, 手順 (S2-1) で Trie_Search(D(K),"アメリカ") = −BASE[2] = 1, Trie_Search(D(K),"合衆国") = −BASE[32] = 9 を得る (図 3 参照). 手順 (S2-2) で KEYID[1] = 1 より, コード列 "409" に対して Tric_Search(DF[1],"409") = −F_BASE[10] = 2 であるので, $r=1$, $u=2$ を出力する. (例終)

例 12 で示した $REC(r,u) = REC(1,2)$ は, 表 1 での $REC(2,32)$ に対応する. よって, 共起情報 ("アメリカ","合衆国",α_5) の検索においても, コード列 "509" に対して Trie_Search(DF[1],"509") = −F_BASE[11] = 2 となり, $REC(1,2)$ を返していることが分かる.

5. 評価と考察

5.1 理論的評価

ダブル配列によるトライのアーク検索の最悪時間計算量は $O(1)$ であるので, X, Y の共起情報を検索する最悪時間計算量は, X と Y の長さとキーの識別番号のバイト長に比例したものとなり, キーの総数とキー X に定義された関係情報数に依存しない非常に高速なものとなる. ダブル配列への追加の最悪時間計算量は, トライのアークラベルの最大数を e, ダブル配列の未使用インデックス番号の数を m とするとき, $O(e\cdot(m+e))$ となる [13] が, 未使用数 m は非常に小さい値となるので, $O(e^2)$ で近似できる. この処理は主記憶上で行われ, 十分実用に耐える速度であることが報告されている [13]. したがって, m が非常に小さいことより, ダブル配列の記憶量はトライのノード数に比例し, 十分にコンパクトであることが分かる.

4 章で導入したキーの識別番号 p に対する KEYID と INDEX の大きさは, キーの総数に比例したものとなり, ダブル配列のインデックス総数のように大きくならない. また, ダブル配列 D(K) で BASE[s] = p が BASE[s'] = p と変化したときでも, KEYID の内容の変更は不要であり, また INDEX[p] = s を s' に変更する時間計算量は $O(1)$ であるので効率的である.

自然言語処理システムでの辞書形態は, 多くの項目が登録された基本辞書が利用されるので, 更新速度よりは検索速度とコンパクト性が重要であり, この観点より, 提案手法は有効であるといえる.

5.2 具体的評価

本手法の構成システムは約 3,000 行の C++言語で記述されており, DELL OptiPlex GXpro (CPU : pentium pro [180 MHz]) 上で稼動している.

本手法の有効性を確認するため, 対象手法としてダブル配列を利用したダブルトライ [9] と, 通常のトライを用いて共起情報を登録したものとの比較を行った. 実験に用いたキー集合は, キー集合中に含まれる基本単語の語数の異なる 5 種類の漢字共起語集合 $KK1$ から $KK5$ である. 漢字を含む日本語文字は 1 文字に

表 2 実験結果
Table 2 The simulation results.

	$KK1$	$KK2$	$KK3$	$KK4$	$KK5$
キー総数	4,499	9,736	14,189	55,660	69,849
平均長 (byte)	8.5	7.3	10.0	10.0	10.0
最大長 (byte)	22	22	10	10	10
基本単語数	2,066	8,015	19,518	13,337	28,360
総状態数					
リンクトライ	14,764	40,257	61,491	57,150	165,624
ダブルトライ	22,552	70,875	112,413	200,548	283,257
従来のトライ	28,685	73,543	143,085	416,988	379,600
総リンク数	4,495	6,810	5,564	52,527	60,230
最大リンク数	2,051	206	102	366	390
検索時間 (ms)					
リンクトライ	1.0	1.0	1.3	1.2	1.2
ダブルトライ	0.9	1.0	1.2	1.2	1.2
従来のトライ	0.9	1.0	1.1	1.0	1.0
追加時間 (ms)					
リンクトライ	2.2	5.0	11.7	8.3	14.3
ダブルトライ	2.3	5.8	13.0	9.1	16.1
従来のトライ	2.0	2.0	7.2	6.0	10.2
削除時間 (ms)					
リンクトライ	1.2	1.3	1.5	1.1	1.2
ダブルトライ	1.3	1.3	1.4	1.1	1.2
従来のトライ	1.0	1.2	1.4	1.0	1.1

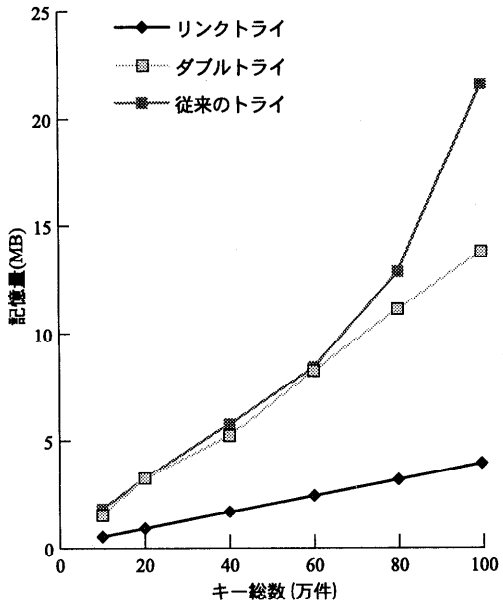


図 6 記憶量の実験結果
Fig. 6 The simulation results for strages.

つき 2 バイトで構成されているが，本実験では遷移のラベルを 1 バイト単位としているので，1 文字の日本語文字から，連続する 2 つの遷移が構成される．

表 2 に実験結果を示す．ここで，検索時間は登録済みの全件検索を，更新時間は未登録の 1,000 件の追加，1,000 件の削除を行った場合の 1 件あたりに要する時間である．

リンクトライの総状態数は，従来のトライに比べて約 14 ～ 55%，ダブルトライと比べても約 28 ～ 65%に減少している．これは，従来のトライでの接尾辞や，ダブルトライでの基本単語の冗長な登録がリンクトライでは一度の登録ですむためである．

検索，更新時間については，従来手法とほぼ対等であるといえ，検索時間は約 1.2 ms と高速で一定となり，更新時間も十分高速であることが分かる．追加時間にばらつきが見られるが，これは理論的評価で示したダブル配列の追加時間計算量 $O(e^2)$ の近似に対するゆれである．

約 10 万語の基本単語から生成した 10 万～ 100 万件の共起情報集合に対する記憶量を**図 6** に示す．共起情報が 100 万件に近くなると，リンクトライの大きさは従来のトライやダブルトライよりも 1/3 以上コンパクトになり，本手法の有効性が分かる．

6. おわりに

共起情報を効率的に格納するために，トライ構造を用いた辞書構造を提案し，それをダブル配列で実現する手法を提案した．また，2 つの基本単語からなる共起語の共起情報に対して実験結果により空間的および時間的有効性が実証された．

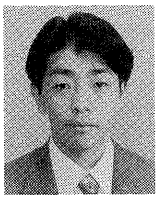
(X, Y, α) に対して，現在では，キー Y が与えられた場合の関係情報の検索ができないので，これを実現するリンク部分の格納構造の考案が今後の課題である．

参考文献

1) 奥村 学，田中穂積：自然言語解析における意味的曖昧性を増進的に解消する計算モデル，人工知能学会誌，Vol.4, No.6, pp.687–694 (1989).
2) 高橋直人，板橋秀一：単語共起頻度を利用した形態素解析，情報処理学会研究報告，88-NL-69-5 (1988).
3) 高松 忍，西田富士夫：動詞パターンと格構造に基づく英日機械翻訳，信学論 (D)，Vol.J64-D, No.9, pp.815–822 (1981).
4) 中岩浩巳，池原 悟：日英翻訳システムにおける用言意味属性を用いたゼロ代名詞照応解析，情報処理学会論文誌，Vol.34, No.8, pp.1705–1715 (1993).
5) 大島義光，阿部正博，湯浦克彦，武市宣之：格文法による仮名漢字変換の多義解消，情報処理学会論文誌，Vol.27, No.7, pp.679–687 (1986).
6) 牧野 寛，木澤 誠：べた書き文のかな漢字変換システムとその同音語処理，情報処理学会論文誌，Vol.22, No.1, pp.59–67 (1981).
7) 山本喜大，久保田淳市：共起グループを用いた

かな漢字変換，第44回情報処理学会全国大会論文集，No.3, pp.189–190 (1992).
8) 澗潟謙一，荒木健治，宮永喜一，栃内香次：表層より得られる単語共起情報とその評価，情報処理学会研究報告，90-NL-80-2 (1990).
9) 森本勝士，入口浩一，青江順一：2つのトライを用いた辞書検索アルゴリズム，信学論（D–II），Vol.J76-D-II, No.11, pp.2374–2383 (1993).
10) 青江順一：キー検索技法—トライ法とその応用，情報処理，Vol.34, No.2, pp.244–251 (1993).
11) Fredkin, E.: Trie memory, *Comm. ACM*, Vol.3, No.9, pp.490–500 (1960).
12) Knuth D.E.: *The Art of Computer Programing*, Vol.3, Sorting and Searching, Ch.6, Addison-Wesley, Reading, MA (1973).
13) 青江順一：ダブル配列による高速ディジタル検索アルゴリズム，信学論（D），Vol.J71-D, No.9, pp.1592–1600 (1988).

(平成 9 年 11 月 13 日受付)
(平成 10 年 7 月 3 日採録)

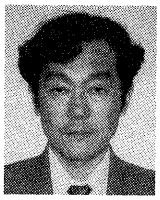


森田 和宏（学生会員）

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．

望月 久稔（正会員）

昭和44年生．平成5年徳島大学工学部知能情報工学科卒業．平成7年同大学院博士前期課程修了．平成10年同大学院博士後期課程修了．工学博士．同年大阪府立工業高等専門学校講師，現在に至る．情報検索，自然言語処理の研究に従事．電子情報通信学会会員．



山川 善弘（正会員）

昭和29年生．昭和52年徳島大学工学部電子工学科卒業．昭和55年同大学院修士課程修了．同年日本電気（株）入社．交換ソフトウェアCASEの研究開発に従事．平成2年同社退社．同年（株）ジャストシステム入社．一太郎UNIX化，インターネットサービスの研究開発に従事，平成10年同社退社．同年（株）富士通徳島システムエンジニアリング入社，現在徳島大学大学院システム工学専攻博士後期課程在学中．検索の研究に従事．電子情報通信学会会員．

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，特に理解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．