DFAによる形態素解析の高速辞書検索

森 信介
京都大学工学研究科

概要

入力文を単語に分割し品詞を付加する形態素解析は，日本語処理における基本的な処理である．日本語には単語間に明確な区切り記号がないので，この処理は入力文の全ての部分文字列に対する辞書検索を含む．本論文では，辞書を決定性オートマトンに変換し，辞書検索を高速に実現する方法を提案する．この方法は，AC法(失敗関数を持つトライ)に基づく方法と比較して，計算時間が少ないという利点と，大きい記憶域を必要とするという欠点がある．これらの方法を実装し実験を行なった結果，決定性オートマトンによる方法はAC法に基づく方法に対して，必要な記憶域は16.1倍であり，辞書検索の速度は1.21倍であった．

1 はじめに

入力文を単語に分割し品詞を付加する形態素解析は，日本語処理における基本的な処理である．このため，今日までにさまざまな研究がなされ，標準的な解析方法が確立するとともに，実用に耐え得る解析精度が達成されている．その結果，情報検索や自然言語の機械学習の研究の多くに形態素解析が使われている．形態素解析の速度向上は，より高速な情報検索の実現や，より大きいコーパスを用いた機械学習や知識獲得の研究に有効であると考えられる．

標準的な形態素解析の処理では，入力文の各位置から始まる任意の長さの部分文字列が辞書に含まれるか否かを調べ，それぞれの部分文字列の間の接続コストや接続確率を調べ，すべての可能な解の中で確率やコストの基準で最適な解を入力文の解析結果とする．これらの処理のうち本論文で問題とするのは，辞書検索である．

辞書検索の標準的方法は，トライを用いる方法[1]である．この方法は，各位置から始まる部分文字列はすべてその中の最長の文字列の接頭辞であるという性質を利用しており，入力文の各位置において1回の辞書検索でその位置から始まる全ての形態素を出力する．この方法の他の利点として，辞書を記憶しておくために必要な記憶域が少なくてすむことが挙げられる．

丸山[2]は，トライを用いる方法では複数回読まれる文字があるという無駄を指摘し，AhoとCorasick[3]が提案した文字列マッチングの方法を形態素解析の辞書検索に応用し高速化を実現している．これはトライの各ノードに失敗関数を持たせることで実現される．この方法の記憶域はトライと同程度であり，速度は理論的にトライを上回ることが保証されているので，トライを用いる方法より優れている．以下では，この方法をAC法と呼ぶ．

本論文では，トライを決定性オートマトンに変換することでAC法よりも高速な辞書検索を実現する方法（以下DFA法）を提案する．DFA法は，AC法と比較して，計算時間が少ないという利点と，大きい記憶域を必要とするという欠点がある．従って，どちらの方法が良いかは計算機資源などの制約に依存する．本論文では，これらの方法を計算時間と記憶域という点で比較する．

速度の点では，辞書検索の部分を切り離して考える限り，入力文字の参照回数という観点では，DFA法よりは速くならないことが以下のようにして示される．文字列マッチングの問題では，少なくとも入力文字列の長さの亜線形の時間[1]が必要であることが証明されている[4]．形態素解析においては，1文字からなる形態素が辞書に含まれていると仮定して良いので，全ての文字を最低でも1回以上読む必要がある．本論文で述べるDFA法は全ての文字を1回しか読まないので．形態素解析の辞書検索においてDFA法は理論的に最も高速なアルゴリズムである．

以下の章では，まずAC法に基づく方法について述べる．次に，DFA法に基づく方法について説明する．さらに，それぞれの長所と短所を実験の結果とともに論じる．最後に，本研究の結論と今後の課題を述べる．

[1] 計算時間が入力の大きさ比例し，その係数が1未満であること．

2 AC法に基づく方法

この章では，高速な形態素解析のための辞書構造として丸山 [2] が提案した方法（AC法）における辞書の構成と検索について述べる．

2.1 辞書構造の概要と検索方法

AC法による辞書は，次の6項組からなる出力付きのオートマトンである．

1. Q は状態の有限集合
2. Σ は入力アルファベットの有限集合
3. g は $Q \times \Sigma$ から $Q \cup \{\text{fail}\}$ への写像（前方関数）
4. h は Q から Q への写像（失敗関数）
5. q_0 は初期状態
6. F は Q から形態素情報への写像（出力関数）

このオートマトンの状態は登録語の表記から得られるトライのノードに対応している．初期状態はルートノードに対応する．前方関数 g は，各ノードから子ノードへの写像であり，1文字を読み込んだ後の遷移先を与える．失敗関数 h は，前方関数 g が定義されていない（g=fail）場合の遷移先を与える．辞書検索のときは，まず1文字読み込み前方関数 g が定義されている状態に到達するまで失敗関数 h で0回以上遷移したあと，前方関数 g のその文字に対する値へ遷移する．ただし，ルートノードに対応する状態では，前方関数が定義されていなければ遷移を終了する．その後，出力関数 F で与えられる形態素情報を出力する．出力される形態素は，初期状態からの経路として与えられる文字列のすべての接尾辞を表記としてもつ形態素の情報である．これは，確率的形態素解析では，文字数と品詞番号と頻度の組である．状態遷移の方法から状態遷移の回数は，読み込む文字数の2倍を越えないことが分かる．つまり，前方関数による遷移はノードの深さを1増やし，失敗関数よる遷移は，ルートノードの場合を除いてノードの深さを1以上減らすことと，ノードの深さは常に非負であることを用いてならし解析 [5] をおこなうと，1文字あたりの平均遷移回数は2以下となる[2]．

たとえば，図1のようなトライから図2のようなAC法の辞書が構成される．AC法の辞書では，トライにノード間の親子関係を表わす実線が前方関数 g に対応

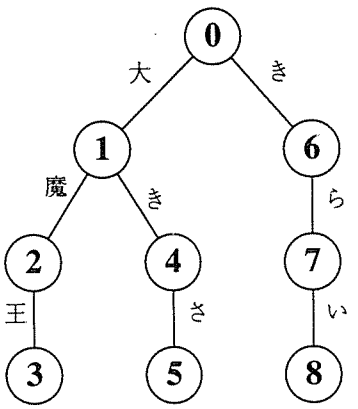


図1: トライを用いた辞書の構造

しする．これに加えて，点線で与えられている失敗関数 h が各ノードに与えられる．状態3の出力関数は，「大魔王」と「魔王」と「王」がそれぞれ名詞であるとすると，$F = \{(3, \text{名詞}), (2, \text{名詞}), (1, \text{名詞})\}$ である．形態素解析のときには文脈が与えられているので，長さから表記が復元できることに注意しなければならない．この例の辞書で文字列「大きらい」に対して辞書検索を行なうと，初期状態から次ような状態遷移と出力を行なう．

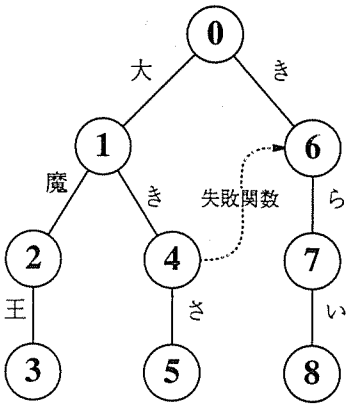
$$
\begin{aligned}
&g(q_0, \text{大}) = q_1, \quad && F(q_1) = \{(1, \text{接頭辞})\},\\
&g(q_1, \text{き}) = q_4, \quad && F(q_4) = \{(2, \text{形容詞})\},\\
&g(q_4, \text{ら}) = \text{fail}, \quad && h(q_4) = q_6,\\
&g(q_6, \text{ら}) = q_7, \quad && F(q_7) = \{(1, \text{接尾語})\},\\
&g(q_7, \text{い}) = q_8, \quad && F(q_8) = \{(3, \text{形容動詞})\}.
\end{aligned}
$$

トライを用いる方法と異なり，出力される形態素は最後に読み込んだ文字の位置で終る形態素の集合である．

実装に際しては，前方関数 g はこれが定義されている文字コードの数と失敗関数 h の値の後に文字コードと遷移先の対の列を付加することで実現した．従って，次の文字の遷移先は，まず前方関数が定義されている文字コードの数と失敗関数の値を読み込み，次の文字を二分探索で探す[3]．これが見つかった場合には，その直後の値を読みこれを遷移先とする．見つからなかった場合には，失敗関数の値を遷移先とする．同じことを，二分探索木で実現することも可能である．この場合には，必要

[2] この考察は，情報処理学会自然言語処理研究会における松本裕二氏のコメントに基づく．

[3] 丸山宏氏によると，文献 [2] では線形探索を用いた結果を報告している．これは，トライに対するAC法の優位性を示すことが目的であるためと考えられる．



ノード4以外の失敗関数は省いてある。

図 2: AC法の辞書の構造

となる記憶域がより大きくなるという欠点と探索に必要な時間がより少なくなるという利点がある.

2.2 辞書の構築

まず，与えられた形態素の表記の集合に対してトライ[6]を作成する．これによって，前述の6項の関数 h, F 以外の項が決定される．出力関数 F の値は，そのノードに対応する文字列の接尾辞のそれぞに対して表記が一致する形態素の情報の集合である．失敗関数 h の値は，以下のようにして計算される．ただし，$\mathrm{depth}(q)$ はノード q の深さを返す関数とする.

$h(q_0) := q_0$
foreach q $(\mathrm{depth}(q) = 1)$
　$h(q) := q_0$
foreach d $(1, 2, \cdots)$
　foreach q $(\mathrm{depth}(q) = d)$
　　foreach p $(g(q, \exists c) = p)$
　　$h(p) := f(h(q), c)$

関数 $f(q, c)$ の定義は以下の通り.

$$f(q,c) = \begin{cases} h(q) & (g(q,c) \neq \mathrm{fail}) \\ f(h(q),c) & (g(q,c) = \mathrm{fail} \wedge q \neq q_0) \\ q_0 & \text{その他の場合} \end{cases}$$

3 DFAに基づく方法

この章では，形態素解析のためのトライ辞書を決定性オートマトンに変換する方法とその検索について述べる.

3.1 辞書構造の概要と検索方法

DFA法による辞書は，次の5項組からなる出力付きのオートマトンである.

1. Q は状態の有限集合
2. Σ は入力アルファベットの有限集合
3. δ は $Q \times \Sigma$ から Q への写像（遷移関数）
4. q_0 は初期状態
5. F は Q から形態素情報への写像（出力関数）

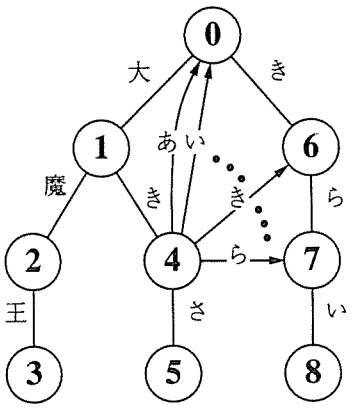
遷移関数 δ 以外の4項は，AC法とまったく同じである．遷移関数 δ は，各状態で1文字読み込んだ後の遷移先を与える．AC法との違いは，状態とアルファベットの全ての組み合わせに対して遷移関数 δ が定義されていることである．AC法における，失敗関数 h による0回以上の遷移と前方関数 g による1回の遷移が，これによる1回の遷移に対応する．出力関数 F の定義はAC法の場合と同じである.

例えば，図1のようなトライに，子ノードへのリンクのラベル以外の全てのアルファベットに対して遷移先を矢印のように決定することで，図3のオートマトンが得られる．この例の辞書で文字列「大きらい」に対して辞書検索を行なうと，初期状態から次ような状態遷移と出力を行なう.

$$\delta(q_0, \text{大}) = q_1, \quad F(1) = \{(1, \text{接頭辞})\},$$
$$\delta(q_1, \text{き}) = q_4, \quad F(4) = \{(2, \text{形容詞})\},$$
$$\delta(q_4, \text{ら}) = q_7, \quad F(7) = \{(1, \text{接尾語})\},$$
$$\delta(q_7, \text{い}) = q_8, \quad F(8) = \{(3, \text{形容動詞})\}.$$

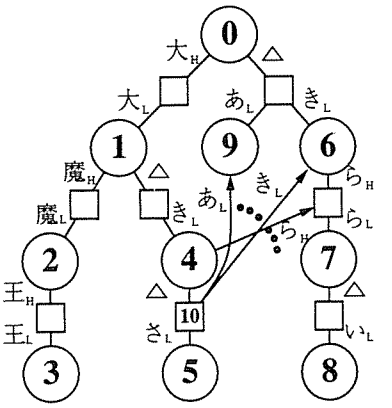
AC法と同様，出力される形態素は最後に読み込んだ文字の位置で終る形態素の集合である.

この例からも分かるように，DFA法における状態遷移はどの入力文字に対しても必ず1回であり，出力関数に関係する部分はAC法と全く同じなので，AC法よりも高速に辞書検索が行なえる．この点に加えて，どの入力文字に対しても遷移関数が定義されていることから，これをアルファベットを添字とする配列を用いて実現できるので，遷移先の参照がAC法と比べて高速に行なわれる.



ノード4以外に付加される矢印は省いてある。

図 3: DFA 法の辞書の構造



△ = $あ_H$ = $い_H$ = $き_H$ = $さ_H$

ノード4と10以外に付加される矢印は省いてある。

図 4: アルファベットを分割する DFA 法の辞書の構造

3.2 辞書の構築

AC 法における辞書の構築と同様に，まず与えられた形態素の表記の集合に対してトライを作成する．これによって，前述の 5 項のうち，関数 δ, F 以外が決定される．出力関数 F の作成は AC 法の場合と全く同じである．関数 δ の値は，以下のようにして計算される．ただし，string(q) は状態 q に対応する文字列を返す関数（1 対 1 写像）であり，findLS(s) は文字列 s のトライに対応する状態をもつ最長の接尾辞を返す関数とする．

```
foreach q (q ∈ Q)
    foreach c (c ∈ Σ)
        s := findLS(string(q) · c)
        δ(q, c) := string^{-1}(s)
```

日本語のように文字数が多い言語にこの方法をこのまま適用すると，各ノードの遷移関数の記述長が非常に長くなり，結果として膨大な記憶域が必要となる恐れがある．実際，次章で述べる実験で用いたコーパスに出現する 92,648 個の異なり形態素をトライに変換すると，そのノード数は 156,917 であった．アルファベットの大きさを 10,000 とし，ノードの識別に 4[byte] 必要であるとすると，全てのノードに対する遷移関数の記述に必要な記憶域は $156,917 \times 10,000 \times 4$[byte] ≈ 5.85[Gbyte] となる．この大きさは近い将来に現実的となると考えられるが，現在の計算機環境を考えると現実的とは言い難い．この問題を回避するために文献 [7] に，アルファベットを別のアルファベットの組合せとして表わす方法が紹介されている．例えば，日本語の文字コードを上位バイトと下位バイトに分割して 100 個のアルファベットで表わすと，状態数は最悪の場合 2 倍になるが，各ノードの遷移関数の記述に必要な記憶域が 1/100 になるので，遷移関数全体の記述に必要な記憶域の上限は $156,917 \times 2 \times 100 \times 4$[byte] ≈ 119.7[Mbyte] となり，現在の計算機環境で実現可能な大きさとなる．これにより，図 3のオートマトンは図 4のようになる．この変更は，必要な記憶域を減少させる効果がある反面，1 文字の入力に対して 2 回の遷移が必要になるため，辞書検索の速度が低下する点に注意しなければならない[4]．次の章で述べる実験では，この改良を採用している．

4 実験結果とその評価

この章では，2章で述べた AC 法と 3章で述べた DFA 法を実装し，辞書検索や形態素解析の時間的・空間的性能について実験した結果を提示し，その評価を与える．

4.1 実験環境

実験には品詞 bigram を用いた確率的形態素解析システムと EDR コーパス [8] を用いた．これを学習コーパス（103,901 文，2,552,332 形態素）とテストコーパス（103,901 文，2,552,715 形態素）に分割した．空間的

[4] 逆に，複数の文字を一つの文字とみなしてアルファベットを再構成することで，必要な記憶域を増加させるという代償の下に辞書検索の速度を高めることができる．

性能は，学習コーパスから得られる辞書の記述に必要な記憶域で評価し，時間的性能はテストコーパスの辞書検索や形態素解析に要する時間で評価した．ただし，頻度2以上の形態素のみ辞書に記述した．実験に用いた計算機は，SPARC Station 20（クロック75MHz，主記憶512[Mbyte]）である．辞書の構築に使用したプログラミング言語はPerl5であり，辞書検索や形態素解析に使用したプログラミング言語はC++である．

形態素解析の解探索の方法は，ダイナミックプログラミングを用いる標準的な方法である[9]．これに用いる表の横方向は入力文中の位置に対応し，縦方向は品詞に対応する（図5参照）．表の各要素はトレリスのノードに対応しており，表中の位置で示された文字と品詞で終る形態素の中で最大の確率を与える形態素と，探索のために便宜的に与えられたポインターを2つ持っている．1つは，そのノードへの最尤の経路における一つ前のノードを指しており，表を埋めた後の最尤の経路の探索に用いられる．もう1つは，その位置で終る他の品詞のノードを指しており，このポインターをたどることで，この位置での接続確率のチェック対象を存在する品詞だけに限定することが可能になる．図5の例の場合，入力文の1文字目で終る形態素の中で最尤経路の探索の対象となるのは，表の1文字目の欄を上から順にポインターをたどることで，「十/数字」と「十/名詞」だけであると分かる．

テストコーパスの解析には，未知語（学習コーパスに現れなかった形態素）の処理が必要である．一般的に，日本語の形態素解析の場合は，文字種などの情報を用いたパターンマッチングのような方法を使う．本研究で用いた未知語モデルは，1つの記号を品詞が「記号」である形態素とみなし，任意長の数字の連接を品詞が「数字」である形態素とみなし，アルファベット，平仮名，片仮名，漢字のそれぞれの任意長の連接を品詞が「名詞」である形態素とみなす．この未知語モデルの実装は，文字種に対応する状態をもつオートマトンと，同じ文字種が何文字連続したかを記憶するカウンタから構成されている．この未知語モデルは，記号を除く同一文字種からなる文字列に対して，全ての部分文字列を形態素とみなすので，部分的に既知語を含む同一文字種からなる文字列（特に片仮名列や漢字列）の解析に有効であると考えられるが，膨大な数の未知語が接続を調べる対象となるため，大幅な速度低下の原因になる．解析速度と

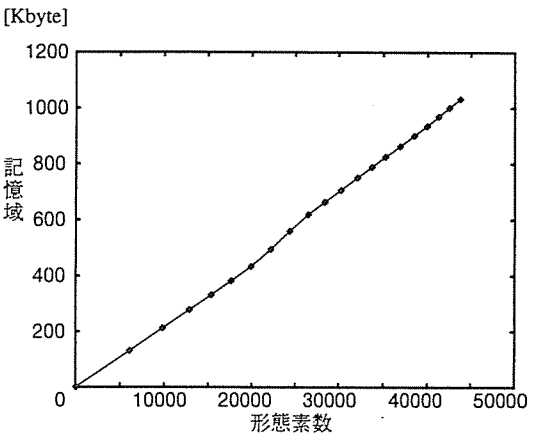


図6: AC法における形態素数と記憶域の関係

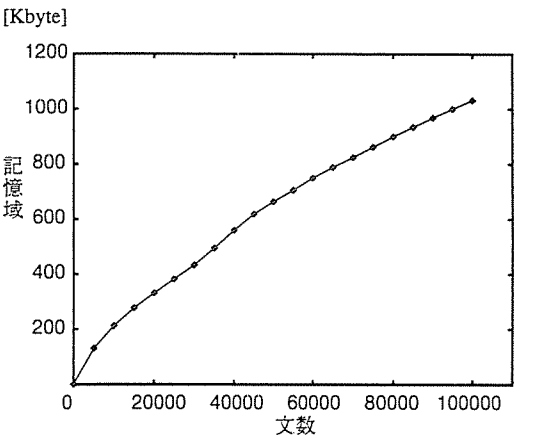


図7: AC法における文数と記憶域の関係

いう観点からは，最長の同一文字種からなる最長の文字列だけを対象にするなどの方法が有効であるが，形態素解析の未知語処理のためにどのようなモデルが最適であるかは，解決すべき課題である．

4.2 実験結果とその評価

学習コーパスから計算されるデータの中で，AC法とDFA法で異なるのは，辞書構造の記述部分だけである．その他のデータ，すなわちAC法とDFA法で共通のデータとその記憶域は以下の通りであった．

1. 品詞番号と品詞名の対応表 (162[byte])
2. 遷移確率行列 (1,096[byte])

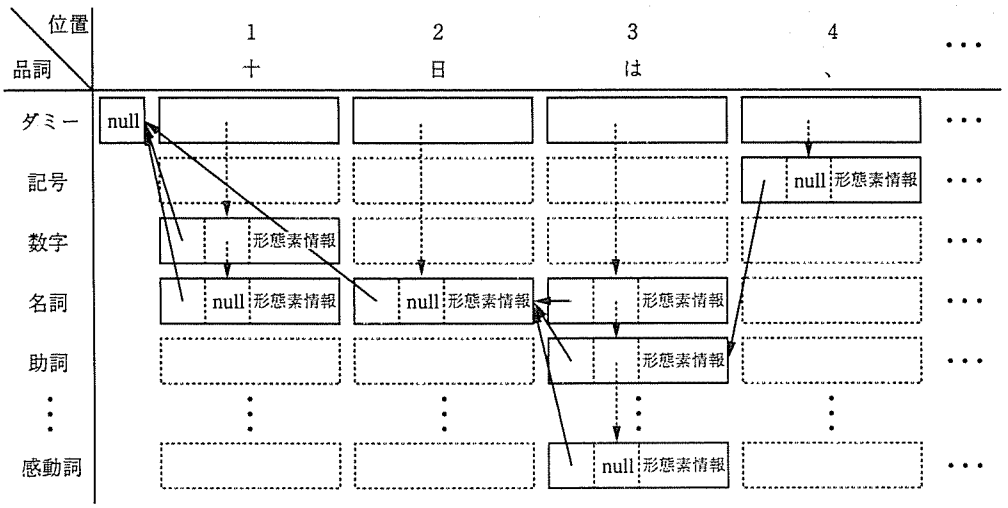


図 5: 最尤解の探索に用いる表

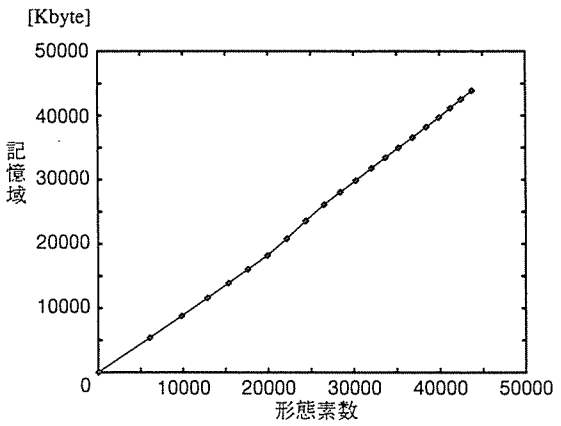


図 8: DFA 法における形態素数と記憶域の関係

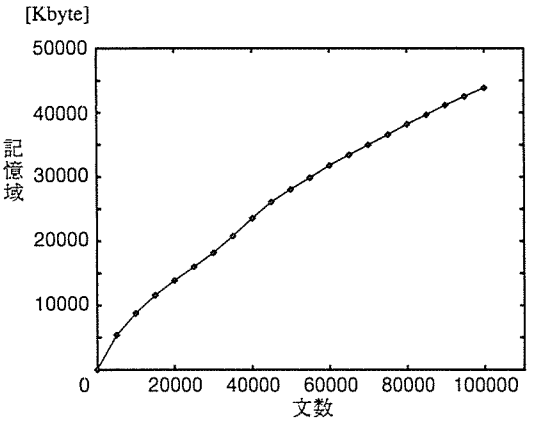


図 9: DFA 法における文数と記憶域の関係

3. 辞書の出力情報 (1.7[Mbyte])

速度の測定は，これらを主記憶に載せて行なった．

AC 法を用いて，学習コーパスから辞書や遷移確率などを計算するの要した時間は 19 分 47 秒であった．辞書の構造の記述に要した記憶域は 1.1[Mbyte] であった．さらに，学習コーパスの文数を変化させ，構造の記述に要する記憶域の変化を調べた．図 6は横軸に（異なり）形態素数をとった場合のグラフであり，図 7は横軸に文数をとった場合のグラフである．図 6を見ると，辞書の記憶域は形態素数にほぼ比例することが分かる．図 7から，文数が増加するにつれて辞書の記憶域は増加するが，その増加率が減少していくことが分かる．これは，文数に対して（異なり）形態素数の増加率が減少していくことの帰結であると推測される．

DFA 法を用いた場合，辞書や遷移確率などの計算時間は 2 時間 25 分 1 秒であった．辞書の構造の記述に要した記憶域は 43.9[Mbyte] であった．AC 法の場合と同様に，学習コーパスの文数を変化させ構造の記述に要する記憶域の変化を調べた．図 8は横軸に（異なり）形態素数をとった場合のグラフであり，図 9は横軸に文数をとった場合のグラフである．これらの図から，文数や

表 1: 辞書検索と形態素解析の速度 (文字 / 秒)

	AC 法		DFA 法		ChaSen-1.0b7
	二次記憶	主記憶	二次記憶	主記憶	
辞書検索	1200.8	61035.1	2590.6	74405.5	29331.0
形態素解析	106.6	178.2	164.1	187.8	747.4*

* ChaSen-1.0b7 の形態素解析の速度は、未知語モデルや最適解の探索や辞書の登録語などが異なるので、AC 法や DFA 法の結果と直接比較できない。

形態素数と記憶域の関係について AC 法の場合と同じことが言える.

次に, 学習コーパスから計算された辞書や遷移確率などを用いて, テストコーパスの辞書検索や形態素解析の実験を行なった. 表 1はそれぞれの方法における所要時間を測定した結果である. ただし, データの主記憶への読み込みにかかる時間を除いて計算してある. この時間は, オートマトンの記述を二次記憶に残す場合はいずれの方法でも 1 秒弱であり, 主記憶に読み込む場合は AC 法で 2 秒, DFA 法で 39 秒であった. 主記憶に読み込む場合は, 遷移関数の値を相対アドレスから絶対アドレスに変換している. DFA 法における状態遷移の回数は文字数の 2 倍の 8,056,630 回である. AC 法における状態遷移の回数は 7,583,413 回であった. 表 1から, 遷移関数を主記憶への読み込んだ場合, DFA 法は AC 法に比較して辞書検索のみの場合で 1.21 倍である. 状態遷移の回数と辞書検索の速度の関係が逆転しているのは, DFA 法における 1 回の状態遷移の計算コストが, AC 法における計算コストにくらべてかなり小さいことによると考えられる. DFA 法においてアルファベットを分割しない場合, 状態遷移の回数は半分になるので, より高速な辞書検索が実現できる. AC 法や DFA 法とは異なる方向のトライの改良である, パトリシア木を用いた辞書 [10] をもつ形態素解析器として茶筌がある. 同じ形態素集合に対してこの方法を用いた場合, 辞書の構造と出力情報の記述に要した記憶域は 1.13[MB] であり, 同じテストコーパスの辞書検索における状態遷移の回数は 64,532,604 回であった. これは, DFA 法による遷移回数の約 8 倍である. さらに, 辞書検索の速度は, 表 1に示したように, DFA 法による速度の 0.39 倍である. 辞書検索は, 見出し語を列挙する処理を含んでいるので, 辞書検索の速度の比は遷移回数の比に一致しないことに注意しなければならない. パトリシア木を用いた辞書構造が本質的に不利な点は, 見出し語と表記の比較が必要であるという点である. 最尤経路の探索も含めた場合, DFA 法の速度は, AC 法に比べて 1.05 倍となった. この値は, 最尤経路の探索において浮動少数演算を避けたり, 簡略化した未知語モデルを用いた場合大きくなる. 実際, 整数のコスト演算と簡単な未知語モデルを持つ茶筌による形態素解析の速度は 747.4[文字 / 秒] であった[5]. 以上のことから, 整数演算の最尤経路の探索エンジンと簡単な未知語モデルと AC 法や DFA 法などの高速な辞書により, より高速な形態素解析器が実現されると結論できる.

以上の結果から分かるように, AC 法と DFA 法では計算時間と記憶域のトレードオフがあるので, 形態素解析システムの実装には, 語彙数と計算機性能を考慮してどちらの方法を採用するかを決定すべきである. AC 法や DFA 法などの高速の辞書検索方法を用いた結果, 形態素解析全体の計算時間に占める辞書検索の時間は大きくなくなるので, 形態素解析の時間的性能は最尤経路の探索のアルゴリズムや未知語モデルに大きく依存することが実験的に示された. オートマトンを主記憶に載せる場合の DFA 法は現状のハードウェアでは最も速い辞書の実装法であると考えられるので, 形態素解析をさらに高速化するためには, 解の探索と辞書検索と未知語モデルを一括して考える必要があると考えられる.

5 おわりに

本論文では, 形態素解析の辞書検索を決定性オートマトンを用いて実現する方法を提案し, その時間的・空間的性能を AC 法と比較しながら論じた. 決定性オート

[5] この実験では, 条件を配布時のままにしているので, 品詞 (状態) の数は約 3 倍であり, 登録語数も約 3 倍である.

マトンを用いる方法は最も高速であることが理論的に示されるので，辞書検索を独立に考える限り，本論文で提案した方法が最も高速である．

今後，形態素解析をさらに高速化するためには，解の探索と辞書検索と未知語モデルを一括して考える必要がある．これは，確率オートマトンの最尤経路の計算がある種の半環上に成分をもつ行列の積の計算と等価であることを用いて実現されると考えられる．

謝辞

本研究を行なう上で，日本IBM東京基礎研究所の丸山宏氏に有益な資料や示唆に富むコメントを頂きました．ここに，感謝の意を表します．

参考文献

[1] 長尾真, 佐藤理史, 黒橋禎夫, 角田達彦: 自然言語処理, 岩波書店 (1996).

[2] Maruyama, H.: Backtracking-Free Dictionary Access Method for Japanese Morphological Analysis, *Proceedings of the 15th International Conference on Computational Linguistics*, pp. 208–213 (1994).

[3] Aho, A. V.: Algorithms for Finding Patterns in Strings, *Handbook of Theoretical Computer Science*, Vol. A: Algorithms and Complexity, Elseveir Science Publishers, pp. 273–278 (1990).

[4] Rivest, R. L.: On The Worst-Case Behavior of String-Searching Algorithms, *SIAM J. Comput.*, Vol. 6, No. 4, pp. 669–674 (1977).

[5] Cormen, T. H., Leiserson, C. E. and Rivest, R. L.: アルゴリズムイントロダクション 第2巻 アルゴリズムの設計と解析手法, 近代科学社 (1995).

[6] 石畑清: アルゴリズムとデータ構造, 岩波書店 (1989).

[7] 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 (1987).

[8] 日本電子化辞書研究所: EDR 電子化辞書仕様説明書 (1993).

[9] Ney, H.: The Use of One-Stage Dynamic Programming Algorithm for Connected Word Recognition, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 32, No. 2, pp. 263–271 (1984).

[10] 山下達雄, "パトリシア木を用いた形態素解析のための辞書検索 第1版," ChaSen Technical Report, CTR-1, Oct 1996.