文字ベースのHMMによる
複合語単語分割の誤り修正

執筆者：池谷 昌紀

指導教官：新納 浩幸

平成１１年３月３日

目 次

Chapter 1

序論

1.1 はじめに

本論文では複合語の単語分割を行うために、通常の形態素解析と文字ベースの HMM とを相補的に利用する方法を提案する。

複合語の単語分割は従来の言語処理システムの一要素技術として重要であるだけでなく、全文検索で生じるインデックス作成や検索式の解析などにも必要とされその重要性は高い。複合語の単語分割は一般に形態素解析によって行えるが、分割精度の他に未知語の扱いも問題となるため、形態素解析を用いない手法も提案されている。その中の一つとして文字ベースの HMM もある。文字ベースの HMM では、 文字ベースなので未知語の問題は生じない。ただし文字ベースの HMM において、状態 i から 状態 j に遷移するときに文字 a が出力されるコスト B を uni-gram や bi-gram などから得ると、長い文字列が一単語となる場合に正しく単語分割が行えないことが多い。そのためにプラスアルファの何らかの工夫が必要となる。山本は単純な文字ではなく、拡張文字として品詞などの情報も文字に付加している。また Tsuji は対訳辞書から形態素数を得る工夫を行っている。また小田は PPM モデルを利用して出力シンボル可変長 $n-gram$ にしている。

ただし上記の問題は形態素解析では掃除ない。形態素解析による単語分割では、長い文字列からなる一単語を正しく認識することはむしろ容易である。一方、形態素解析による単語分割の誤りは文字列の局所的な部分であり、あるパターンが存在する。このような文字列の局所的な部分の単語分割に対しては、文字ベースの HMM が有効である。

つまり単語分割に対して文字ベースの HMM と一般の形態素解析は相補的に利用できる。

そこで本論文では、形態素解析で生じた誤りを文字ベースのHMMから修正することで単語分割を行う。形態素解析の誤りをどのように判断するかが問題であるが、ここでは形態素解析の誤りパターンに注目し、該当パターン部分を文字ベースのHMMによる単語分割結果と比較することで、誤りかどうかを判断する。

また本論文で利用した形態素解析システムはJUMAN 3.5であることを注記しておく。

Chapter 2

HMM

2.1 マルコフ過程

まず始めに、HMM法（Hidden Markov Model, 隠れマルコフモデル）の基礎となるマルコフ過程について述べる.

時刻 t における確率変数を $Y_t(t=1,2,\cdots)$ 、観測値を y_t としよう. $Y_1=y_1$, $Y_2=y_2$,$\cdots$, $Y_t=y_t$ と知ったときの、ナル確率が次式で与えられるとき、n 重マルコフ過程という.

$$P(Y_{t+1}=y_{t+1}|\boldsymbol{Y}_1^t=y_1^t)=P(Y_{t+1}=y_{t+1}|\boldsymbol{Y}_{t-n-1}^t=y_{t-n+1}^t$$

ここで、 $\boldsymbol{Y}_1^t(y_1^t)$ は Y_1 , Y_2 , $\cdots$, Y_t (y_1 , y_2 ,$\cdots$, y_t) の時系列をあらわし、$\boldsymbol{Y}_1^t=y_1^t$ は、$Y_1=y_1$, $Y_2=y_2$, $\cdots$ $Y_t=y_t$ をあらわすものとする. 特に、1 重マルコフ過程のときには単純マルコフ過程という.

確率過程 $\{Y_t\}$ のとりうる値を有限個または可付番号無限個として、これを非負の数だけに限っても一般性を失わない. 単純マルコフ過程においては、$Y_1=n_1$, $Y_2=n_2$, $\cdots$, $Y_t=n_t$ のとき、$Y_{t+1}=n_{t+1}$ なる条件を考えれば、

$$P(Y_{t+1}=n_{t+1}|\boldsymbol{Y}_1^t=n_1^t)=P(Yt+1=n_{t+1}|\boldsymbol{Y}_t=n_t)$$

が成立する. これが常に成立するものとして、

$$P_{ij}(t,t+1)=P(Y_{t+1}=j|Y_t=i)$$

と書くと、

$$P_{ij}(t,t+1)\geq\sum_j P_{ij}(t,t+1)=1$$

である. $Pij(t,t+1)$ は、時刻 t に状態 i であったものが時刻 $t+1$ で状態 j に遷移する確率を表す. この遷移確率が t に無関係なとき, Y_1 , Y_2 , $\cdots$ を定常な遷移確率をもつマルコフ過程という. 以下、定常性を仮定して $Pij(t,t+1)$ を単に P_ij と書くことにする. また、初期状態確率 π を次式で定義する.

$$\pi_i = P(Y_1 = i), \sum_j \pi = 1$$

n 重マルコフ過程を考えるとき、一個の確率変数 Y_t のかわりに n 次元ベクトル (Y_t , Y_{t+1} ,$\cdots$, Y_{t+n-1}) = $\boldsymbol{Y}_t$ に着目すれば、新しいベクトル確率過程 $\{\boldsymbol{Y}_t\}$ が得られる. この過程では $\boldsymbol{Y}_{t-1}$ の最初のベクトル成分は y_{t-1} であり、それ以外の成分は $\boldsymbol{Y}_t$ に含まれている成分と同じであって、現在の事象はその一つ前の事象だけの影響下にあることとなり、確率変数の n 次元のベクトルを考えるだけで n 重マルコフ過程は単純マルコフ過程に帰着されることになる.

$\{\boldsymbol{Y}_t\}$ のとりうる値を L_1 , L_2 ,$\cdots$, L_S という S個のシンボルとすれば、$\boldsymbol{Y}_t = (Y_t, Y_{t+1}, \cdots, Y_{t+n-1})$ のとりうる値はたかだか S^n (S^m の状態) のシンボルとなる.

定常な遷移確率 P_{ij} を持った単純マルコフ過程を考えよう. 遷移確率 P_{ij} を要素とする行列

$$\boldsymbol{P} = \begin{pmatrix} P_{00} & P_{01} & P_{02} & \cdots \\ P_{10} & P_{11} & P_{12} & \cdots \\ P_{20} & P_{21} & P_{22} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

を遷移確率マトリクスという. 一般に次の関係が成立する.

$$P_{ij}(m+n) = \sum_v^{\infty} P_{iv}(m) p_{vj}(n)$$

ここで、左辺はあるとき i であった場合に $(n+m)$ 回の遷移で j になる確率である. これを行列形式で書けば、

$$\boldsymbol{P}(m+n) = \boldsymbol{P}(m)\boldsymbol{P}(n) = (\boldsymbol{P})^{m+n}$$

となり、これは Kolmogorov-Chapman の方程式といわれている. $t=1$ における初期状態の確率を $\sum_i$ とすれば、$Y_t = j$ となる確率は、

$$P_j(t) = \sum_i \pi_i \cdot P_{ij}(t-1)$$

となる. $P_{ij}(t) > 0$ であるような t が存在することは、状態 i から状態 j へいつかは到達可能であることを意味し、任意の i, j について $P_{ij}(t) > 0$ ならこのマルコフ過程は正則であるという. どのような状態から出発しても何回かの後には必ずほかの任意の状態に移ることのできるマルコフ過程はエルゴード的であるという. 正則なマルコフ過程はエルゴード的である. 正則なマルコフ過程では、初期上位の確率に関係なく、サンプル系列 $\{n_1, n_2, \cdots, n_t\}$ に対して状態 j を通過する回数を $u_j(t)$ とすれば、状態 j を通過する確率 $\frac{u_j(t)}{t}$ は、t を無限大にすれば一定値となる. これはエルゴードの定理と呼ばれている.

n 重マルコフ過程は状態遷移図で表現するころができる. ベクトル ($y_t, y_{t+1}, \cdots, y_{t+n-1}$) がとりうる値の一つを ($L_{i0}, L_{i1}, \cdots, L_{in-1}$) とすればこれを 1 状態に対応させる. また、各状態について、その状態から 1 回の遷移で到達することができる状態（たかだか ($L_{i1}, L_{i2}, \cdots, L_{in-1}, L_k$) , $k = 1, 2, \cdots, S$ の S個の状態）を方向を持った線で結ぶ. また、この遷移に対して、状態遷移確率を付随させ、さらに、一つのシンボル L_k を発生させるものとする. 図 2.1、図 2.2 は $\{a, b\}$ の二つの出力シンボルをもつ単純マルコフ過程、2 重マルコフ過程の状態遷移図である. 状態遷移確率は、このマルコフ過程によって生成されうる長い訓練用サンプル系列を用いることによって容易に求められる. 例えば、図 2.1 で N_A を状態 A を通った回数、$N_A(b)$ をシンボル b が状態 A から生成された回数とすれば、

$$P(b|a) = \frac{N(a,b)}{N(a)} = \frac{N_a(b)}{N_A}$$

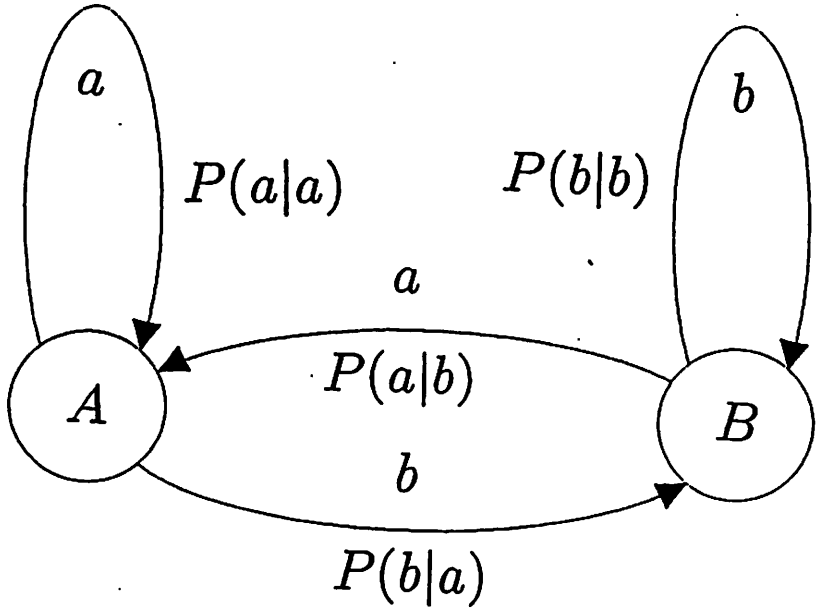

図 2.1　単純マルコフ過程の例

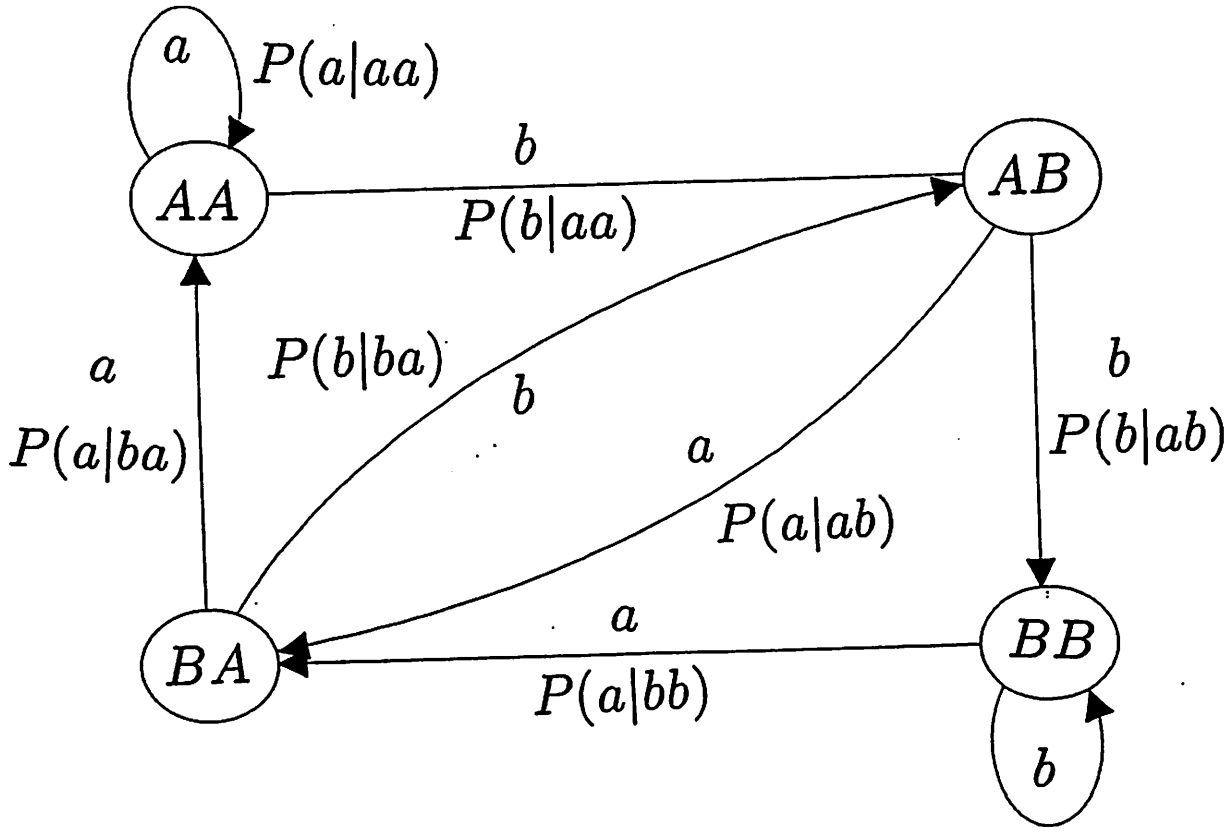

図 2.2　二重マルコフ過程の例

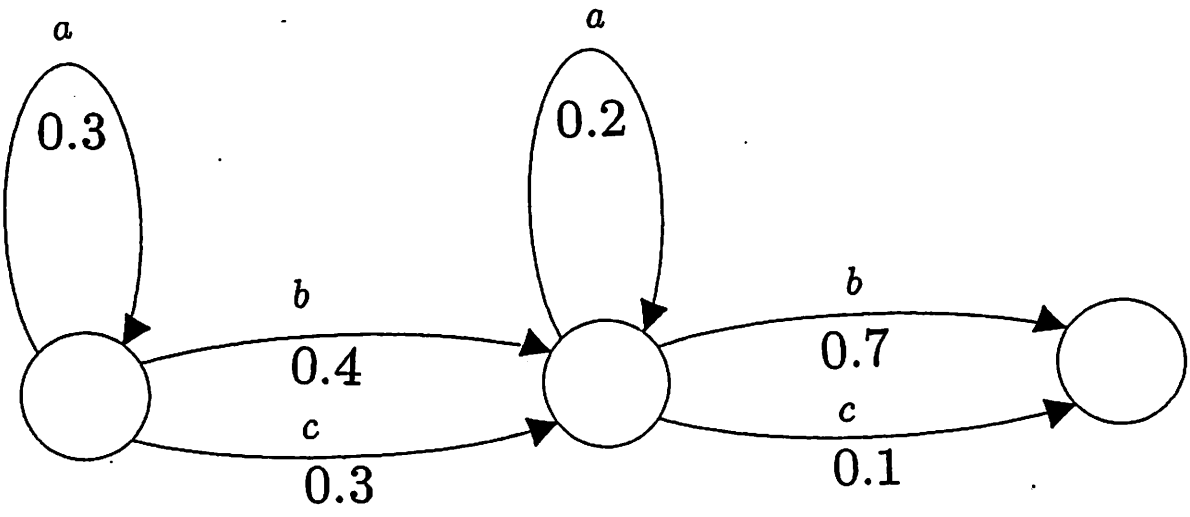

図 2.3　ユニフィラーな有限状態情報源

同様に、図 2.2 で N_{AB} を状態 AB を通った回数、$N_{AB}(b)$ をシンボル b が状態 AB から生成された回数とすると、

$$P(b|ab) = \frac{N(a,b,b)}{N(a,b)} = \frac{N_{AB}(b)}{N_{AB}}$$

となる. マルコフ過程のように、各状態から出ているすべての遷移がそれぞれ異なったシンボルを生成する有限状態情報源はユニフィラー（unifilar）と呼ばれる. ユニフィラーな有限情報源から生成されるシンボル系列を生成する確率は容易に求められる. たとえば、図 2.3 の例で、状態 1 が初期状態、状態 3 が最終状態であるとき、シンボル系列 aabaaac の生成確率は、

$$P(aabaaac) = 0.3 \times 0.3 \times 0.4 \times 0.2 \times 0.2 \times 0.2 \times 0.2 \times 0.1 = 2.88 \times 10^{-5}$$

となる.

2.2 基本HMMの定式化

HMM（Hidden Markov Model, 隠れマルコフモデル）は、出力シンボルによって一意に状態遷移が決まらないという意味での非決定有限状態オートマトンとして定義される（一般に、マルコフモデルは最終状態の概念はないが、今回の実験での単語分割に用いたマルコフモデルは初期状態、最終状態を設定する）. 定義から当然HMMはユニフィラーではない. ずなわち、出力シンボルが与えられても状態遷移系列は唯一に決まらない. 観測できるのはシンボル系列だけであることから隠れ (hidden) マルコフモデルと呼ばれている. 簡単な例を図 2.4 に示す. 図中のアーク上のスカラ値は状態遷移確率を、アーク上のベクトル値はシンボル a, b の状態遷移による条件付きの出力確率を示している. すなわち、ベクトルの第一要素が a の、第二要素が b の出力確率を示している. この場合シンボル系列が abb であった場合可能性のある状態遷移系列は $S_1S_2S_3$ と $S_1S_2S_2S_3$ の二通りがある. 今回は出力シンボル系列は最終状態に達するものとして扱う. 通常HMMは全状態が最終状態となりうるが、複合語の単語分割では一部分の状態のみが最終状態となる. 一般性を失うことなく、状態遷移にナル遷移（シンボルを何も出力しない）は存在せず二つの状態間での遷移はたかだか一つと仮定できる. 前者の場合は、ナル遷移のときは空記号 λ が出力され

たと考えればよい. また、ナル遷移を除去することもできる. これによって後者が満たされ

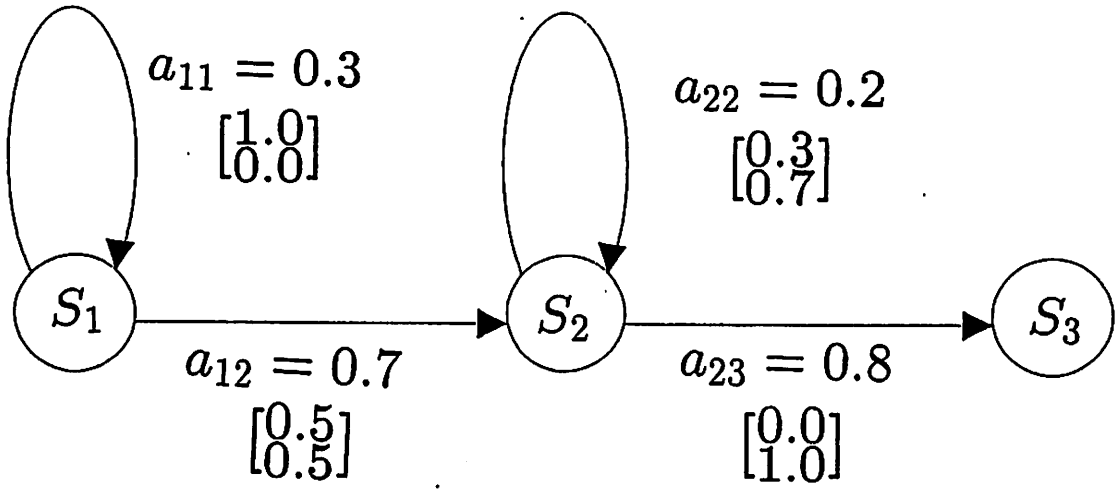

図 2.4　HMMの例

なくなるが、二つの状態遷移 $S_2 \to S_3$ に対して、新たな状態$\dot{S}_3$ を設けて、一方の遷移を $S_2 \to \dot{S}_3$ に変換し、状態 $\dot{S}_3$ の遷移先を状態 S_3 の遷移先に等しくすればよい. また、 y, x をそれぞれ出力と状態の確率変数値の系列としよう. 今考えているHMMを M とすれば、マルコフモデルの仮定から

$$P(x_i|\boldsymbol{x}_i^{i-1}, M) = P(x_i|x_{i-1}, M)$$

また、非決定性（hidden）の仮定から、

$$P(y_i|\boldsymbol{x}_1^i, \boldsymbol{y}_1^{i-1}, M) = P(y_i|x_{i-1}, x_i, M) = P(y_i|x_{i-1}, M)$$

となる. 式は状態遷移によって yi の出力確率が定まるとするもので、式は遷移元の状態によって定まるとするものである. 混同のない限りHMMを示す M の条件項を省くことにする. 単語 w_i に対するHMMを上述の M とすれば、式の $\boldsymbol{Y}_{t_i-1}^{t_i}$ をあらためて $\boldsymbol{y}_1^T$ とおけ

ば、$P(\boldsymbol{y}_1^T|M)$ は、

$$P(\boldsymbol{y}_1^T) = \sum_x P(\boldsymbol{y}_1^T|x)P(\boldsymbol{x})$$

となる. また、

$$P(\boldsymbol{x}) = \prod_i P(x_i|\boldsymbol{x}_1^{i-1}) = \prod_i P(x_i|x_{i-1})$$

から、

$$P(\boldsymbol{y}) = \sum_{\boldsymbol{x}} \prod_i P(x_i|x_{i-1}) \cdot P(y_i|x_{i-1}, x_i)$$

となる. 図 2.4 の例で $P(abb)$ の値を式を用いて求めてみよう. 先にも述べたとおり $\prod_i P(y_i|x_{i-1}, x_i)$ が 0 にならないのは二つの状態系列だけであったので、この各々の系列による値を $P_1(abb)$, $P_2(abb)$ とすれば、

$$P_1(abb) = 0.3 \times 1.0 \times 0.7 \times 0.5 \times 0.8 \times 1.0 = 0.084$$

$$P_2(abb) = 0.7 \times 0.5 \times 0.2 \times 0.7 \times 0.8 \times 1.0 = 0.0392$$

ゆえに、$P(abb)$ は、

$$P(abb) = P_1(abb) + P_2(abb) = 0.084 + 0.0392 = 0.1232$$

となる. 以上から、HMM $\boldsymbol{M}$は次の六つの組
で定義される.

- $\boldsymbol{S}$: 状態の有限集合 ; $\boldsymbol{S} = \{s_i\}$
- $\boldsymbol{Y}$: 出力シンボルの集合
- $\boldsymbol{A}$: 状態遷移確率の集合 ; $\boldsymbol{A} = \{a_{ij}\}$; a_{ij} は状態 s_i から状態 s_j への遷移確率、ここで
  $\sum_j a_{ij} = 1$。
- $\boldsymbol{B}$: 出力確率の集合 ; $\boldsymbol{B} = \{b_{ij}(k)\}$; $b_ij(k)$ は状態 s_i から状態 s_j への遷移の際にシンボル k を出力する確率. ここで、

$$\sum_k b_{ij}(k) = 1(\text{離散 HMM})$$

$$\int_{-\infty}^{\infty} b_{ij}(k)dk = 1(\text{連続 HMM})$$

- $\boldsymbol{\pi}$：初期状態確率の集合；$\boldsymbol{\pi} = \{\pi_i\}$ ；π_i は初期状態が s_i である確率.

$$\sum_j \pi_j = 1$$

- $\boldsymbol{F}$：最終状態の集合

ここで注意を要するのは、出力 $\boldsymbol{Y}$ が有限集合の場合と無限集合の場合とがあることである。各々の場合について $\boldsymbol{B}$ の表現方法が異なってくる。前者の場合は、出力確率分布は離散的でノンパラメトリックである。シンボルは有限集合だからあらかじめ $\boldsymbol{B}$ を求めてテーブル化しておける。一方、後者の場合のそれは、ガウス分布のような (連続分布) で与えられる。連続分布の時は、シンボルは無限集合となり、あらかじめテーブル化しておけないので、出力シンボルが観測されるごとに $b_{ij}(k)$ を求めなければならない。

また y_i は時間的に等間隔に観測される場合と可変時間間隔ごとに観測される場合とがある。

$\boldsymbol{Y}$の出力形式以外に HMM に関して重要な基本問題として次の四つを挙げることができる。

1. モデルの設計：状態数や遷移先の種類、状態継続時間制御などどのような HMM を用いるかを決定すること。

2. モデルの評価：モデル $\boldsymbol{M}$ がシンボル系列 $\boldsymbol{y} = y_1, y_2, \cdots, y_T$ を出力する確率 (尤度) $P(\boldsymbol{y}|\boldsymbol{M})$ を求めること (認識時には、各モデルに対して、$P(\boldsymbol{y}|\boldsymbol{M})$ が最大になる $\boldsymbol{M}$ を決定する。

3. 最適状態系列の推定：モデル $\boldsymbol{M}$ がシンボル系列 $\boldsymbol{y}$ を出力するときの最も可能性の高い状態遷移系列を推定し、その系列に対する確率を求めること。

4. モデルの推定：訓練用シンボル系列 $\boldsymbol{y}$を与えて、$P(\boldsymbol{y}|\boldsymbol{M})$ が最大になるようにモデル $\boldsymbol{M}$のパラメータを推定すること。

2.3 前向きパスアルゴリズム

出力シンボル系列を与えた時、通ってきた状態遷移系列を推定することで単語分割が行なえる。推定する方法として全解探索がある。しかしこの方法では出力シンボルの数を n

とすると組み合わせの数が 2^{n-1} となり、長い文字では推定のコストが高い。そこで、本実験では 動的計画法の一種である Viterbi アルゴリズムを使うことにする。

ここでは、Vitervi アルゴリズム の主要なモジュールである、前向きパスアルゴリズムを示す。

図 2.4 について考えてみる. 入力が abb の場合、可能な二つの状態遷移系列にたいして式で示したように各々 5 回の乗算が必要であった. また、$P(abb)$ の計算回数は１０回の乗算と 1 回の加算であった. この計算は次のように書き換えられる。

$$P(abb) = [P(ab, S_1 \to S_1 \to S_2) + P(ab, S_1 \to S_2 \to S_2)] \times P(b, S_2 \to S_3)$$

このようにすると、全計算回数は 8 回の乗算と 1 回の加算ですむ. 一般的に次に示す前向きパスアルゴリズムによって効率よく $P(y_1, y_2, \cdots, y_T)$ を求めることができる. ここで $\alpha(i,t)$ を $y_1, y, \cdots, y_t$ を生成して状態 i に達する確率としよう. また、$\boldsymbol{F}$ を最終状態の集合とする. すると、

$$P(y_1, y_2, \cdots, y_T) = \sum_{i \in \boldsymbol{F}} \alpha(i, T)$$

となる. $\alpha(i,T)$ の算出アルゴリズムを示す.

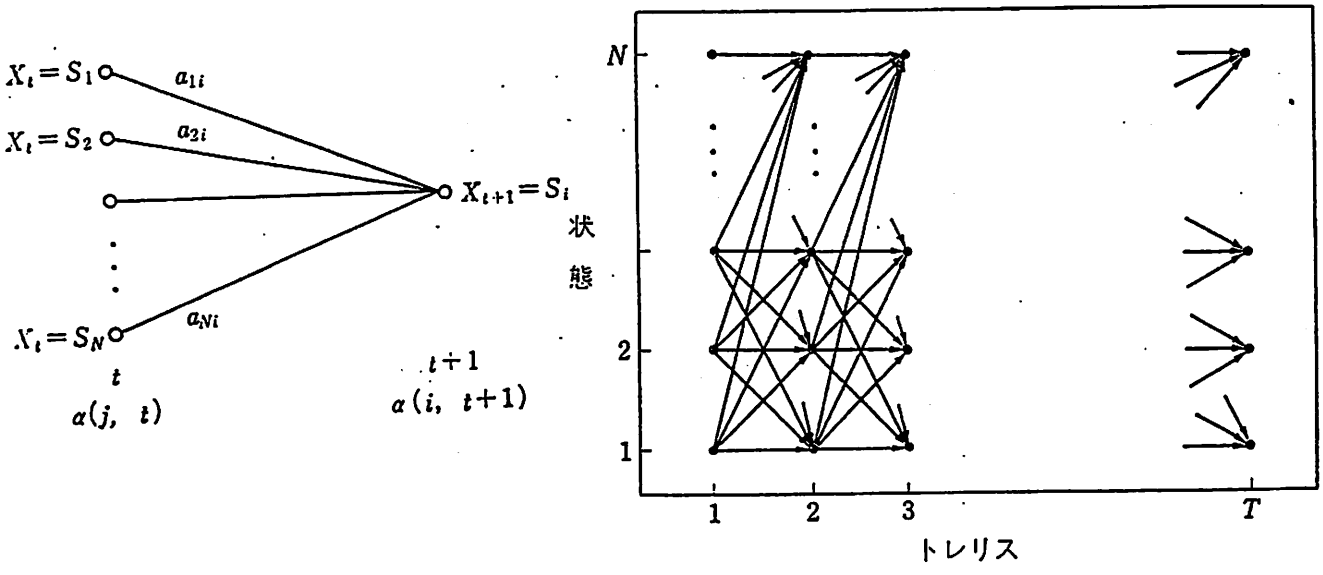

図 2.5　α（i，t）の計算手順

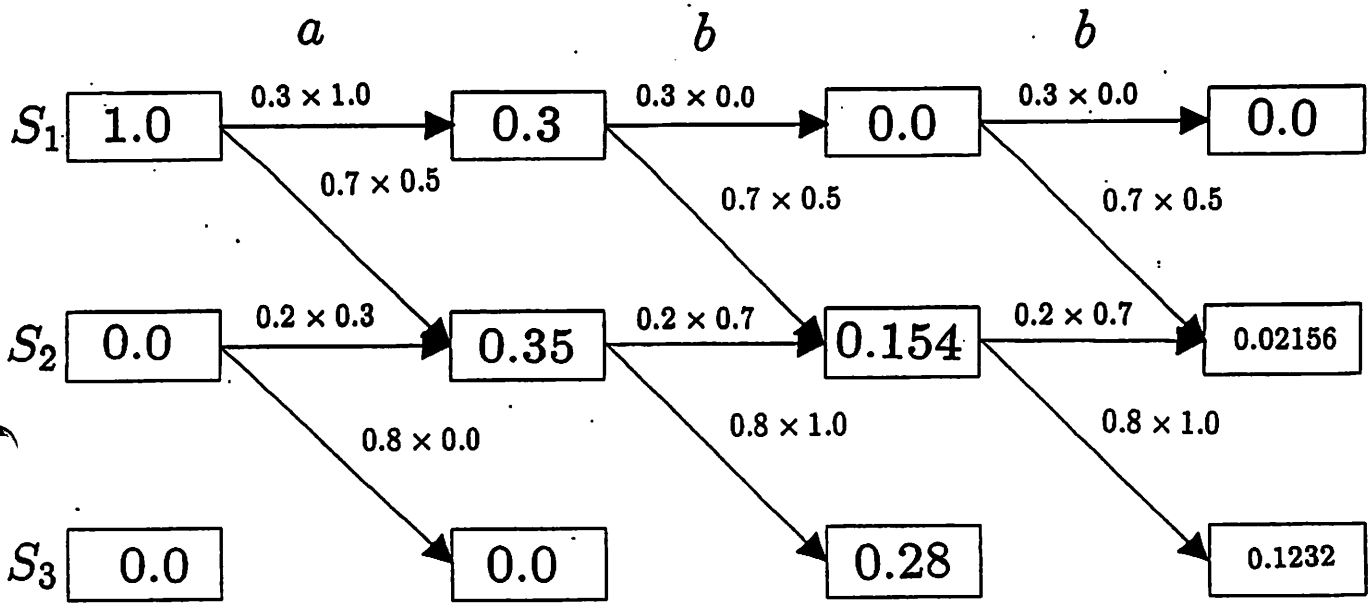

図 2.6　P(abb) のトレリス上の計算

前向きパスアルゴリズム

前向きパスアルゴリズムとは以下のようにして $P(y_1, y_2, \cdots, y_t)$ を求めるアルゴリズムである。ただし、$\alpha(i,t)$ は $y_1, y_2, \cdots, y_t$ を生成して状態 i に達する確率とする。

1. 初期化　すべての状態 i に対して $\alpha(i,0) = \pi$
2. $t = 1, 2, \cdots, T$ に対して　3 , 4　を実行
3. すべての状態 i に対して　4　を実行
4. $\alpha(i,t) = \sum_j \alpha(j, t-1) a_{ij} b_{ji}(y_t)$
5. $P(y_1, y_2, \cdots, y_T) = \sum_{i \in F} \alpha(i,T)$

これは、出力シンボル系列に対応する時間経過を横軸に、各状態を縦に並べて許される状態遷移を示すトレリス（trellis）の上で考えると理解しやすい．図 2.6 は図 2.4 のHMMで出力シンボル系列が abb の場合のトレリス上での $\alpha(i,t)$ の値を示している．これより $P(abb) = \alpha(S_3, 3) = 0.1232$ となることが理解できるだろう．

2.4 Viterbi アルゴリズム

二つの単語 w_1 と w_2 に対応する HMM M_1とM_2 を連結させて一つのHMM M_3 を構成したとしよう．この時、M_1 の最終状態からM_2 の初期状態へナル遷移を設けて連結するものとしよう（あるいは、M_1 の最終状態をM_2 の初期状態に縮退させても良い）．M_3 と入力文字列に対応する出力シンボル系列 $y1, y2, \cdots, y_T$ に対して前述のアルゴリズムで $P(y_1, y_2, \cdots, y_T|M)_3$ を求めることができる．これは一般に、

$$P(y_1, y_2, \cdots, y_T)|M_3 = \sum_t P(y_1, y_2, \cdots, y_t)|M_1 \times P(y_{t+1}, y_{t+2}, \cdots, y_T)|M_2$$

と表現できる．これは、M_1 の最終状態からほかの状態への遷移先はM_2 の初期状態にかぎられる事から明らかである．上式の関係から、$P(y_1, y_2, \cdots, y_T|M_3)$ の算出の際には、$w_1(M_1)$ にもっともよく対応するシンボル系列区間 $y_1, y_2, \cdots, y_t$ と $w_2(M_2)$ にもっともよく対応するシンボル系列区間 $y_1, y_2, \cdots, y_T$ を決定する事はできない．ところが複合語の単語分割のためにこの最適な単語境界時点 t を知りたい．これは次のような定義

$$P'(y_1, y_2, \cdots, y_T|M_3)$$

から求められる．

$$P'(y_1, y_2, \cdots, y_T|M_3) = \max_t P(y_1, y_2, \cdots, y_t|M_1) \times P(y_{t+1}, y_{t+2}, \cdots, y_T|M_2)$$

上式の定義は近似的ではあるが妥当な定義とも考えられる．言い換えれば y_t をHMM M_3 の唯一の状態遷移に対応させたことになる．この考えをすべての y_t に拡張すれば、式

$$P(y_1, y_2, \cdots, y_T) = \sum_x \prod_i P(x_i|x_{i-1}) \cdot P(y_i|x_{i-1}, x_i)$$

の代わりに、

$$P''(y_1, y_2, \cdots, y_T) = \max_x \{\prod_i P(x_i|x_{i-1}) \cdot P(Y_i|x_{i-1}, x_i)\}$$

を求めることになる.

これは式の {} 内の項を最大にする状態遷移系列上での確率で $P(y_1, y_2, \cdots, y_T)$ を近似するものである. これによって、任意の y_t は唯一の状態遷移に対応付けられる. この状態遷移系列を最適パスと呼ぶ.

最適パスとこのパス上での確率を求めるためには、動的計画法を用いる. このアルゴリズムは最初 Viterbi によって提案された事から、Viterbi アルゴリズムと呼ばれている. $\boldsymbol{Q}(i,t)$ を $P''(y_1, y_2, \cdots, y_t) = P(y_1, y_2, \cdots, y_t, \{\boldsymbol{x}\})$ を最大にするパス（状態遷移系列）、$\boldsymbol{f}(i,t)$ をその確率と定義すると Viterbi アルゴリズムは以下で与えられる.

Viterbi アルゴリズム

1. 初期化

   すべての状態 i に対して $\boldsymbol{f}(i,0) = \pi_i$

2. $t = 1, 2, \cdots, T$ に対して (3)、(4) を実行。

3. すべての状態 i に対して (4) を実行。

4. $\hat{i} = \underset{j}{\operatorname{argmax}}\, \boldsymbol{f}(k,t) a_{ki}$

   $\boldsymbol{f}(i,t) = \max\{\boldsymbol{f}(\hat{j}, t-1) a_{\hat{j}i} \cdot b_{\hat{j}i}(y_t)\}$
   $\boldsymbol{Q}(i,t) = \boldsymbol{Q}(\hat{j}, t-1) \otimes \hat{j}$

5. $i = \underset{i \in F}{\operatorname{argmax}}\, \boldsymbol{f}(i,T)$

   $P''(y_1, y_2, \cdots, y_T) = P(y_1, y_2, \cdots, y_T, \boldsymbol{Q}(\hat{i}, T)) = \boldsymbol{f}(i,T)$

ここで、$\otimes$ は状態遷移系列と状態を連結し、新たな状態遷移系列をつくるオペレータである. 最適状態遷移系列は（最適パス）は $Q(\hat{i}, T)$ で与えられる. HMM法による認識では、$P(y)$ をもちいる代わりに、この $f(\hat{i}, T)$ を用いて最大値を与えるモデルのカテゴリー

を認識結果とする方法もあり、同等の認識精度が得られている.

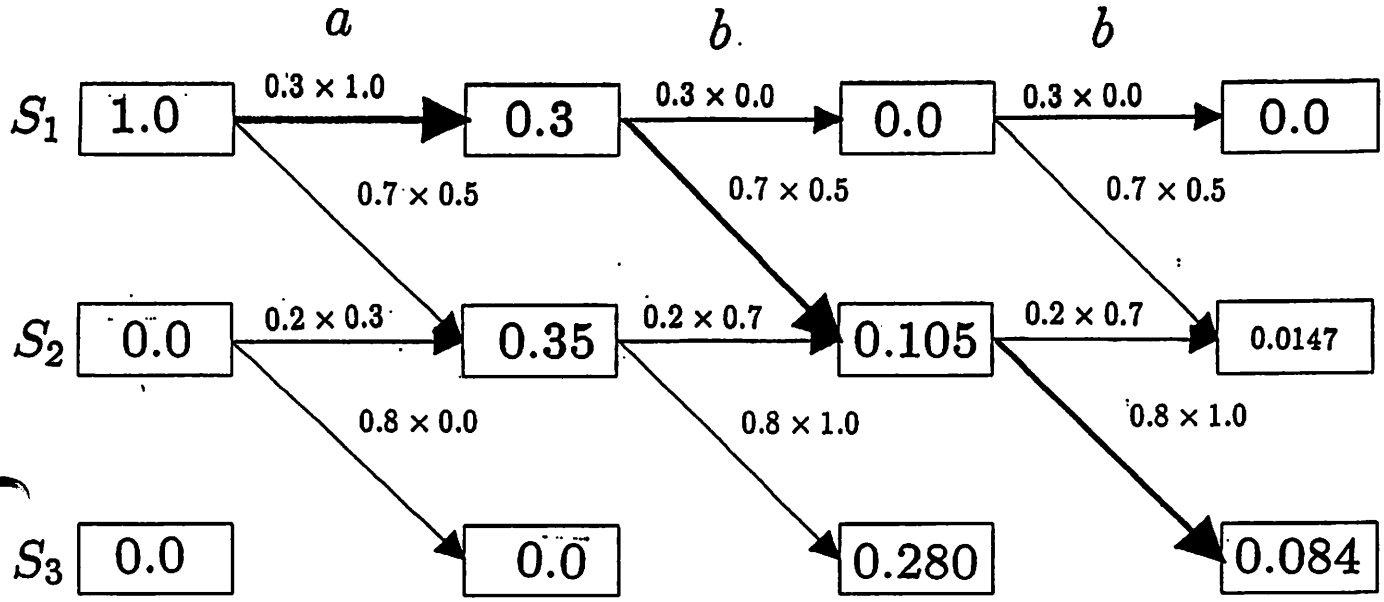

図 2.7　HMMにおけるシンボル系列 abb に対する最適パス算出例

2.5　3．4 Viterbi アルゴリズムの例

形態素解析にもちいるHMMのモデル上では、ある状態からある状態へ遷移するさいにあるシンボルを出力する. あるじょうたいへの遷移は存在する状態の数だけとることができ、それぞれ確率によって決まっている. 初期状態から最終状態まで遷移し、観測できるのは出力シンボルのみである. HMMは出力シンボルが与えられても状態遷移系列は唯一にしか決まらない. さきの3．4の例でHMMのモデルを考えてみる. このようなモデルがあったとき、最大の状態遷移確率と状態遷移系列を同時に求めるのが Viterbi アルゴリズムである. つまり、このアルゴリズムによって状態遷移を最適にする状態遷移系列と、この系列上での確率を求めることができる.

2.3 確率モデルの推定

確率モデルの特徴　統計的に情報源をモデル化するため、適用範囲が広く頑健なシステムを構築できる。

2.3.1 最尤推定

標本分布を忠実に表すように確率モデルを推定する方法としては、モーメント法と、最尤推定法がある。

1. モーメント法

   観測データ $y_1 \cdots y_n$ があり、この観測データを確率変数 Y の実現値と考えた時、$E[Y]$ を

   $$\frac{1}{n}\sum_{i=1}^{n} y_i^k$$

   によって推定する。

2. 最尤推定法

   観測データ $y_1 \cdots y_n$ があり、この観測データを確率変数 Y の実現値と考えた時、同時確率分布

   $$P(Y_1 = y_1, \cdots, Y_n = y_n) = f(y_1, \cdots, y_n; \theta)$$

   が最大になるようにパラメータ を推定する。

最尤推定法において確率密度関数が $f(y, \theta)$ の時、確率変数 $\boldsymbol{Y} = Y_1^n (= Y_1, \cdots, Y_n)$ の観測値 $\boldsymbol{y} = y_1^n (= y_1, \cdots, y_n)$ が同時に生起する確率

$$L(\theta; \boldsymbol{y}) = \prod_{i=1}^{n} f(y_i, \theta)$$

を θ の関数と見て 尤度関数と呼ぶ。これを最大にする $\theta = \theta(y)$ を最尤推定値、最尤推定を用いて y_i の点推定 $f(y_i, \theta)$ を行なうことを最尤推定法と呼ぶ。

2.3.1.1 多項分布の場合

$\theta = \{\theta_1, \theta_2, \cdots, \theta_m\}$ に対して

$$f(y_i) = \theta_i, \sum_{i=1}^{m} \theta_i = 1$$

が与えられるとき、

$$L(\theta; \boldsymbol{y}) = \prod_{i=1}^{n} \theta_{y_i}$$

という同時確率分布を最大化する θを決める問題となる。最大するために、尤度の対数をとる。

$$\log L(\theta; \boldsymbol{y}) = \sum_{i=1}^{n} \log \theta_{y_i}$$

これを対数尤度と呼ぶ。

$$T_j(\boldsymbol{y}) = \sum_{i=1}^{n} \delta(y_i, j)$$

とおくと、

$$\log L(\theta; \boldsymbol{y}) = \sum_{j=1}^{m} T_j(\boldsymbol{y}) \log(\theta_j)$$

となる。ラグランジュの未定係数法を使うと

$$\theta_j = \frac{T_j(\boldsymbol{y})}{n}$$

を得る。

2.3.1.2 マルコフモデルの状態遷移確率の場合

2つの状態 から成る2元情報源があり、観測の結果、各遷移の生起回数 が得られたとすると、これらの事象の同時確率は

$$P = W_0 P_{A\to A}^{c_1} P_{A\to B}^{c_2} P_{B\to A}^{c_3} P_{B\to B}^{c_4} F$$

W_0: 初期状態確率

F: 最終状態確率

となる。 ここで $\log P$ を

$$\sum c_i = N \quad \sum P_A = 1 \quad \sum P_B = 1$$

の下でラグランジュの未定係数法を用いて解くと、次式が得られる。

$$P_{A\to A} = \frac{c_1}{c_1 + c_2}$$

$$P_{A\to B} = \frac{c_2}{c_1 + c_2}$$

$$P_{B\to A} = \frac{c_3}{c_3 + c_4}$$

$$P_{B\to B} = \frac{c_4}{c_3 + c_4}$$

最尤推定の例

偏頭痛の患者 20 人に薬を服用させたところ、6 人が効いたと答え 14 人は変化なしと答えた。

ここで頭痛の軽減率 p を推定する。

偏頭痛の例で、観測結果を固定して、つまり $n = 20, x = 6$ を固定して、p を変数とみなし、f を p の関数と考えると、尤度関数

$$L(p;x) = \binom{20}{6} p^6(1-p)^{14}$$

が得られる。p の種々の値に対応する尤度 L の値は下に示す。

$p = 0.1 \to L = 0.88670 \times 10^{-2}$

$p = 0.15 \to L = 0.45373 \times 10^{-1}$

$p = 0.2 \to L = 0.10910$

$p = 0.25 \to L = 0.16861$

$p = 0.3 \to L = 0.19164$

$p = 0.35 \to L = 0.17123$

$p = 4 \to L = 0.12441$

$p = 6 \to L = 0.48544 \times 10^{-2}$

ここで 20 人の患者の中で 6 人が頭痛が軽くなったと答えたということは、直観的にこの薬の頭痛軽減率の推定値は、$\frac{6}{20} = 0.30$ であることがわかる。ところがこの 0.30 は 尤度を最大にする値でもあることが上からわかる。

このような尤度関数の値を最大にする値を **最尤推定値** とよぶ。

2.6 EM アルゴリズム

EM アルゴリズムとは

最尤推定が出来ない隠れマルコフモデルや確率文脈自由文法（状態遷移系列や構文解析木が非観測で一意に決定出来ないので）に対して繰り返しにより逐次、対数尤度が増加するようにパラメータを推定する方法。

実際に観測されたデータ y がある。そのデータを生成した内部状態は非観測で一意に決定できない。このデータを不完全データ、対し内部状態が既知であるデータ x を完全データと呼ぶ。この時、完全データから不完全データへの写像を次のように表す。

$$\pi : X \to Y$$

y を生成した x の集合を $X(y)$ 、確率モデルのパラメータを $\phi \in \Phi$ 、x と y の確率密度をそれぞれ $f_\phi(x)$, $g_\phi(y)$ とすると、$g_\phi(y)$ は周辺確率密度として、

$$g_\phi(y) = \infty_{X(y)} f_\phi(x)dx$$

とあらわすことができる。つぎに、以下のような確率密度を定義する。

$$k_\phi(x|y) = \frac{f_\phi(x)}{g_\phi(y)}$$

これにより、パラメータ ϕ に関する対数尤度関数 $L(\phi)$ は次のようになる。

$$L(\phi) \equiv \log f_\phi(x) - \log k_p hi(x|y)$$

また、ϕ に関する期待値を E_ϕ とし、求める新しい推定値を ϕ' とすると、

$$L(\phi) = e_\phi[\log g_\phi(y)|y]$$

$$L(\phi') = E_\phi[\log f_{\phi'}(x)|y] - E_\phi[\log k_{\phi'}(x|y)|y]$$

また $Q(\phi'|\phi)$ および $H(\phi'|\phi)$ を 次のように決めると、

$$Q(\phi'|\phi) = E_\phi[log f_\phi(x)|y] = \frac{1}{g_\phi(y)} \int_{X(y)} \log f_{\phi'}(x) f_\phi(x)dx$$

$$H(\phi'|\phi) = E_\phi[\log k_{\phi'}(x|y)|y] = \frac{1}{g_\phi(y)} \int_{X(y)} \log \infty f_{\phi'}(x) g_{\phi'}(x) f_\phi(x) dx$$

よって

$$L(\phi') - L(\phi) = \{Q(\phi'|\phi) - Q(\phi|\phi)\} + \{H(\phi|\phi) - H(\phi'|\phi)\}$$

新しい推定値 ϕ' は、対数尤度が増加するように決める。$H(\phi|\phi) - H(\phi'|\phi) \geq 0$ となり、$Q(\phi'|\phi)$ が最も増加するように ϕ' を推定すればよいことがわかる。

EM アルゴリズムによるパラメータ推定アルゴリズム

E と M の意味

- **E ステップ：パラメータ のもとで周辺確率分布 を求めること (expectation)**
- **M ステップ： その の下で期待値 を に関する関数とみなして最大化を行なうこと (maximization)**

推定アルゴリズム

1. 初期パラメータ $\phi_0 \in \Phi$ を設定する。
2. 次の **E** ステップと **M** ステップを交互に繰り返す。
   E ステップ ： $Q(\Phi|\Phi_m)$ を計算する。
   M ステップ ： $\phi_{m+1} = \underset{\phi \in \Phi}{\operatorname{argmax}} Q(\phi|\phi_m)$ となるようにϕ_{m+1} を決める。

2.7 HMM のパラメータ推定

HMM のパラメータは、状態遷移系列が非観測であることから、直接、最尤推定することができない。このため、EM アルゴリズムを用いた繰り返しアルゴリズムにより、パラメータを推定する。前向き確率 α に加えて、時刻 t に状態 s_i に滞在し、観測データ $y_t, y_{t++1}, \cdots, y_T$ を生成する後向き確率 β を定義する。

$$p(y_t, y_{t+1}, \cdots, y_T) = \sum_i \beta(i,t)$$

$$\beta(i,t) = \sum_j a_{ij} \cdot b_{ij}(y_t) \cdot \beta(j, t+1)$$

さらに、観測データ y を生成するすべての状態遷移系列の中で、時刻 t に状態 s から s に遷移する回数の期待値を求め、これを全体の確率で正規化した生起確率の期待値 γ を次のように定義する。

$$\gamma(i,j,t) = \frac{\alpha(i,t-1) \cdot a_{ij} \cdot b_{ij}(y_t) \cdot \beta(j,t)}{g_\phi(y)}$$

したがって、EM アルゴリズムより、HMM の各パラメータの最推定値はそれぞれの条件付きの相対確率で表される。

$$\hat{\pi}_i = \frac{\sum_j \gamma(i,j,1)}{\sum_t \sum_j \gamma(i,j,1)}$$

$$\hat{a}_{ij} = \frac{\sum_t \gamma(i,j,t)}{\sum_t \sum_j \gamma(i,j,t)}$$

$$\hat{b}_{ij}(k) = \frac{\sum_{t:y_t=k} \gamma(i,j,t)}{\sum_t \gamma(i,j,t)}$$

この前向き確率と後向き確率を用いてパラメータの最推定を行う方法を、前向き・後向きアルゴリズムと呼んでいる。また、パラメータ学習に N 組の観測データを使用するときには、上記のパラメータ更新を全観測データを用いて一度に行う。例えば、遷移確率の更新は、次のように行われる。

$$\hat{a}_{ij} = \frac{\sum_{n=1}^{N} \sum_t \gamma_n(i,j,t)}{\sum_{n=1}^{N} \sum_t \sum_j \gamma_n(i,j,t)}$$

Chapter 3

形態素解析について

3.1 形態素解析

形態素解析の目的は、文を構成する形態素 (nmorpheme) を認定することである。形態素とは、正確に言えば、言語学的に意味がある (文を構成する) 最小言語単位の事であるが、以下では、辞書に登録されている単語 (単に語とも呼ぶこともある) を形態素と呼ぶこともある。

自然言語解析の通常の流れは、文の形態素解析を行ってから次のステップに進む。文中の形態素が認定できなければ次の処理が不可能なため、形態素解析は自然言語処理の要素技術と言える。

これまでのほとんどの形態素解析の手法は、統語解析と独自のヒューリスティクスに基づくアルゴリズムを採用してきた。次ではこれらの既存の形態素解析の手法を紹介する。

3.2 既存の形態素解析の方法

3.2.1 右方向最長一致法

右方向最長一致法は、文を左から右の向きに見て、もっとも長い形態素を優先させて分割する物である。例えば「アルプスの少女は美しい」を最長一致法を用いて分割してみる。

まず始めに、文全体も最長の形態素の候補であるが、それは辞書にない。そこで最右端の一文字「い」を削除した「アルプスの少女は美し」について同じ事を繰り返す。これも辞書にないから最右端の一文字を更に削除していくと、最終的に「アルプスが右方向最長の形態素として認定される。次は「アルプス」を取り除いた残りの「の少女は美しい」について同様な事を繰り返すと、「の」、「少女」、「は」、「美しい」を得る。このようにして正しい形態素結果が得られる。

3.2.2 文節数最小法

分節数最小法は横型探索をベースにしたヒューリスティクスで、入力文に負組まれるあらゆる単語の候補からなる表を用意し、その表を左から右に走査して、文節数が最小となる単語の系列を優先して出力する方法である。例えば、「くるまでまつ」に対して、あらゆる単語の候補を切り出した表を始めに作る。文節数最小法は、文節数２の単語列を文節数３の単語列に優先させるということである。図からわかるように、文節数最小法のヒューリスティクスは、入力文が短く、文を構成する文節数が少ない場合にはさほど有効に働かない。文節数最小法は、入力文の長さが長いほど有効とされているが、不完全な方法であるという事に変わりがない。またこの方法は膠着語一般に適用可能な物でなく、文節という概念が日本語に特有な物であること、もともと平仮名のべた書き文にたいして考えられた方法であることに注意しなければならない。

3.3 JUMAN

JUMAN は，計算機による日本語の解析の研究を目指す多くの研究者に共通に使える形態素解析ツールを提供するために開発されました.

3.3.1 辞書について

JUMAN で用いられている辞書は，JUMAN2.0 までのバージョンでは，長尾研究室で行われた Mu プロジェクトで開発された辞書，Wnn かな漢字変換システムの辞書，および，ICOT から公開された日本語辞書を利用しました．また，JUMAN3.0 以降のバージョンでは，(株) 日本電子化辞書研究所 (EDR) から許諾を得て EDR 日本語単語辞書の一部を利用しています．

3.3.2 形態素解析アルゴリズム

JUMAN は，EUC コードの日本語文字列を標準入力から一行ごとに読み込んで入力とし，連接規則辞書によって許容された形態素からなる束 (lattice) 状の構造を出力とする．

JUMAN の解析アルゴリズムは，入力としてあたえられた日本語の文字列に対する次の基本動作よりなる．改行をもって一つの入力文字列の終了とする．

- ある特定の位置からはじまるすべての可能な形態素を辞書引きによって得る．
- 辞書引きによって得られた個々の形態素に対して，その直前の位置に存在するすべての形態素との連接可能性のチェック，および，コストの計算を行なう．

3.3.3 未定義語の取り扱い

JUMAN では，入力文字列中のあらゆる位置で未定義語が存在する可能性を考慮している．平仮名および漢字については一文字ずつを一語の未定義語としてきりだす．それ以外の文字については，同種の文字 (カタカナ，アルファベット，数字 等) の終りまでをまとまった一語の未定義語とする．そして，カタカナ文字列には「カタカナ」，アルファベット文字列には「アルファベット」，それ以外には「その他」という品詞細分類を与える．

未定義語という品詞と，その細分類である「カタカナ」,「アルファベット」,「その他」は形態品詞辞書で定義しておかなければならない．連接関係を連接規則辞書で定義すること，コストをリソースファイルの「品詞コスト」欄で定義することは通常の品詞と同様である．

3.3.4 JUMAN を用いた形態素解析の例

JUMAN を用いた形態素解析の一例を示す。次の様なデータに形態素解析をかけてみた。

==========

歩行者優先道路

鈴木健四郎

リレハンメル五輪

延岡市

東大卒

==========

結果は次のようになった。

==========

歩行者	(ほこうしゃ)	歩行者	普通名詞
優先	(ゆうせん)	優先	サ変名詞
道路	(どうろ)	道路	普通名詞
EOS			
鈴木	(すすき)	鈴木	人名
健	(けん)	健	人名
四郎	(しろう)	四郎	人名
EOS			
リレハンメル	(りれはんめる	リレハンメ	地名
五輪	(ごりん)	五輪	普通名詞
EOS			
延	(のべ)	延	普通名詞
岡市	(おかいち)	岡市	人名
EOS			
東	(ひがし)	東	普通名詞
大卒	(だいそつ)	大卒	普通名詞
EOS			

==========

Chapter 4

文字ベースの HMM による複合語の単語分割

4.1 HMM を単語分割利用する

複合語の単語分割は文字間に単語の境界が存在するかしないかのどちらかの記号を割り当てる問題に一般かできる。

ここでもう一度 HMM の定義を示す。

HMM は次の六つで定義できる。

- $\boldsymbol{S}$: 状態の有限集合 ; $\boldsymbol{S} = \{s_i\}$
- $\boldsymbol{Y}$: 出力シンボルの集合
- $\boldsymbol{A}$: 状態遷移確率の集合 ; $\boldsymbol{A} = \{a_{ij}\}$; a_{ij} は状態 s_i から状態 s_j への遷移確率、ここで
  $\sum_j a_{ij} = 1$。
- $\boldsymbol{B}$: 出力確率の集合 ; $\boldsymbol{B} = \{b_{ij}(k)\}$; $b_ij(k)$ は状態 s_i から状態 s_j への遷移の際にシンボル k を出力する確率. ここで、

$$\sum_k b_{ij}(k) = 1(\text{離散 HMM})$$

$$\int_{-\infty}^{\infty} b_{ij}(k)dk = 1(\text{連続 HMM})$$

- π : 初期状態確率の集合 ; $\pi = \{\pi_i\}$; π_i は初期状態が s_i である確率.

$$\sum_j \pi_j = 1$$

- F : 最終状態の集合

図 4.1 に今回単語分割に利用した HMM M を示す。また図 4.2 に今回のモデルでの単語分割の例を示す。

定義に従って示すと、状態の集合Sは単語の境界が存在する (1) としない (0) の 2 つの状態とした。また、出力シンボルの集合 Y として文字を考えた。ここで言う文字とは、普段使っている文字の事である。状態遷移確率 A は考慮せず、初期状態確率 π と最終状態の集合 F は { 1 } とおいた。あとは出力コストの集合 B を設定すれば HMM Mが構築できる。

HMM では出力シンボル系列がどのような状態をたどってきたかを動的計画法の一種である Viterbi アルゴリズムにより推定することができる。すなわち文字間に単語の境界が存在するかしないかが推定でき、単語分割が行える。

また、本当の定義では出力コストではなく出力確率なのだが、状態遷移の尤もらしさを比較できればよいので、得点やコストの形でも不都合はないので出力コストとした。

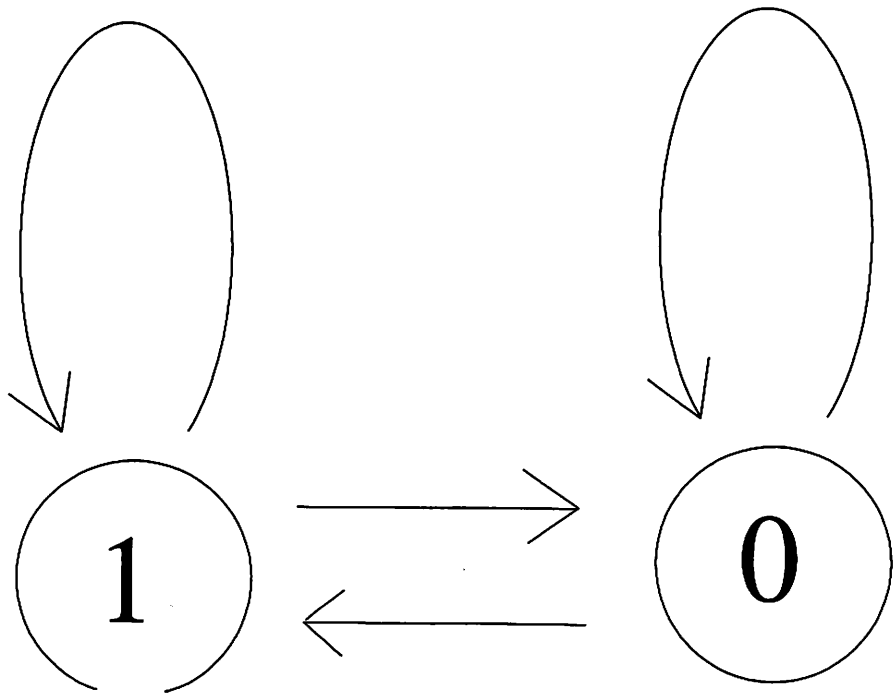

図 4.1　今回使用したHMM

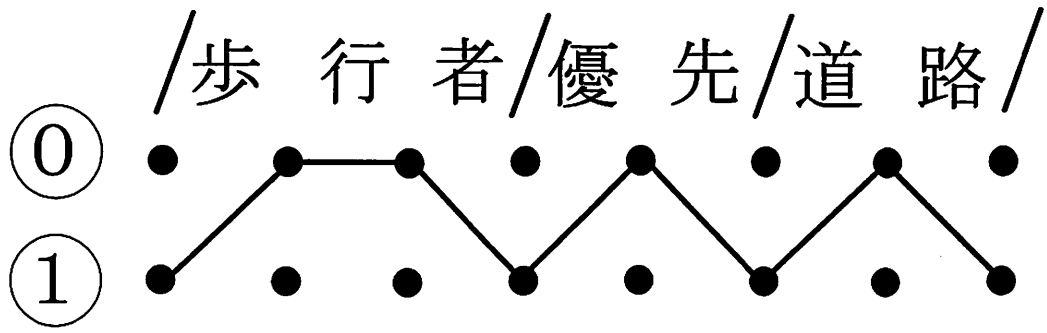

図 4.2　HMMによる単語分割

4.2　文字の出現確率の算出

B を状態 i から 状態 j へ移るとき文字 a が出現するコストとする。この場合、各文字 a に対する $B_{11}(a), B_{10}(a), B_{01}(a), B_{11}(a)$ を求めることが $\boldsymbol{B}$ の構築に対応する。

$B_{ij}(a)$ を求めるために、まず日経新聞 **CD-ROM'90** の **1** 年分の新聞記事を形態素解析し、複合語を取り出した。取り出した複合語の各文字を見ると、その文字の左右に単語境界があるかどうか、つまり状態がわかる。今、文字 a の左の状態が i 、右の状態が j 、であるような個数を $C(i,j)$ とする。例えば「自然言語」という単語は形態素解析により、/自然/言語/ と分割されるので、$C(1,0,\text{自}), C(0,1,\text{然}), C(1,0,\text{言}), C(0,1,\text{語})$ にそれぞれ 1 が加えられる。$B_{ij}(a)$ は以下の式で示した。

$$B_{ij}(a) = \log_2 \frac{C(i,\bar{j},a)+1}{C(i,j,a)+1}$$

ここで $\bar{j}$ は状態 j ではない状態とする。つまり $j=0$ なら $\bar{j}=1$ であり、$j=1$ なら $\bar{j}=0$ である。

上記の形だけでも $\boldsymbol{B}$ は構築できるが、精度をあげるために bigram を利用する。まず

$C(i,j,ab)$ を考える。これは文字 a の左の状態が i、 右の状態が j で更に、文字 a の次の文字が b である個数を示す。この値は先に取り出した複合語から得られる。また $B_{ij}(ab)$ を状態 i から状態 j へ移るときに文字 a が出現し、しかも文字 a の次の文字が b であるときのコストとする。$B_{ij}(ab)$ は以下の式で計算した。

$$B_{ij}(ab) = \log_2 \frac{C(i,\bar{j},ab)+1}{C(i,j,ab)+1}$$

$B_{ij}(a)$ と $B_{ij}(ab)$ から線形和をとり新たな $B'_{ij}(a)$ を以下の様に定義した。

$$B'_{ij}(a) = \alpha \cdot B_{ij}(a) + (1-\alpha) \cdot B_{ij}(ab)$$

ここでは $\alpha = 0.3$ とした。

文字の出現コストの計算例

以下に今回用いたデータから得られた文字の出現コストの計算例を示す。
ただし、**Bi-gram** では、文字列の最初と最後に “$*$” が出力されているものとする。また表 **4.1** に文字の出現コストの例を示す。

A 00 は A というパターンであり **A 01** は A/ でありA **10** は /A というパターン。またA **11** は /A/ となっているパターンである。**C** はデータの中でそれぞれの パターンが出力された回数。ここで “期” について計算例を示す。

$$B_{00}(\text{期}) = \log_2 \frac{931+1}{41907+1} = \log_2 \frac{932}{41908} = \log_2 0.022239191 = 5.49075$$

$$B_{01}(\text{期}) = \log_2 \frac{41907+1}{931+1} = -\log_2 \frac{932}{41908} = -B_{00} = -5.49075$$

$$B_{10}(\text{期}) = \log_2 \frac{12064+1}{17177+1} = 0.509734$$

$$B_{11}(\text{期}) = \log_2 \frac{17177+1}{12064+1} = -0.509734$$

表 4.1:文字の出現コスト

a,i,j	C(i,j,a)	Bij(a)
期 00	931	5.49075
期 01	41907	-5.49075
期 10	12064	0.509734
期 11	17177	-0.509734
欠 00	3	5.45121
欠 01	174	-5.45121
欠 10	455	-3.97490
欠 11	28	3.97490
奏 00	247	1.46786
奏 01	685	-1.46786
奏 10	79	-6.32192
奏 11	0	6.32192
偕 00	0	0.0
偕 01	0	0.0
偕 10	17	-4.08746
偕 11	0	4.08746

a,i,j	C(i,j,a)	Bij(a)
有田 00	15	0.169925
有田 01	17	-0.169925
有田 10	122	-5.35755
有田 11	2	5.35755
伯＊ 00	0	6.61471
伯＊ 01	97	-6.61471
伯＊ 10	0	4.16993
伯＊ 11	17	-4.16993
回落 00	0	3.45943
回落 01	10	-3.45943
回落 10	0	1.58496
回落 11	2	-1.58496
居西 00	3	-2.0
居西 01	0	2.0
居西 10	0	0.0
居西 11	0	0.0

Chapter 5

文字ベースのHMMによる単語分割の修正

5.1 形態素解析が誤るパターン

一般に形態素解析の誤りは未知語によって生じる。日本語の単語はほとんどの場合、2文字あるいは3文字から構成される。このため未知語部分はほとんど以下のように過分割される。

2文字の未知語　○○　→　/○/○/
3文字の未知語　○○○　→　/○/○/○/ OR /○/○○/

また未知語ではないが、JUMANの場合、以下の誤りのケースに共通する特徴として、先頭が1文字で分割されている点がある。つまり形態素解析で1文字単語と認識去れている部分は、誤りである可能性がある。

形態素解析　　正しい分割/○/○○～　　/○○/○～

一方、文字ベースのHMMは、このような局所的な単語分割に有効である。

例で示すと、「鈴木健四郎」の単語分割は、/鈴木/健四郎/であるはずだが、形態素解析では登録されていない「健四郎」が/健/四郎/と過剰に分割される。これは局所的な誤りといえる。

文字ベースのHMMでは、この場合、“健”の前に単語境界があるという仮定の元で“健”と“四”あるいは“健”と“四郎”の接続の強さを測ることになる。“健”と“四”の間の接続は弱そうだが、/健/○/という単語分割のパターンより/健○(健康、健次など)という単語分割のパターンが多いので、結果的に“健”と“四”の間には単語境界を置かずに正解が得られる。表5.1は 形態素解析の誤りの例である。

表5.1:形態素解析の誤りの例

正解	形態素解析の誤り
/鈴木/健四郎/	/鈴木/健/四郎/
/延岡/市/	/延/岡市/
/苑田/	/苑/田/
/河口/利加/さん/	/河/口利/加さん/
/京都府/出身/	/京/都府/出身/
/小泉/順一郎/郵政相/	/小泉/順/一郎/郵政相/
/織田/大次郎/店長/	/織田/大/次郎/店長
/清涼/感/	/清/涼感/
/積極/度/	/積/極度/
/総合/力/	/総/合力/
/罰金/額/	/罰/金額/

5.2 HMMが誤るパターン

文字ベースのHMMでは長い文字列が一単語となるような場合に過剰に分割を行い単語分割が誤る。これは文字ベースが基本的に局所的な部分から、単語の境界があるかどうかを判断するために生じている。例えば、「リレハンメル五輪」の単語分割は/リレハンメル/五輪/ が正解であるが、HMMでは、/リレハン/メル/五輪/と過剰に分割してしまう。これはリレハンメルという単語を局所的に単語境界を判断しているためである。形態素解析では辞書に登録されているので正しく分割できる。

またJUMANでは「一次産品共通基金」という文字列が一単語となっている。この文字列が言語的に一単語かどうかは問題ではない。想定しているアプリケーションにおいて一反語として扱いたいと考え、辞書に一単語として登録してあれば、一単語として解析すべ

きであろう。

さてこの「一次産品共通基金」を単語分割する場合は一単語とするのが正解である。一方「一次産品」を単語分割する場合は、/1/次産品/ と 3 つの単語列として解析するのが正解である。つまり “1” と “次” の間に単語境界が存在するかどうかを判断するには、“1” と “次” だけの局所的な関係では判断できない。さらにこれは “1” と “次産” の関係あるいは “1” と “次産品” の関係などに拡張しても単語境界が存在するかどうかは判断できない。

一方、一般の形態素解析は「一次産品共通基金」を一単語、「一次産品」を /1/次/産品/ と解析するのは容易である。表 5.2 に HMM の誤りの一例を示す。

表 5.2 : HMM での誤りの例

正解	HMM での誤った分割
/引き分け/	/引き/分け/
/やじうま/写真/大賞/	/やじ/うま/写真/大賞/
/高規格幹線道路/	/高/規格/幹線/道路/
/死者/数/	/死/者/数/
/三役/経験者/	/三役/経験/者/
/缶詰め/状態/	/缶/詰め/状態/
/振りそで/姿/	/振り/そで/姿/
/道交法/違反/	/道/交法/違反/

5.3 相補的利用方法

前述したように、形態素解析と文字ベースの HMM のそれぞれの欠点は、互いに補えることが分かる。

本論文では、形態素解析と文字ベースの HMM を相補的に利用した複合語の単語分割を行う。まず形態素解析による単語分割と文字ベースの HMM による単語分割を並行して行い、両者の結果を比較する。基本的には形態素解析の結果を採用するが、以下のパターン部分は、その部分を HMM の結果に修正する。

表 5.3 : 修正するパターン

形態素解析	HMM
/○/○~/	/○○~/

注意として、/○/○～/ と/○○～/ の文字列の長さは等しく、しかも、単語分割のマークである“/”が一致している部分は、最初と最後の部分の2個所しか存在しない。例えば表？の解析結果の下線部分が上記のパターンに当てはまる。

表5.4：パターンの例

HMM	形態素解析
/鈴木/健四郎/	/鈴木/健/四郎/
/延岡/市/	/延/岡市/
/苑田/	/苑/田/
/河口/利加/さん/	/河/口利/加さん/
/京都府/出身/	/京/都府/出身/
/永島/帝二/さん/	/永島/帝/二/さん/
/積極/度/	/積/極度/

Chapter 6

実験

6.1 実験の手順

1. 毎日新聞の新聞記事 ('94 年度版) 1 年分から、複合語を取り出す
2. 形態素解析を用いて単語分割を行う
3. その結果を用いて文字の出現コストを算出する
4. 文字ベースの HMM で単語分割を行う
5. 形態素解析と文字ベースの HMM での単語分割の結果を比較し一致した物としなかった物にわける
6. 一致しなかった物の中で修正が生じた物としなかった物にわける
7. 修正が生じた物のなかで有効な修正だったものと悪影響だった物にわける。

6.2 実験の結果

8543 種類の複合語に対して実験を行い、形態素解析の結果と文字ベースの HMM の結果が一致した物が 7760 種類 (90.8 %)、一致しなかった物は 783 種類 (9.2 %) であった。一致しなかった物の中で修正が生じた物は 212 種類 (2.5 %) であった。一部を表 6.1 に示す。また修正が生じなかった物は 571 種類であった。この 571 種類に対しては、形態素解析の結果を採用する。

修正が生じた 212 種類について調べると、128 種類 (84.0 %) は正解であったが、34 種類 (16.0 %) は不正解であった。ただし、34 種類の中には、形態素解析による単語分割でも誤りがある場合や、正解が曖昧な物が 15 種類あった。純粋に形態素解析の方が正しかった物は 34 種類中 19 種類である。つまり修正による悪影響は、修正した物全体に対して実質 $\frac{19}{212} = 9.0$ % であった。

図 6.1 は実験の結果を図にした物である。

表 6.1：分割の修正

形態素解析	HMM
/延岡/市/	/延/岡市/
/奥谷/喬/司/	/奥谷/喬司/
/追/加点/	/追加/点/
/病/人食/	/病人/食/
/島/内/監督/	/島内/監督/
/若/田光/一/さん	/若田/光一/さん
/若/貴兄/弟/	/若貴/兄弟/
⋮	⋮
/小泉/順一郎/郵政相/	/小泉/順/一郎/郵政相/
/織田/大次郎/店長/	/織田/大/次郎/店長

「若田光一さん」、「若貴兄弟」の例は少し特殊である。一見、今回のパターンでないようだが次の段階で修正出きる

修正の段階（正解：/若貴/兄弟/）

/若/貴兄/弟/

↓

/若貴/兄/弟/

↓

/若貴/兄弟/

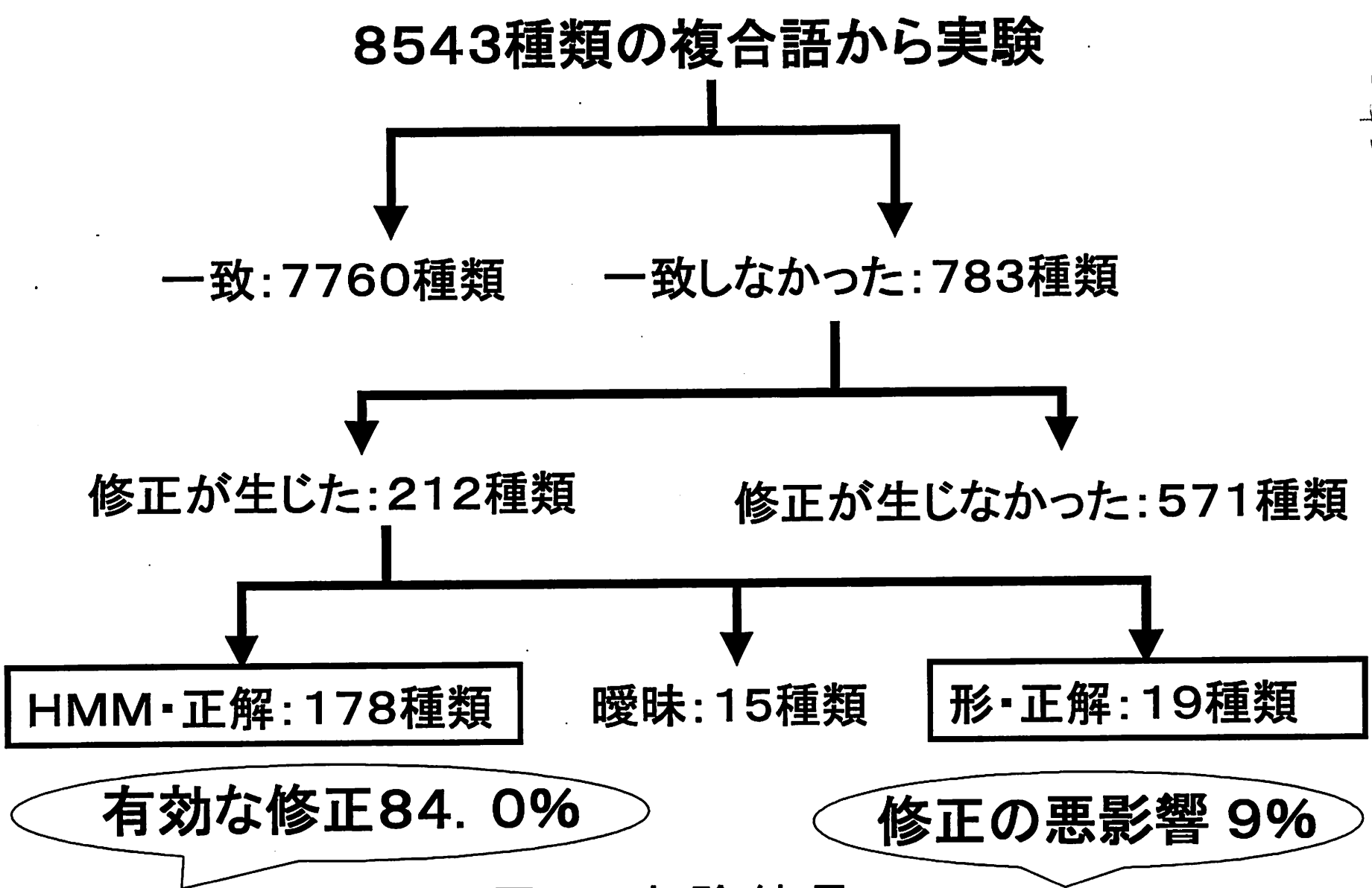
8543種類の複合語から実験
一致:7760種類
一致しなかった:783種類
修正が生じた:212種類
修正が生じなかった:571種類
HMM・正解:178種類
曖昧:15種類
形・正解:19種類
有効な修正84.0%
修正の悪影響 9%

図6.1 実験結果

Chapter 7

考察

ここで提案した手法は未知語に対処することが主な目的である。未知語に対処するための形態素解析を用いない従来手法では、ここで問題とした長い文字列からなる単語を過剰に分割するという欠点が残っている。本手法はこの欠点を克服している点で優れている。

また誤りのカバー率は次の様になった。ランダムに選んだ783種類の中で誤りは28種類あり、そのうち26種類(92.8 %)が本研究の相補敵利用で注目した誤りとなっていた。実験では84.0 %を修正できているので、結果的に誤りの78.0 %を修正できると考えられる。

問題点としては本研究のHMMでは形態素解析の結果をトレーニングデータとして利用してパラメータを得ているため、トレーニングデータの形態素解析の結果の誤りがHMMの精度を悪くしている原因の一つになっていると考えられる。

このことから今後は、今回の手法で得られた結果を用いて、形態素解析の結果を修正することを課題とする。

Chapter 8

おわりに

本研究では複合語の単語分割を行うために、通常の形態素解析と文字ベースの HMM を相補的に利用する手法を提案した。

文字ベースの HMM による単語分割では、長い文字列からなる一単語が過剰に分割されやすい。また形態素解析による単語分割の誤りは、局所的であり、ある単語列のパターンが認められる。このパターンの場合に、文字ベースの HMM による単語分割を採用する。

実験では新聞記事から得た 8543 の複合語に対して単語分割を行った。形態素解析による複合語単語分割の誤りの 178 種類を正しく修正できた。

謝辞

本研究の遂行及び論文の作成において多大な御助言及び御指導を賜った新納 浩幸 教官 (茨城大学工学部システム工学科) に深い感謝の意を表します。

また本研究で利用したコーパスおよび評価文は，日本経済新聞 CD-ROM '90 版と毎日新聞 CD-ROM '94 版から得ています. 利用を許可していただいた日本経済新聞社と毎日新聞社に深く感謝します.

最後に本研究を進めるにあたり助言、協力を頂きました、同研究の河野 靖明 君に感謝致します。

Bibliography

[1] 池谷 昌紀, 新納 浩幸 : “文字ベースの HMM による複合語単語分割の誤り修正”, 言語処理学会、第 5 回年次大会発表論文集, (発表予定) .

[2] 大内弘和 : “既存知識を用いた HMM のパラメータ推定”, 茨城大学システム工学科 '96 年度卒業論文,(1997).

[3] 小田裕樹, 北 研二 : “PPM モデルによる日本語単語分割.Technical Report NL-128-2”, 情報処理学会自然言語処理研究会,(1998).

[4] 北 研二, 中村 哲, 永田昌政 : “音声言語処理-コーパスに基づくアプローチ”, 森北出版株式会社 (1996).

[5] 黒橋禎夫, 長尾真 : “日本語形態素解析システム juman version3.5 使用説明書 ver 2.0”, 京都大学長尾研究室 (1998).

[6] 坂本慶行, 石黒真木夫, 北川源四郎 : “情報量統計学”, 共立出版株式会社 (1993).

[7] 中川聖一郎 : “確率モデルによる音声認識”, 電子情報通信学会 (1988).

[8] 森脇 敏, 河部 恒, 辻井潤一 : “辞書を使わない日本語専門用語の自動分割”, 言語処理学会、第 2 回年次大会発表論文集, pp.273–276 (1996).

[9] 山本幹雄, 増山正和 : “品詞区切り情報を含む拡張文字の連鎖確率を用いた日本語形態素解析”, 言語処理学会、第 3 回年次大会,pp.421-424 (1997).

プログラムリスト

```
/*************************************************************/
/***************前向きパスアルゴリズム*********************/
/*************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

#define S 2                          /* 今回の場合，状態数は 2 */


float bij(int f,int g,char h,char hh);
void quit(char *s);

int n;

main(int argc, char *argv[])
{

   FILE *f1,*fopen();                /* data の入力と
                                        出力確率を求める際に必要 */
```

```
float a[S][S] = {{0.127946401609985,0.264322243777538},
                 {0.264322243777538,0.343409110834939}};
                                  /* 状態遷移確率とその初期設定 */
float pai[S] =  {1.0,0};  /* 初期状態確率とその初期設定  */
int fin[S] = {1,0};       /* 最終状態の集合とその初期設定 */

int i,j,T,t;
float e,Al[S][128],p;
/* 状態はD とC   */

char y[128];  /* これに出力シンボルの系列  abb が入る*/
int r;

if(argc != 2) quit("引数の数が違う"); /* prog  KensakuFile */
if((f1 = fopen(argv[1],"r")) == NULL) quit("データファイルが開けない");
while(fgets(y,128,f1) != NULL) {

  /* ここにに前向きパスアルゴリズムの本体を書く  */

   T = strlen(y);    /* Tをもとめる '\n' をつけると１５   */


 for(i = 0;i < S;i++) /*i=1*/
    for(t = 0;t < T;t=t+2)
       Al[i][t] = 0.0;

   /* 初期値を求める */
   for(i=0;i<S;i++)
      Al[i][0]=pai[i];
```

```
        /* 2.3.4 の実行 */

        for(t = 2;t < T;t+=2){
           for(i=0;i<S;i++){
              for(j=0;j<S;j++){
                  e = bij(j,i,y[t-2],y[t-1]);
                  Al[i][t] += Al[j][t-2]*a[j][i]*e;
              }
           }
        }

       for(i = 0;i < S;i++)
         for(t = 0;t < T;t+=2)
            printf("Al[%d][%d] = %f\n",i,t,Al[i][t]);

       /* 5 最後の計算 */
        for(i=0;i<S;i++)
           if(fin[i]==1)
              p=p+Al[i][T-1];

        y[T-1]='\0';
       /* 結果の出力 */
        printf("%s の結果は \n",y);
        printf(" p= %f\n",p);
        p=0;
    /*   *************************      */
    }
    if((r = fclose(f1)) == -1) quit("データファイルが閉じれない");
}
```

```
/****  End of function main  *****/



void quit(char *s)
{
  printf(s); putchar('\n');
  exit(1);
}



/****  bij を求める。 ***/

float bij(int f,int g,char h,char hh)
{

  FILE *f2;
  char f_name[10],m[128];
  char kakuritu[20];
  float k=0.0,z;
  char ff,gg;
  int n=0,N=0;

/*
  putchar(h);
  putchar(hh);
  printf("---\n");
*/
```

```
ff='0'+f;
gg='0'+g;


f_name[0]=ff;
f_name[1]='.';
f_name[2]=gg;
f_name[3]='\0';

f2=fopen(f_name,"r");

if(f2==NULL){printf("Error \n");exit(0);}

while(fgets(m,128,f2) != NULL){

   if((m[0]==h)&&(m[1]==hh)){
       while(m[n]!=' ')
           n++;
       while(m[n]==' ')
           n++;
       while((m[n]!=' ')&&(m[n]!='\n')){
           kakuritu[N]=m[n];
           n++;
           N++;
           }
       kakuritu[N] = '\0';
       break;
       }
   n=0;
```

```
    N=0;
      }

  n=strlen(kakuritu);

  k =  kakuritu[0] - '0';
  if(kakuritu[1]=='.'){
      for(N=2;N<n;N++){
          z=kakuritu[N]-'0';
          k=k+(z*pow(10,-1*(N-1)));
          }
      }

  fclose(f2);
  return(k);

}


/*  End of program  */



/*****************************************************/
/**************Viterbi アルゴリズム*****************/
/*****************************************************/

/* D が 0 C が 1 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```
#include <math.h>

#define S 2                          /* 今回の場合，状態数は 2 */


float bij(int f,int g,char h,char hh);
float bij2(int ff2,int gg2,char h2,char hh2,char hhh2,char hhhh2,long lll2);
long Index(int ff3,int gg3,char h3,char hh3,char hhh3,char hhhh3);

void quit(char *s);

int n;

main(int argc, char *argv[])
{

   FILE *f1,*fopen();            /* data の入力と
                                    出力確率を求める際に必要 */

   float a[S][S] = {{1.0,1.0},
                    {1.0,1.0}};
                                 /* 状態遷移確率とその初期設定 */
   float pai[S] =  {1.0,0};     /* 初期状態確率とその初期設定  */

   int fin[S] = {1,0};          /* 最終状態の集合とその初期設定 */
   char kkk;
   int last,zzz;
   int i,j,T,t;
   float e,ee,FF[S][128],p;
   int QQ[S][128]  /*, Qi*/;
```

```
char Qi[128];
float max1=-100000.0,max2=0.0;
int argmax1=0,argmax2=0;
float emax,eemax;
float tmp;
long lll;


/* 状態はD とC   */

char y[128];  /* これに出力シンボルの系列  abb が入る*/
int r;

if(argc != 2) quit("引数の数が違う"); /* prog  KensakuFile */
if((f1 = fopen(argv[1],"r")) == NULL) quit("データファイルが開けない");

while(fgets(y,128,f1) != NULL) {



    T = strlen(y);    /* Tをもとめる '\n' をつけると15   */

    y[T-1]=-95;
    y[T]=-10;
    y[T+1]='\n';
    y[T+2]='\0';

    T=T+2;
    for(i = 0;i < S;i++)    /*i=1*/
    for(t = 0;t < T-2;t=t+2)
```

```
            FF[i][t] = 0.0;

        /* 初期値を求める */
         for(i=0;i<S;i++)
             FF[i][0]=pai[i];


        /* 2.3.4 の実行 */
         for(t = 2;t < T-2;t+=2){
             for(i=0;i<=1;i++){
                for(j=0;j<=1;j++){
                    lll=Index(j,i,y[t-2],y[t-1],y[t],y[t+1]);
                    e = bij(j,i ,y[t-2],y[t-1]);
                    ee = bij2(j,i,y[t-2],y[t-1],y[t],y[t+1],lll);
/*                    printf("%c%c%c%c%d%d  e:%f  ee:%f\n"
                           ,y[t-2],y[t-1],y[t],y[t+1],j,i,e,ee);*/
                     tmp=FF[j][t-2]+e*0.3+ee*0.7;
                    if(max1<tmp){
                        max1=tmp;
                        argmax1=j;
                        emax = e;
                        eemax = ee;
                    }
                }
                if((t==2)&&(argmax1==1))  argmax1=0;

                if(argmax1 != 5){
/*                    lll=Index(argmax1,i,y[t-2],y[t-1],y[t],y[t+1]);
                    e = bij(argmax1,i ,y[t-2],y[t-1]);
                    ee = bij2(argmax1,i ,y[t-2],y[t-1],y[t],y[t+1],lll);
```

```
*/
                    e= emax;
                    ee= eemax;
                    FF[i][t]=FF[argmax1][t-2]+e*0.3+ee*0.7;

                    QQ[i][t]=argmax1;
/*                    printf("i = %d , t = %d QQ = %d f = %f \n"
                              ,i,t,QQ[i][t],FF[i][t]); */
                }

            argmax1=5;  max1=-100000;
            }

        }


  /*******************************/


        argmax2=0;

        last=argmax2;          /* int last */

        t=T-3; i=0;


  /***  辿った状態遷移を求める  ****/
        while(t-2 >= 0 ){
            Qi[i]=QQ[argmax2][t]+'0';
```

```
            i++;
            argmax2=QQ[argmax2][t];
            t=t-2;
        }
    Qi[i]='\0';
    i = 0;zzz = 0; /*  int zzz */

/** 逆になっているのを  ならびかえる ***/
    for(i=(strlen(Qi)-1);i>=0;i--){

 /* 状態が 0 すなわち D のとき / をいれる */
        if(Qi[i]=='0') printf("/");
        printf("%c%c",y[zzz],y[zzz+1]);
        zzz+=2;
    }


    if(last==0) printf("/");
    printf("\n");
    argmax2=0;
    max2=0;


   /*   *************************      */

} /*** fgets のおわり***/


 if((r = fclose(f1)) == -1) quit("データファイルが閉じれない");
```

```
}

/****  End of function main  *****/


void quit(char *s)
{
  printf(s); putchar('\n');
  exit(1);
}


/****  bij を求める。 ***/

float bij(int f,int g,char h,char hh)
{

  FILE *f2;
  char f_name[10],m[128];
  char kakuritu[20];
  double  k=0.0,z;
  char ff,gg;
  int n=0,N=0;
  int test1=0;


  ff='0'+f;
```

```
gg='0'+g;


f_name[0]=ff;
f_name[1]='.';
f_name[2]=gg;
f_name[3]='1';
f_name[4]='\0';

f2=fopen(f_name,"r");

if(f2==NULL){printf("Error in Bij\n");exit(0);}

while(fgets(m,128,f2) != NULL){

   if((m[0]==h)&&(m[1]==hh)){
       test1=1;
       while(m[n]!=' ')
           n++;
       while(m[n]==' ')
           n++;
       while((m[n]!=' ')&&(m[n]!='\n')){
           kakuritu[N]=m[n];
           n++;
           N++;
           }

       kakuritu[N] = '\0';
       break;
     }
```

```
    n=0;
    N=0;
  }
if(test1==0)    {fclose(f2); return(0.0);}

n=strlen(kakuritu);
if(kakuritu[0]=='-'){

    if(kakuritu[2]=='.'){
        k =  kakuritu[1] - '0';
        for(N=3;N<n;N++){
            z=kakuritu[N]-'0';
            k=k+(z*pow(10,-1*(N-2)));
        }
        k=k*(-1);
    }
    else if(kakuritu[3]=='.'){
        k =  kakuritu[1] - '0';
        k=k*10;
        k = k + (kakuritu[2] - '0');

        for(N=4;N<n;N++){
            z=kakuritu[N]-'0';
            k=k+(z*pow(10,-1*(N-3)));
        }
        k=k*(-1);
    }
}

else{
```

```
        if(kakuritu[1]=='.'){
            k =  kakuritu[0] - '0';
            for(N=2;N<n;N++){
                z=kakuritu[N]-'0';
                k=k+(z*pow(10,-1*(N-1)));
            }
        }
        if(kakuritu[2]=='.'){
            k =  kakuritu[0] - '0';
            k=k*10;
            k = k + (kakuritu[1] - '0') ;
            for(N=3;N<n;N++){
                z=kakuritu[N]-'0';
                k=k+(z*pow(10,-1*(N-2)));
            }
        }
      }

    fclose(f2);
    return(k);

}


/****  bij2 を求める。 ***/

float bij2(int ff2,int gg2,char h2,char hh2,char hhh2,char hhhh2,long lll2)
{

    FILE *f3,*fopen();
```

```
char buf3[128];
int r3;
char a3[1500][228];
char b3[1500][228];
char ret[100];
int k3;
int kenc3,z3,l3=0;
char kk3[20];
int ii3=0,i3=0,j3=0;
int low3,high3,mid3;
char fname2[25]="0.02";
char astr[200];

astr[0]=h2;
astr[1]=hh2;
astr[2]=hhh2;
astr[3]=hhhh2;
astr[4]='\0';



fname2[0]='0'+ff2;
fname2[2]='0'+gg2;


if((f3 = fopen(fname2,"r3")) == NULL) quit("h4g が開けない");

fseek(f3,lll2,SEEK_SET);
```

```
while(fgets(buf3,128,f3) != NULL) {

  if(ii3==1000) break;
  else{
    /***データの右側の値を配列 b にいれる*************/
    while(buf3[l3]!=' ') l3++;
    while(buf3[l3]==' ') l3++;


    while((buf3[l3]!=' ')&&(buf3[l3]!='\n')&&(buf3[l3]!='\0')){
        b3[i3][j3]=buf3[l3];
        j3++; l3++;
    }
    b3[i3][l3]='\0';


    /***データの左側の値を抜き出し整数に直し*/
    /****** 配列 a にいれる。*******/
    j3=0;
    while(buf3[j3]!=' ') j3++;
    buf3[j3-2]='\0';
    j3=0,l3=0;
    strcpy(a3[i3],buf3);
    i3++;
  ii3++;}

}
```

```
low3=0;high3=ii3;


/****二分探索 ****/
while (low3<=high3){
    mid3=(low3+high3)/2;
    if (strcmp(a3[mid3],astr)<=0)  low3=mid3+1;
    if (strcmp(a3[mid3],astr)>=0)  high3=mid3-1;
}

/****結果の出力*****/
    if(low3==high3+2) {  /*printf("%s \n",b3[mid3]); */
 if((r3 = fclose(f3)) == -1) quit("ファイル1が閉じれない");
       return(atof(b3[mid3])) ; }
          /* データの中にある場合 */


     else { /*printf("%s\n",b3[high3]); */
         if((r3 = fclose(f3)) == -1) quit("ファイル1が閉じれない");
         /*return(atoi(b3[high3]));}*/
         return(0.0);}


/*
  if((r3 = fclose(f3)) == -1) quit("ファイル1が閉じれない");
*/
  /* 返す値を作成して return */

 /*   return(k);*/
```

```
}

long Index(int ff3,int gg3,char h3,char hh3,char hhh3,char hhhh3)
{

  FILE *f4,*fopen();

  char buf2[228];
  int r4;
  char indexname[25]="0.02.index";
  char a2[1500][228];
  char b2[1500][228];
  char bstr[200];

  int i2=0,j2=0,l2=0,k2,ii2;
  int low2,high2,mid2;

  indexname[0]='0'+ff3;
  indexname[2]='0'+gg3;

  bstr[0]=h3;
  bstr[1]=hh3;
  bstr[2]=hhh3;
  bstr[3]=hhhh3;
  bstr[4]='\0';
```

```
if((f4 = fopen(indexname,"r4"))==NULL) quit("index がひらけません");



while(fgets(buf2,128,f4)!=NULL){

   /***データの右側の値を配列 b にいれる*************/
   while(buf2[l2]!=' ') l2++;
   while(buf2[l2]==' ') l2++;
   while(buf2[l2]!=' ') l2++;
   while(buf2[l2]==' ') l2++;


   while((buf2[l2]!=' ')&&(buf2[l2]!='\n')&&(buf2[l2]!='\0')){
        b2[i2][j2]=buf2[l2]; j2++; l2++; }
   b2[i2][l2]='\0';


   /***データの左側の値を抜き出し整数に直し 配列 a にいれる。*******/
   j2=0;
   while(buf2[j2]!=' ') j2++;
   buf2[j2-2]='\0';
   j2=0,l2=0;
   strcpy(a2[i2],buf2);
   i2++;

}

/***<<<<<fgets 終わり>>>>>***/
```

```
  low2=0;high2=i2;


  /****二分探索 ****/
  while (low2<=high2){
      mid2=(low2+high2)/2;
      if (strcmp(a2[mid2],bstr)<=0)  low2=mid2+1;
      if (strcmp(a2[mid2],bstr)>=0)  high2=mid2-1;
  }

  /****結果の出力*****/
      if(low2==high2+2) {  /*printf("%s \n",b2[mid2]); */
              /* データの中にある場合 */
        if((r4=fclose(f4))==-1) quit("ファイルが閉じれないよ");
        return(atol(b2[mid2]));       }

      else /*printf("%s\n",b2[high2]);*/ {
        if((r4=fclose(f4))==-1) quit("ファイルが閉じれないよ");
           return(atol(b2[high2]));}
/*
  if((r4=fclose(f4))==-1) quit("ファイルが閉じれないよ");

  return(****);
*/
}
```

```
/*  End of program  */


/*******************************************************/
/*********形態素解析が正解の場合**********************/
/*******************************************************/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>

main(int argc,char *argv[])
{
  FILE *f1;
  static char buf[300],kk[150],HMM[150];
  char kk_2[100],HMM_2[100];
  char kk_3[100],HMM_3[100];
  int r;
  int posk,posH;
  int sw1=0,sw2=0;
  int i,i1,i2,j,j1,j2,ii,iii,k;
  int l,low,high,count;
  int s,kH,k2;
  int max=0,right=0,left=0;
  int keiOK;

  char slH[50][50],slk[50][50];
  int kanji;
  if(argc !=2) { printf("引数の数が違う"); exit(1); }
  if((f1 = fopen(argv[1],"r"))==NULL) exit(1);
```

```
while(fgets(buf,300,f1)!=NULL){
   if(buf[0]=='1'){
      i=0;
      while(buf[i]!=' ')
          i++;
      while(buf[i]==' ')
          i++;
      l=i;
      while(buf[i]!=' '){
          kk[i-l]=buf[i];i++;
      }
      kk[i-l]='\0';
      while(buf[i]==' ')
          i++;
      l=i;
      while(buf[i]!='\n'){
          HMM[i-l]=buf[i];i++;
      }
      HMM[i-l]='\0';

      i=0;kH=0;
      kanji=0;

      while(HMM[i]!='\0'){
          if(HMM[i]!='/'){
              i+=2; kanji++;
          }
          else /*if(HMM[i]=='/')*/{
              if(kanji<10){
                  slH[kH][0]='0';
```

```
                slH[kH][1]=kanji+'0';
                slH[kH][2]='\0';
                kH++;i++;
            }
            else {
                slH[kH][0]= kanji/10+'0';
                slH[kH][1]=(kanji-kanji/10*10)+'0';
                slH[kH][2]='\0';
                kH++;i++;}
            }
        }

i=0;k2=0;
kanji=0;
while(kk[i]!='\0'){
    if(kk[i]!='/'){
        i+=2; kanji++;
    }
    else if(kk[i]=='/'){
        if(kanji<10){
            slk[k2][0]='0';
            slk[k2][1]=kanji+'0';
            slk[k2][2]='\0';
            k2++;i++;
        }
        else {
            slk[k2][0]= kanji/10+'0';
            slk[k2][1]=(kanji-kanji/10*10)+'0';
            slk[k2][2]='\0';
            k2++;i++;
```

```
      }
  }
}
s=0;

keiOK=0;
for(i=0;i<k2-1;i++){
    left=0; right=0;
    for(j=0;j<kH;j++) {
        if(strcmp(slk[i],slH[j])==0){  left=1; j1=j;}
        if(strcmp(slk[i+1],slH[j])==0) {right=1; j2=j;}
    }
    if((left==1)&&(right==1)){
         keiOK++;
         if((j2-j1)==1) ; /*同じということ 勘違いするな*/


         else {j=0;count = 0;

             while(count<atoi(slH[j1])){
                 if(HMM[j]=='/') j++;
                 else if(HMM[j]!='/') {count++;j+=2;}

             }

             for(i2=0;i2<((j2-j1)-1);i2++){
                 i1=j;
                 while(HMM[i1]=='/') i1++;
                 while(HMM[i1]!='/') i1++;
```

```
                        while(HMM[i1-1]!='\0'){
                            HMM[i1]=HMM[i1+1];
                            i1++;
                        }
                    }
                    /*printf("1   %s    %s \n",kk,HMM);*/
                }
            }

        }
/*          if(keiOK==k2-1) printf("kei  %s\n",kk); */
          printf("1   %s   %s\n",kk,HMM);

     }
    else printf(buf);
  }

  if((r=fclose(f1))==-1) exit(1);
}
```

```
/*************************************************/
/***************形態素解析が誤るパターン********/
/*************************************************/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
```

```
main(int argc,char *argv[])
{
  FILE *f1;
  static char buf[300],kk[150],HMM[150];
  char kk_2[100],HMM_2[100];
  char kk_3[100],HMM_3[100];
  int r;
  int posk,posH;
  int sw1=0,sw2=0;
  int i,i1,i2,j,j1,j2,ii,iii,k;
  int l,low,high,count;
  int s,kH,k2;
  int max=0,right=0,left=0;
  int keiOK;

  char slH[50][50],slk[50][50];
  int kanji;
  if(argc !=2) { printf("引数の数が違う"); exit(1); }
  if((f1 = fopen(argv[1],"r"))==NULL) exit(1);
  while(fgets(buf,300,f1)!=NULL){
     if(buf[0]=='1'){
        i=0;
        while(buf[i]!=' ')
            i++;
        while(buf[i]==' ')
            i++;
        l=i;
        while(buf[i]!=' '){
            kk[i-l]=buf[i];i++;
        }
```

```
kk[i-1]='\0';
while(buf[i]==' ')
    i++;
l=i;
while(buf[i]!='\n'){
    HMM[i-1]=buf[i];i++;
}
HMM[i-1]='\0';

i=0;kH=0;
kanji=0;

while(HMM[i]!='\0'){
    if(HMM[i]!='/'){
        i+=2; kanji++;
    }
    else /*if(HMM[i]=='/')*/{
        if(kanji<10){
            slH[kH][0]='0';
            slH[kH][1]=kanji+'0';
            slH[kH][2]='\0';
            kH++;i++;
        }
        else {
            slH[kH][0]= kanji/10+'0';
            slH[kH][1]=(kanji-kanji/10*10)+'0';
            slH[kH][2]='\0';
            kH++;i++;}
        }
    }
```

```
i=0;k2=0;
kanji=0;
while(kk[i]!='\0'){
    if(kk[i]!='/'){
        i+=2; kanji++;
    }
    else if(kk[i]=='/'){
        if(kanji<10){
            slk[k2][0]='0';
            slk[k2][1]=kanji+'0';
            slk[k2][2]='\0';
            k2++;i++;
        }
        else {
            slk[k2][0]= kanji/10+'0';
            slk[k2][1]=(kanji-kanji/10*10)+'0';
            slk[k2][2]='\0';
            k2++;i++;
        }
    }
}
s=0;

keiOK=0;
for(j=0;j<kH-1;j++){
    left=0; right=0;
    for(i=0;i<k2;i++) {
        if(strcmp(slH[j],slk[i])==0){  left=1; i1=i;}
        if(strcmp(slH[j+1],slk[i])==0) {right=1; i2=i;}
```

```
}
if((left==1)&&(right==1)){
     keiOK++;
     if((i2-i1)==1) ; /*同じということ 勘違いするな*/

     else {i=0;count = 0;
         while(count<atoi(slk[i1])){
             if(kk[i]=='/') i++;
            else if(kk[i]!='/') {count++;i+=2;}
         }

        /* for(j2= 0;j2<((i2-i1)-1);j2++){
             j1=i;
             while(kk[j1]=='/') j1++;
             while(kk[j1]!='/') j1++;
             while(kk[j1-1]!='\0'){
                 kk[j1]=kk[j1+1];
                 j1++;
             }
         }*/
/***************************/
        if(kk[i+3]=='/'){
             j1=i+3;
             while(kk[j1-1]!='\0'){
                 kk[j1]=kk[j1+1];
                 j1++;
             }
         }
/*****************************/
         /*printf("1   %s    %s \n",kk,HMM);*/
```

```
                    }
                }

            }
/*          if(keiOK==k2-1) printf("kei  %s\n",kk); */
          printf("1   %s   %s\n",kk,HMM);

      }
     else printf(buf);
  }

  if((r=fclose(f1))==-1) exit(1);
}
```