括弧制約を利用した統語解析精度の向上[1]

宮田高志 †　　橋田浩一 ††
miyata.t@carc.aist.go.jp　　hasida.k@aist.go.jp
† 独立行政法人 科学技術振興機構, CREST
‡ 独立行政法人 産業技術総合研究所 情報技術研究部門

1　括弧制約下のまとめあげ解析

この論文では、Hidden Markov Model に代表されるような、文を固定された長さの履歴だけに依存して逐次的に出力された語の列と見なすような言語モデルを持つ解析器に対して、括弧で制約された入力を言語モデルに矛盾することなく扱えるように拡張する方法を提案する。さらに、形態素解析器「茶筌」[4]・文節まとめあげ解析器 YamCha [2]・文節係り受け解析器「南瓜」[3] に対して提案する拡張を実際に施し、付与した括弧の影響を評価する。

1.1　IOB モデル

まとめあげ解析は、語に B または I のラベルを付与することに帰着できる [2, 5]。ここで、B はまとまりの開始を、I はまとまりの内側または最後を表す[2]。形式的には次のように定義される:

- 文中の各語に対して、B と I の二種類の状態を対応させる。語は、状態の種類に応じてあらかじめ決められた分布 $\Pr(w_i \mid L)$ に基づいて出力されるとする。ここで w_i は i 番目の語、L は B または I である。
- 状態 L_{i-1} から状態 L_i への遷移は条件付き確率 $\Pr(L_i \mid L_{i-1}, L_{i-2}, \ldots, L_{i-h})$ によって決まる。ここで L_i は i 番目の状態、h はあらかじめ決められた履歴の長さである。

与えられた語の列に対して最も確率の高い状態遷移列、すなわち最も確率の高いラベル列を求めるには、Viterbi アルゴリズムを用いる。この言語モデルでは、二種類の状態間の 4 通りの遷移 (B→B, B→I, I→B, I→I) が全て許される。

次の例は「逐次調査を依頼」という句に対する可能なまとめあげ方を、この言語モデルによって表現する方法を示している:

1. (逐次　調査　を)　(依頼)
   B　I　I　B
2. (逐次)　(調査　を)　(依頼)
   B　B　I　B

このような曖昧性をもつ句に対して、次の括弧は可能なまとめあげ方を制限し、結果として 2. の可能性を排除する:

3. (逐次　調査)　を　依頼

この括弧は次の解析の可能性は排除しないことに注意されたい。

4. ∗ (逐次)　(調査)　(を)　(依頼)
   B　B　B　B

4. が解析結果として採用されないのは、実際の現象として「を」がまとまりの開始位置には出現しにくいためである。つまり「括弧制約を正しく扱う」とは、付与した括弧と矛盾する解候補だけを排除し、矛盾しない解候補は言語モデルで計算される確率の大きさで優劣を判断するということである。

1.2　まとめあげ解析の拡張

このような括弧制約を扱うために、先の言語モデルを拡張して、括弧で囲まれた語の列に対しても状態を対応させる。図 1 は括弧が付与された語の列 w_1 ((w_2 w_3 w_4) w_5 (w_6) w_7) w_8 に対する状態遷移図である (ただし、簡単のため到達不能な状態は省略してある)。#は開始状態、$は終了状態を表す。各状態の添字は括弧の**深さ**を表す。例えば、(w_2, w_3, w_4) に対応する B というラベルをもつ状態は、図中で左中あたりに中くらいの大きさの B_1 という四角形で表されているが、その添字である 1 は、この三語を囲む括弧の外側にもう一つ別の括弧があることを意味する。以下では、括弧で囲まれた語の列に対応する状態のことを**括弧状態**、通常の語に対応する状態のことを**語状態**とよぶことにする。図 1 には、開始状態と終了状態を除いて 5 つの括弧状態と 13 個の語状態が描かれている。

可能なパスに括弧制約を反映させるために、状態間の遷移を次のように制限する:

- 開始状態 (#) の直後は B でなければならない。
- i 番目の状態 L_i とその次の状態 L_{i+1} の間に左括弧がある場合、次の遷移が可能である:

[1] Improvement of Parsing Accuracy with Bracketing Constraints, MIYATA Takashi† and HASIDA Kôiti†‡ (†Japan Science and Technology Agency, CREST, ‡National Institute of Advanced Industrial Science and Technology)

[2] オリジナルのアルゴリズムではまとまりの外側を表す O というラベルも用いるが、ここでは枝分かれのない句のバーレベルを無視することで、O は使わない。またラベルをさらに詳細化することで、品詞や統語範疇のような情報を持たせることもあるが、ここでは簡単のために省略する。

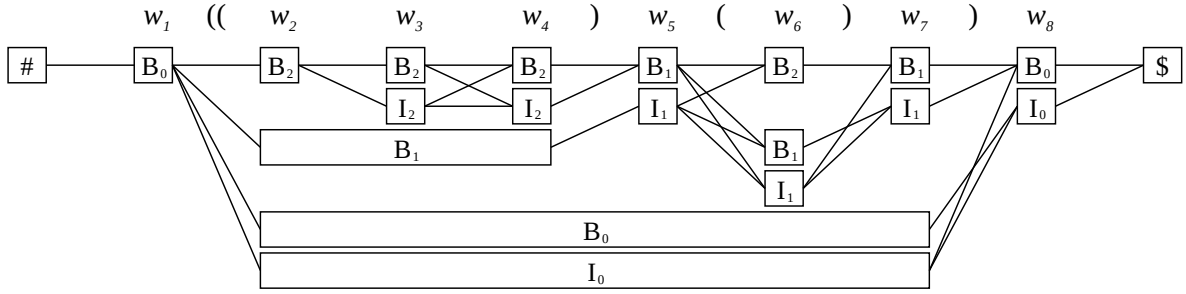

図 1: 括弧制約の下でのまとめあげ解析における束構造

(	$L_{i+1} = \mathrm{I}$	$L_{i+1} = \mathrm{B}$
$L_i = \mathrm{I}$	OK	OK
$L_i = \mathrm{B}$	深さ一致	OK

L_i が B で L_{i+1} が I の時は、二つの状態の深さが一致している時のみ、遷移できる。

- i 番目の状態 L_i とその次の状態 L_{i+1} の間に右括弧がある場合、次の遷移が可能である:

)	$L_{i+1} = \mathrm{I}$	$L_{i+1} = \mathrm{B}$
$L_i = \mathrm{I}$	深さ一致	OK
$L_i = \mathrm{B}$	深さ一致	L_i が語状態

L_{i+1} が I の時は L_i と L_{i+1} が同じ深さの時のみ、遷移できる。また、L_i と L_{i+1} が共に B の時は L_i が語状態、すなわち L_i と終点を共有する状態の中で L_i が最も深い時のみ、遷移できる。

- 上記以外の場合は自由に遷移可能とする。

さらに、w_i から w_{i+j} を覆う括弧状態 S は、語の列 $(w_i, w_{i+1}, \ldots, w_{i+j})$ を次の確率で生成すると定義する:

$$\prod_{k=0}^{j} \left\{ \Pr(w_{i+k} \mid L_{i+k}) \times \sum_{\text{path}} \Pr(L_{i+k} \mid L_{i+k-1}, \ldots, L_{i+k-h}) \right\} \quad (1)$$

上の計算において、各括弧状態は語状態の列とみなしていることに注意。すなわち、式 (1) 中の総和において、括弧状態 S が B (I) なら、$L_i, L_{i+1}, \ldots, L_{i+j}$ を $L_i = \mathrm{B}$ (I), $L_{i+1} = \ldots = L_{i+j} = \mathrm{I}$ に固定し、$L_{i-1}, L_{i-2}, \ldots, L_{i-h}$ を可能なパス上で動かして和をとることを意味する。

図 1 から分かるように、括弧の深さの最大値を d とすると、高々 $2(d+1)$ 個の括弧状態が追加される。また、状態 L_i から L_{i+1} への遷移に対して、L_i において長さ $h-1$ の履歴を保存する必要があるので、2^{h-1} の記憶領域が必要である。よって、文の長さを n とすると、このアルゴリズムの計算量は、$O(\{2(d+1)\}^2 2^{h-1} n) = O(n(d+1)^2 2^{h+1})$ である。

2 括弧制約による効果の上限

YamCha と南瓜を上で説明したように修正し、解析器と一緒に配布されているモデルファイルのうちの小さい方、すなわち京大コーパス中の 7,615 文で学習したものを使い、残りの 30,768 文で括弧制約による効果の上限を評価した。評価尺度は下記の F 値とした。

$$\frac{2 \times (\text{bracket accuracy}) \times (\text{bracket recall})}{(\text{bracket accuracy}) + (\text{bracket recall})} \quad (2)$$

ここで bracket accuracy とは解析器が出力した括弧のうち、正解に含まれているものの割合、bracket recall とは正解に含まれている括弧のうち、解析器が出力したものの割合である。オリジナルの YamCha と南瓜の F 値は、それぞれ 0.9750 と 0.9143 である。

2.1 評価実験

次の手順に従って、解析誤りを含む文に括弧を付与することで最大どのくらい誤りを減少させられるかを調べる。

1. すでに正しく解析されている文または正解よりも多くの括弧を含んでいるような文を評価セットから取り除く。以下では、後者のような文を**過分割された文**とよぶ。
2. 残りの文について、次のように括弧を付与して解析木を得る:
   i. 正解から解析結果に含まれていないような括弧を一つ選び、付与する。正解と比べて、解析結果に m 個の括弧が足りない場合、これによって m 通りの括弧付き文が得られる。
   ii. m 通りの括弧付き文をそれぞれ解析し、最も誤りが少なかったものをその文に対する解析結果とする。以下では、この時の括弧をその文に対する**最適な括弧**とよぶ。
3. 文全体に対する F 値を計算する。以下では、この手順一回を**ステージ**とよぶ。

正解に達した文および過分割された文を除きつつ、上の手順を繰り返して 2, 3, 4, 5 個の括弧を付与してゆく。正

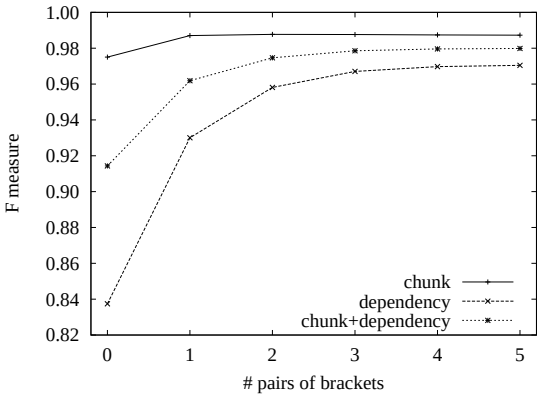

図 2: 最適な括弧の数に対する F 値

stage	解析	正解	過分割
0	30768	13894	29
1	16845	8598	818
2	7429	3377	587
3	3465	1163	466
4	1836	382	397
5	1057	107	276

表 1: 各ステージにおける文数

解に比べて解析結果に m 個の括弧が足りない場合、上の手順の二番目ではそれぞれ $\binom{m}{2}$, $\binom{m}{3}$, $\binom{m}{4}$, $\binom{m}{5}$ 通りの文が作られることになる。

2.2 結果

付与した最適な括弧の数に対する F 値のグラフを図 2 に示す。図から最適な括弧をわずか二つ付与しただけで F 値が 0.9143 から 0.9746 に向上することが読みとれる。また、括弧二つないし三つくらいで F 値は飽和していることも分かる。

表 1 に、各ステージで解析された文・正解となった文・過分割された文の数を示す。表によると、4 つの括弧を付与した段階で、正解となった文より過分割された文の方が多くなることが分かる。これは図 2 で F 値が飽和することと合致する。

3 括弧制約の応用

前節で説明した最適な括弧は、解析器の ‘弱点’ を補う情報とみなすことができる。もし最適な括弧を何らかの方法で汎化できれば、未知の文に対しても解析精度を向上させることが期待できる。

3.1 アルゴリズムと評価実験

ここでも京大コーパスの 7,615 文を訓練セットとして使う。

1. 前節で説明した手順を、訓練セットに対して適用する。
2. 最適な括弧を、前後の二形態素とともに収集する。すなわち、形態素の列 $(w_i, w_{i+1}, \ldots, w_j)$ に付与された最適な括弧に対して $w_{i-2}, w_{i-1}, (w_i, \ldots, w_j), w_{j+1}, w_{j+2}$ を収集する。ただし、n を文の長さとして、$k < 1$ もしくは $n < k$ なる w_k に対しては、$w_k =$ # もしくは $w_k =$ \$ とする。以下では、このような形態素列を**素パターン**とよぶ。
3. 素パターンに含まれる各品詞の頻度を計測し、各品詞に対して適当な汎化レベルを人手で決める。
4. 品詞の汎化レベルに従って素パターンを汎化する。このようにして得たパターンを以下では**括弧付与ルール**とよぶ。
5. 評価セットに対して括弧付与ルールを適用し、解析を行なう。もし互いに交差するような括弧付与ルールが二つ以上適用可能だった時は、最も狭い括弧を優先する。幅が等しい時は最も左のものを優先する。

前節で見たように、多くの括弧を付与しても必ずしも精度は向上しない。そこで、ステージ 1 および 2 で得られた 825 個の素パターン (延べ 837 回) のみを使うことにした。この中に出現する品詞のうち、頻出する 21 種類について汎化レベルを決め、残りの 40 種類については最も粗いレベルとした。この結果、805 個の括弧付与ルールを得た。

括弧付与ルールは、評価セットに対して 309 回 (302 文) 適用できた。これは評価セット全体に対して約 1%であり、全体の精度にはほとんど影響がないが、適用された文だけで見ると、精度は大きく向上していることが分かった。

	文節	係り受け	両方
括弧なし	0.9057	0.7050	0.8153
括弧あり	0.9539	0.7926	0.8809

3.2 括弧付与ルールの例

図 3 に訓練セット中で解析誤りを含んでいた文およびその文に対する最適な括弧の例を、表 2 にそれから得られた括弧付与ルールをそれぞれ示す。図 3 では、各形態素を / で区切り、文節および係り受けを丸括弧で表す。最初の行は解析器の出力であり、次の行が正解である。最適な括弧は角括弧 ([ ]) で表す。また表 2 は、形態素 (原形)・品詞・汎化した品詞の三つの列からなる。左の二つの列が素パターン、右の一列が括弧付与ルールに相当する。

表 2 の最初の素パターンおよび括弧付与ルールは、図 3 の最初の文から抽出されたものである。この文に対する最適な括弧は、「その」と「場」を別々の文節にするかどうかに関する誤りを修正するものである。この種の

1. * ((((ブザー/が) (鳴っ/た)) (瞬間/、)) ((その/場/に) (座り込ん/だ)))
((((ブザー/が) (鳴っ/た)) (瞬間/、)) (((その) [場/に]) (座り込ん/だ)))

2. * (((((３０/カ国/を) (対象/に)) (し/て)) (行っ/た)) (分析))
((((３０/カ国/を) [(対象/に) (し/て)]) (行っ/た)) (分析))

図 3: 訓練セット中の解析誤りと最適な括弧

1.

素パターン		括弧付与ルール
、	記号-読点	記号-読点
その	連体詞	連体詞
# <x>		
場	名詞-一般	名詞-一般
に	助詞-格助詞-一般	助詞-格助詞
# </x>		
座り込む	動詞-自立	動詞-自立
だ	助動詞	助動詞

2.

素パターン		括弧付与ルール
カ国	名詞-接尾-助数詞	名詞-接尾
を	助詞-格助詞-一般	助詞-格助詞-一般
# <x>		
対象	名詞-一般	名詞-一般
に	助詞-格助詞-一般	助詞-格助詞-一般
する	動詞-自立	動詞-自立
て	助詞-接続助詞	助詞-接続助詞
# </x>		
行く[3]	動詞-非自立	動詞
た	助動詞	助動詞

表 2: 素パターンと括弧付与ルールの例

括弧付与ルールは、主に訓練データの不足に起因する誤りを局所的に修正するもので、最も多く適用された (34 文)。これらの文のルール適用後の F 値は 0.7955 である。

また、表 2 の二つ目の素パターンおよび括弧付与ルールは、図 3 の二つ目の文から抽出されたものである。文中の 5 つの文節は正しく解析されているが、それらの間の係り受けに誤りがある。実際には「男性を対象に調査を行なう」のように、ヲ格が「対象に」に係ることもある[4]ので、より広い範囲を見なければ正しく解析できない。この括弧付与ルールは 18 文に対して適用され、これらの文のルール適用後の F 値は 0.6945 であった。

4 考察とまとめ

図 2 に示されているように、最適な括弧でもその効果は二つないし三つで飽和してしまう。これは表 1 から分かるように、括弧を付与するにしたがって正解となる文よりも過分割された文の方が多くなるからである。

3 節では、訓練セットから得られた最適な括弧を汎化して、未知の文に対する括弧付与ルールを抽出した。なお今回は個々の括弧付与ルールを独立に適用した。ルールの適用率が低く、ほとんどの文にについて高々一つしか適用されていないので、このことはあまり問題とはならなかったが、アルゴリズムとしては最適な括弧の組合せも得られるので、より大きな訓練セットを用いてより複雑なルールを得ることも可能である。また今回は訓練セットから得られた素パターンの数が少なかったので人手で汎化したが、理想的にはより大きな訓練セットを用いてパターンマイニング [1] のような系統的な方法を用いるべきである。

我々の方法は、独立に開発されている自然言語解析モジュールを、曖昧性の不要な展開をできるだけ避けるように、かつできるだけ少ない変更で有機的に連携させるために有効である。括弧制約を利用することで、例えば、固有名詞を同定するモジュールと構文解析モジュールを簡単に連携させることが可能となる。

[3] これはコーパスの誤りで、正しくは「行なう / 動詞-自立」

[4] 「して」が省略されていると分析することも可能

参考文献

[1] Tatsuya Asai, Kenji Abe, Shinji Kawasoe, Hiroki Arimura, Hiroshi Sakamoto, and Setsuo Arikawa. Efficient substructure discovery from large semi-structured data. In *Proceedings of the 2nd Annual SIAM Symposium on Data Mining*, pp. 158–174, April 2002.

[2] Taku Kudo and Yuji Matsumoto. Chunking with support vector machines. In *Proceedings of the 2nd Meeting of NAACL*, USA, 2001.

[3] Taku Kudo and Yuji Matsumoto. Japanese dependency analysis using cascaded chunking. In *CoNLL 2002: Proceedings of the 6th Conference on Natural Language Learning 2002 (COLING 2002 Post-Conference Workshops)*, pp. 63–69, 2002.

[4] Yuji Matsumoto and et al. *Japanese Morphological Analysis System ChaSen version 2.2.1*. Computational Linguistics Laboratory, Graduate School of Information Science, Nara Institute of Science and Technology, December 2000.

[5] Erik F. Tjong Kim Sang and Jorn Veenstra. Representing text chunks. In *Proceedings of the 6th Conference of the European Chapter of the ACL*, pp. 173–179, Bergen, Norway, June 1999.