Spinal TAG のための高速な構文解析

木曽 鉄男 [1]　　林 克彦 [2]　　新保 仁 [1]　　松本 裕治 [1]
[1] 奈良先端科学技術大学院大学 情報科学研究科
[2] 日本電信電話株式会社 NTT コミュニケーション科学基礎研究所

[1]{tetsuo-s,shimbo,matsu}@is.naist.jp
[2]hayashi.katsuhiko@lab.ntt.co.jp

1　はじめに

木接合文法 (Tree Adjoining Grammar; TAG) [7] は句構造木を基本要素 (基本木) とし, 木に対する置換または接合操作を繰り返し適用することで, 構文木を構成する形式文法である. 近年, TAG の一種である spinal TAG による構文解析が高い解析精度を達成している [10, 1].

文献 [1] における spinal TAG では, 葉に単語を持ち, その単語が主辞となる句を表す前終端記号及び非終端記号の Unary 系列 (spine) を基本木とする. そして, この基本木に対して, 2 種類の接合操作: sister adjunction と regular adjunction を行うことで解析を進める.

Spinal TAG の基本木は, 構文木に対して主辞規則 (例えば [3]) を適用した後に獲得される. 図 1 に構文木の例と, その構文木から獲得される基本木の例を示す. 図 2 に接合操作の例を示す. 図では, 接合される基本木のすべての非終端記号に対して, 前終端記号を始点として, インデックス付けしている [1]. 本稿では @ を使って, このインデックスを明示する.

文献 [1] で提案された構文解析法は高い解析精度を誇る一方で, 計算量が $O(n^4G)$ (n は文長, G は文法の大きさ) であり, 実応用を考えた場合には現実的な計算量とは言えない. そこで本稿では, spinal TAG に対するより高速な構文解析法を提案する. 提案法では, 動的計画法に基づく shift-reduce 法 [6] を spinal TAG へ拡張することで, 高精度かつ高速な解析を実現する [2]. さらに, 構文解析の前処理として supertagging を行うことで, 解析速度のさらなる高速化を試みる.

[1]ただし, regular adjunction の場合は, 接合操作時に, 接合される非終端記号と同じ記号が新たに親となり, 構文木が構成されるため, 便宜上インデックスは変えていない.

[2]文献 [10] では, TAG に対する漸進的構文解析法を提案しているが, 本研究とは異なる種類の TAG に基づくため, 以後議論しない.

2　構文解析アルゴリズム

木接合文法に対する shift-reduce 法の状態は

$$[\ell, i, j, \sigma] : \pi \tag{1}$$

として定義できる. ℓ はステップ数, σ はスタック, i, j はスタック先頭要素のスパンを表す. π は動的計画法 [6] を適用するための予測前状態へのポインタ集合となるが, 提案法の理解には関係ないため, 説明を省略する.

スタック要素 s は 3 つの変数を持つ集合で表す.

$$s = \langle HS, start, isRegular \rangle. \tag{2}$$

HS は主辞となる基本木を表す変数である. $start$ は HS が表す基本木への接合が可能な非終端記号の開始位置を表す変数で, 1 節で導入した基本木上のインデックスを表す. $isRegular$ は s を構築したアクションが regular adjunction であったかを表す 2 値変数である [3].

キューにセットされた入力文 $x_1 \ldots x_n$ が与えられたとき, 提案手法の解析は公理 $[0, 0, 1, \epsilon] : \emptyset$ から始まり, 以下 6 つのアクションを使って解析を進め, finish によって最終状態へと至り終了する. 各アクションは演繹推論規則を使って定義する.

- shift: キューの先頭から単語を一つ取り出し, 基本木 h を割り当てて, スタックの先頭に積む.

$$\frac{\overbrace{[\ell, i, j, \sigma] : \pi}^{p}}{[\ell + 1, j, j + 1, \sigma | s_0] : \{p\}}\ .$$

ここでは, $s_0.HS = h$, $s_0.start = @2$, $s_0.isRegular = false$ となる. $start$ を最下層の非終端記号の位置インデックス @2 としているのは, 前終端記号への接合を禁止するためである.

[3]$isRegular$ はバックトレースして構文木を構築する際, sister と regular adjunction を区別するために使われる.

- sister adjunction left: $s_1.HS$ が示す基本木を, $s_0.HS$ が示す基本木へと接合する. 接合する位置は変数 x で表す.

$$\frac{\overbrace{[\_, i, k, \sigma' | s_1' | s_1] : \pi'}^{p} \quad [\ell, k, j, \sigma | s_1 | s_0] : \pi}{[\ell + 1, i, j, \sigma' | s_1' | s_1 \overset{s}{\rightarrow}^{x} s_0] : \pi'} \quad p \in \pi \wedge s_0.start \leq x \leq height(s_0.HS)$$

ここで $s_1 \overset{s}{\rightarrow}^{x} s_0$ によってできる新たなスタック要素の各変数は, $HS = s_0.HS$, $start = x$, $isRegular = false$ となる. $height(h)$ は基本木 h の高さを返す関数である.

- regular adjunction left: sister adjunction left とほぼ同様の定義であり, 次のようになる.

$$\frac{\overbrace{[\_, i, k, \sigma' | s_1' | s_1] : \pi'}^{p} \quad [\ell, k, j, \sigma | s_1 | s_0] : \pi}{[\ell + 1, i, j, \sigma' | s_1' | s_1 \overset{r}{\rightarrow}^{x} s_0] : \pi'} \quad p \in \pi \wedge s_0.start \leq x \leq height(s_0.HS)$$

ここで $s_1 \overset{r}{\rightarrow}^{x} s_0$ によってできる新たなスタック要素の各変数は, $HS = s_0.HS$, $start = x$, $isRegular = true$ となる.

- sister/regular adjunction right: sister/regular adjunction left の操作を $s_1 \overset{s}{\leftarrow}^{x} s_0$, または, $s_1 \overset{r}{\leftarrow}^{x}$ として逆の操作を行う. 定義は adjunction left の場合とほぼ同様なので省略する.
- finish: shift や adjunction をこれ以上適用できない場合,

$$\frac{[2n - 1, 0, n + 1, s_0] : \pi}{[2n, 0, n + 1, s_0] : \pi}$$

として, 終了状態を導く.

提案法による解析例を図 3 に示す.

3 素性

紙面の都合上, 用いた素性は概略のみを示す. 基本的な素性は文献 [11] で定義されている句構造解析のための素性を参考にした. さらに, 文献 [1] の素性 [4] を参考に, 基本木に関する素性を定義した. ここで注意したい点は, sister および regualr adjunction の操作時には spine 中のどの位置に接合するかによって構文解析に大きな影響を与える. そのため, 接合する側の基本木の根の非終端記号, 接合される側の基本木の非終端記号とその子の非終端記号の 3 つ組に関する素性セットを特別に “grammatical features” と呼ぶ. 実験では, “grammatical features” を加えた場合と加えなかった場合の精度への影響について議論する. さらに, スタック中の基本木が regular adjunction で作られたかどうかを示す 2 値素性 (“regular features” と呼ぶ) を追加し, この素性の精度への影響についても検証する.

[4] 文献 [1] では, sibling や grand-parent といった高次の単語間の係り受け関係の素性を利用しているが, 本研究ではこれらの素性は利用していない.

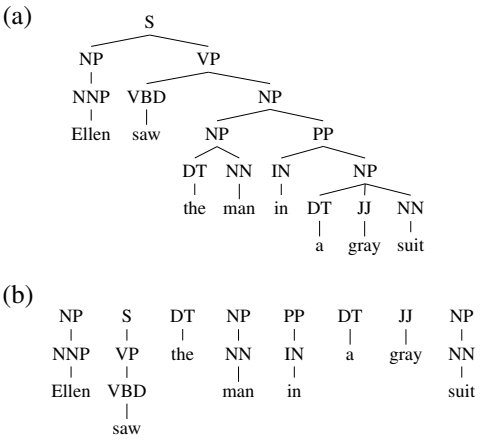

図 1: (a) 構文木の例; (b) 基本木の例.

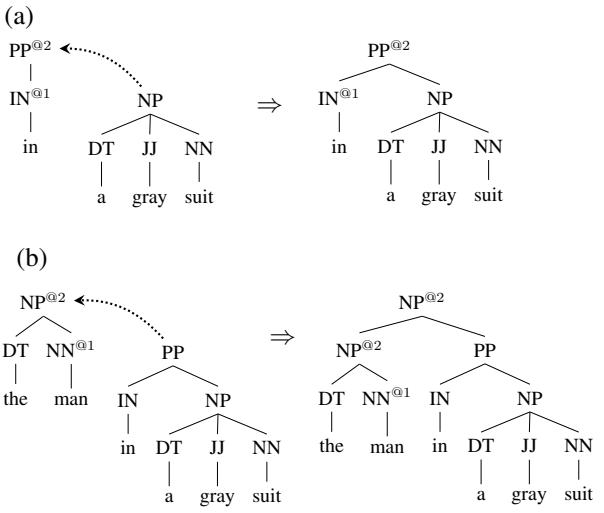

図 2: 接合操作の例. (a) sister adjunction (b) regular adjunction.

4 実験および考察

4.1 設定

提案法の有効性を検証するために, 構文解析で標準的に用いられる英語の WSJ Penn Treebank を用いた. 先攻研究と同じデータ分割方法に従い, 学習データにセ

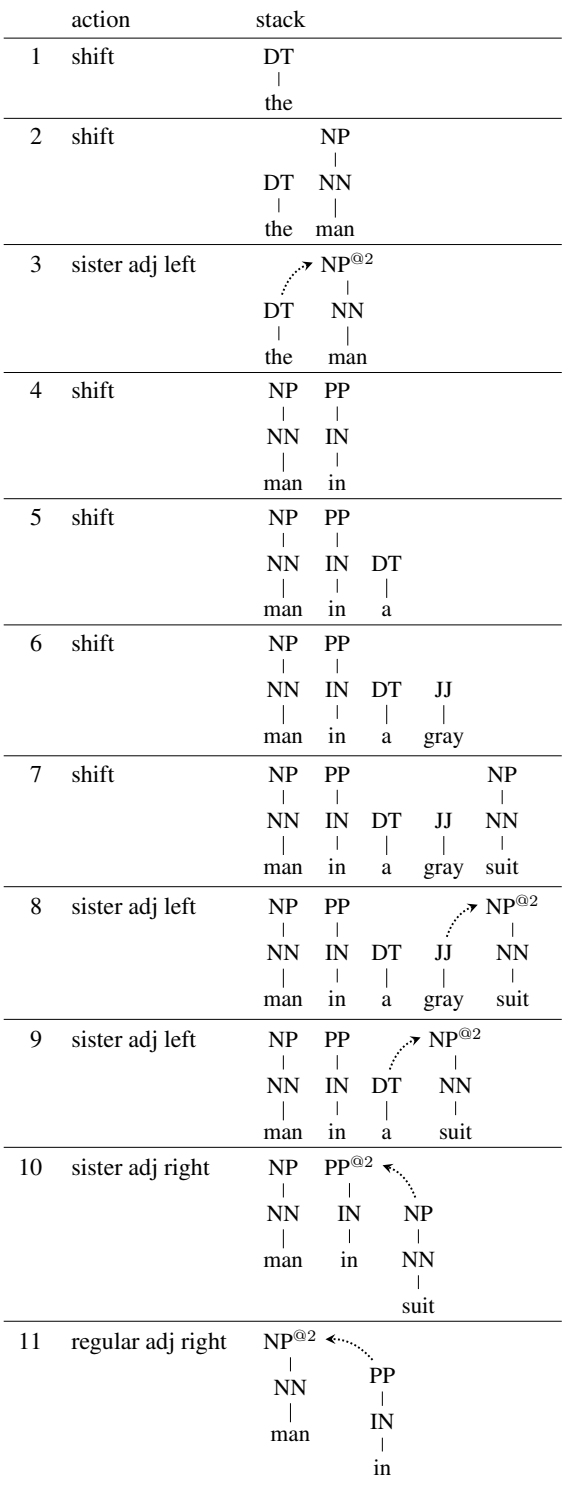

図 3: shift-reduce 法による単語列 *the man in a gray suit* の解析例.

クション 2-21, 開発データにセクション 22, 従来法との比較用にセクション 23 を用いた. TAG の基本木および導出は文献 [1] の方法に従い, 学習データから獲得した. 開発データおよびテストデータは Stanford POS Tagger[5] を用いて品詞付与を行った. 評価には EVALB[6] を用いた. なお, 学習時, テスト時ともにビーム探索のビーム幅を 16 に設定した. モデルの学習には violation-fixing perceptron, パラメータ更新の方法は max-violation を用いた [5]. 最適な反復回数は開発データでの解析精度により決定した.

[5] http://nlp.stanford.edu/software/tagger.shtml

	適合率	再現率	F_1
baseline	90.66	90.45	90.56
+ grammatical features	90.77	90.66	90.72
+ regular features	90.80	90.64	90.72

表 1: 異なる素性セットごとの開発データの解析精度の比較. grammatical features, regular features を逐次的に加えた場合の精度を表している.

	beam	F_1	tokens/seconds
w/o DP	16	87.22	1320
	32	87.29	662
	48	87.53	464
	128	87.66	185
w/ DP	2	89.06	4306
	4	90.15	2285
	8	90.63	1325
	16	90.72	752
w/ DP + supertagging	2	89.07	6989
	4	90.12	5181
	8	90.40	3896
	16	90.57	2183

表 2: ビーム幅と解析精度および解析速度の関係.

4.2 素性の影響

各素性セットごとの開発データの解析精度を表 1 に示す. 表より, “grammatical features” を加えることで “baseline” に対して精度が向上していることが分かる. “regular features” を加えると再現率が低下するものの, 適合率が向上していることが分かる.

4.3 状態の結合の影響

解析時に同じステップでビームに含まれている等価な状態 [7] を結合する場合 (“w/ DP”) と結合しない場合 (“w/o DP”) で別々にモデルの学習を行い, 開発データの精度への影響を調査した.

表 2 に, 開発データに対して, ビーム幅を変えた時の解析精度と解析速度の関係を示す. 状態を結合しない場合は, 状態を結合する場合に比べて, 約 3 から 5 近く精度が低下していることが分かる. 状態を結合しない場合, ビーム幅を上げることで解析精度が向上してい

[6] http://nlp.cs.nyu.edu/evalb/

[7] ここでいう状態の等価性は文献 [6] に基づいている.

ることから, ビームに同じような解析結果が含まれていて, 状態の結合が精度に大きく寄与していることが分かる. また, 解析速度についても注目してみると, 解析時に等価な状態を結合することによって, 精度が大きく向上するだけでなく, 精度を大きく犠牲にしないまま, 高速に解析できていることが分かる.

4.4 Supertaggingによる高速化

提案法は shift 時に単語の品詞に対して割り当て可能な基本木をすべて考えるため, 基本木の候補数が解析速度に大きく影響する. そこで, 構文解析の前処理として, supertagging を行い, あらかじめ各品詞に対して K-best の基本木を割り当ててから, 構文解析を行うことで解析速度の向上が可能か検証した. supertagging の方法は文献 [5] をベースとしたビーム探索[8] を用い, モデルの学習には構文解析と同じ方法を用いた. 素性として, 単語 unigram, 品詞 unigram, 品詞 bigram, 基本木 unigram および基本木 bigram を用いた.

テストデータにおける supertagging の精度は 94.43 であった. 表 2 に supertagging を行ってから構文解析を行った場合の解析精度と解析速度を示す. なお, 解析速度は, supertagging の速度と構文解析の速度の両方を合わせた速度である. 表より, supertagging を行わない場合 (“w/ DP”) と比べて, 解析精度は 0.03 から 0.15 低下が見られるが, 解析速度は 1.6 から 2.9 倍の高速化が達成できていることが分かる.

4.5 従来法との比較

表 3 にテストデータにおける従来法との精度の比較を示す. 提案法は shift-reduce 法に基づく方法として最も高い精度として報告されている文献 [11] と同等の精度となっていることが分かる. また, 提案法のビーム幅を 40 としてテストデータを解析した場合でも, 文献 [1] の方法と比べて F 値が 0.6 低いことが分かる. これは文献 [1] の方法と比べて, 構文解析アルゴリズムとそれに伴う素性が異なるためであると考えられる. 実際, 文献 [1] で用いられている sibling 素性を入れて実験を行ってみたが, 精度向上は見られなかった. 文献 [1] では, さらに grand-parent の素性も使っているが, 動的計画法に基づく shift-reduce 法の場合, grand-parent を入れることは技術的に難しい. 素性選択による更なる精度向上については今後の課題である.

[8] ビーム幅は 6 に設定した.

	適合率	再現率	F_1
†Sagae & Lavie 06 [9]	88.1	87.8	87.9
Petrov & Klein 07 [8]	90.2	89.9	90.1
† **提案手法** (b=16)	**90.6**	**90.2**	**90.4**
† **提案手法** (b=16) + supertagging	**90.5**	**90.2**	**90.3**
†Zhu et al. 13 [11]	90.7	90.2	90.4
† **提案手法** (b=40)	**90.6**	**90.3**	**90.5**
Carreras et al. 08 [1]	91.4	90.7	91.1
Charniak & Johnson 05 [2]	–	–	91.4
Huang 08 [4]	–	–	91.7

表 3: テストデータにおける従来法との精度比較. †shift-reduce 法.

5 おわりに

本稿では, 文献 [1] で提案された spinal TAG のための動的計画法に基づく shift-reduce 法を提案した. 提案法は, 英語の構文解析タスクにおいて, shift-reduce 法に基づく構文解析法の中で, 最も高い精度として報告されている構文解析法と同程度の精度を達成できることが分かった. また, 解析時に等価な状態の結合をすることによって, 精度が大幅に向上するだけでなく, より小さいビーム幅で高速に解析できることが分かった. さらに, 構文解析の前処理として supertagging を行うことで, 解析精度を大幅に犠牲にすることなく, 解析速度の高速化を達成できることが分かった.

参考文献

[1] Xavier Carreras, Michael Collins, and Terry Koo. TAG, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proc. of CoNLL*, pp. 9–16, 2008.

[2] Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proc. of ACL*, pp. 173–180, 2005.

[3] Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proc. of ACL*, pp. 16–23, 1997.

[4] Liang Huang. Forest reranking: Discriminative parsing with non-local features. In *Proc. of ACL-HLT*, pp. 586–594, 2008.

[5] Liang Huang, Suphan Fayong, and Yang Guo. Structured perceptron with inexact search. In *Proc. of NAACL*, pp. 142–151, 2012.

[6] Liang Huang and Kenji Sagae. Dynamic programming for linear-time incremental parsing. In *Proc. of ACL*, pp. 1077–1086, 2010.

[7] Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars. *Handbook of Formal Languages*, Vol. 3, pp. 69–124, 1997.

[8] Slav Petrov and Dan Klein. Improved inference for unlexicalized parsing. In *Proc. of HLT-NAACL*, pp. 404–411, 2007.

[9] Kenji Sagae and Alon Lavie. A best-first probabilistic shift-reduce parser. In *Proc. of ACL*, pp. 691–698, 2006.

[10] Libin Shen and Aravind Joshi. Incremental LTAG parsing. In *Proc. of HLT-EMNLP*, pp. 811–818, 2005.

[11] Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. Fast and accurate shift-reduce constituent parsing. In *Proc. of ACL*, pp. 434–443, 2013.