Vol.2012-NL-206 No.4
Vol.2012-SLP-91 No.4
2012/5/10

Split Head Automata による依存構造解析

林　克彦[†1]　渡辺太郎[†2]
浅原正幸[†1,*1]　松本裕治[†1]

データ駆動方式 (data-driven) の依存構造解析 (dependency parsing) は高速かつ高精度な構文解析が行えるため，近年盛んに研究が行われている．しかし，従来提案されてきた解析法は上昇型 (bottom-up) の手法に基づいたものであり，高速かつ高精度な解析が行える下降型 (top-down) の解析法は著者らの知る限り未だ提案されていない．本稿では下降型の依存構造解析アルゴリズムを考案し，その手法が解析速度，精度の面において，現在主流となっている解析法と遜色がないことを実験的に示す．

Parsing by Split Head Automata

Katsuhiko Hayashi ,[†1] Taro Watanabe ,[†2] Masayuki Asahara [†1,*1]
and Yuji Matsumoto[†1]

This paper presents a novel top-down data-driven dependency parsing algorithm. Our algorithm is a top-down transition-based parser combined with a **weighted prediction** from a statistical model inspired by the Earley prediction. Experiments on the English Penn Treebank data and the Chinese CoNLL-06 data show that the proposed algorithm achieves comparable results with other data-driven dependency parsing algorithms, while showing different characteristics from pure bottom-up dependency parsers.

1. はじめに

依存構造解析 (dependency parsing) ではデータ駆動方式 (data-driven) に基づいた高速かつ高精度なアルゴリズムが様々に提案されてきた[6),16),18),21)]．これらの手法は語彙化された文脈自由文法の特殊形に CKY 法や shift-reduce 法のような上昇型 (bottom-up) 構文解析アルゴリズムを応用したものと考えることができる．本稿では上昇型のアルゴリズムをベースにした解析法ではなく，下降型 (top-down) のアルゴリズムに基づいたデータ駆動方式の依存構造解析法を提案する．

提案手法は解析効率の問題を考慮して，shift-reduce 法と同様の決定的な遷移ベース (transition-based) のアルゴリズムを採用している．しかし，決定的な手法では探索エラーを頻繁に起こすため，ビームサーチ法[22)]を導入することで探索エラーを軽減している．さらに，ビームサーチの効率化を計るために Huang と Sagae によって提案された動的計画法 (dynamic programming)[9)]も導入することで解析速度の高速化を行った．

提案法の特徴は主辞 (head) から従属辞 (dependent) への関係を文全体から予測的に深さ優先 (depth-first) で探索しながら構文解析する点にある．これは上昇型のアルゴリズム，特に局所的な情報を使って解析を行う shift-reduce 法では考慮することの難しかった文全体の大域的な情報を取り込むことが可能になるという利点がある．提案手法の評価には英語の Penn Treebank，中国語の CoNLL-06 のタスクセットを用いた．実験結果からは従来手法と比べ，解析速度や精度の面で遜色のない結果を得ることができた．また，解析結果を分析することで従来法とは異なる解析傾向を観察することができたので，それらについて事例を提示する．

2. 非交差依存構造の定義

ここでは依存構造グラフ (dependency graph) を次のように定義する．

定義 2.1 (依存構造グラフ) $\mathbf{n}_0$ をルート記号 $\$_0$ とする入力文 $W = \mathbf{n}_0 \ldots \mathbf{n}_n$ が与えられたとき，依存構造グラフは有向グラフ $G = (V_W, A_W)$ で定義される．ここで $V_W = \{0, 1, \ldots, n\}$ は W におけるノード (のインデックス) 集合であり，$A_W \subseteq V_W \times V_W$ はノードとノードを結ぶエッジの集合である．エッジの集合 A_W は x を親となるノード (主辞)，y を x の子供となるノード (従属辞) とするペア (x, y) の集合である．

†1 奈良先端科学技術大学院大学
Nara Institute of Science and Technology, Ikoma, Nara, Japan
†2 情報通信研究機構
National Institute of Information and Communications Technology, Sorakugun, Kyoto, Japan
*1 現在，国立国語研究所
Presently with National Institute for Japanese Language and Linguistics, Tachikawa, Tokyo, Japan

入力文: $W = \mathbf{n}_0 \ldots \mathbf{n}_n$

公理 (p_0): $0 : \langle 1, 0, n+1, \mathbf{n}_0 \rangle : \emptyset$

$$\text{pred}_{\frown}: \quad \frac{\overbrace{\ell : \langle i, h, j, s_d | ... | s_0 \rangle : \_}^{\text{state } p}}{\ell + 1 : \langle i, k, h, s_{d-1} | ... | s_0 | \mathbf{n}_k \rangle : \{p\}} \; \exists k : i \leq k < h$$

$$\text{pred}_{\frown}: \quad \frac{\overbrace{\ell : \langle i, h, j, s_d | ... | s_0 \rangle : \_}^{\text{state } p}}{\ell + 1 : \langle i, k, j, s_{d-1} | ... | s_0 | \mathbf{n}_k \rangle : \{p\}} \; \exists k : i \leq k < j \; \wedge \; h < i$$

$$\text{scan}: \quad \frac{\ell : \langle i, h, j, s_d | ... | s_0 \rangle : \pi}{\ell + 1 : \langle i + 1, h, j, s_d | ... | s_0 \rangle : \pi} \; i = h$$

$$\text{comp}: \quad \frac{\overbrace{\_ : \langle \_, h', j', s'_d | ... | s'_0 \rangle : \pi'}^{\text{state } q} \qquad \overbrace{\ell : \langle i, h, j, s_d | ... | s_0 \rangle : \pi}^{\text{state } p}}{\ell + 1 : \langle i, h', j', s'_d | ... | s'_1 | s'_0 \frown s_0 \rangle : \pi'} \; q \in \pi, \; h < i$$

最終状態: $3n : \langle n + 1, 0, n + 1, s_0 \rangle : \emptyset$

図 1 下降型依存構造解析法の演繹論理システム: _ は任意とする.

定義 2.2 (well-formedness) ある依存構造グラフ $G = (V_W, A_W)$ が **well-formed** とは次を満たすときである.

- $(x, 0) \in A$ であるようなノード x が存在しない.
- もし $(x, y) \in A_W$ であるとき, $(x', y) \in A$ and $x' \neq x$ となるようなノード x' が存在しない.
- $x_0 = x_l$ となるような A_W の部分集合 $\{(x_0, x_1), (x_1, x_2), \ldots, (x_{l-1}, x_l)\}$ が存在しない.

これら上記 3 つの条件をそれぞれ順に **root**, **single-head**, **acyclicity** と呼ぶ.

定義 2.3 (非交差) ある依存構造グラフ $G = (V_W, A_W)$ が次を満たすとき, **非交差** (projective) であると呼ぶ. エッジ $(x, y) \in A_W$ と $x < z < y$ または $y < z < x$ となるノード $z \in V_W$ において, 集合 $\{(x, x_1), (x_1, x_2), \ldots, (x_{l-1}, z)\} \in A_W$ が存在する.

この **well-formed** で**非交差**の依存構造グラフが連結 (connected) であるとき, 依存構造木, そうでないとき, 依存構造森と呼ぶ.

3. 下降型依存構造解析法

提案法はアクションに基づいて, ある状態から次の状態へと遷移しながら構文解析を行う遷移ベースの手法である. 提案法は 4 つの遷移アクション **predict**$_{\frown}$(pred$_{\frown}$), **predict**$_{\frown}$(pred$_{\frown}$), **scan**, **complete** (comp) を持ち, その動作はスタック (stack) データ構造で記述した Earley 法[5] と類似している. より形式的には状態 (state) を

$$\ell : \langle i, h, j, S \rangle : \pi$$

として定義する. ここで ℓ は遷移回数, S は窓幅 d のスタック $s_d|...|s_0$ であり, 慣習的に s_0 をスタック先頭の木とする. i は入力文を入れたキュー (queue) における先頭ノードのインデックス, h は s_0 のルートノードのインデックス, j は pred$_{\frown}$ の探索範囲の右端となるインデックス ($j - 1$ を含む) である. π はスタック S にインデックス h のノードを積む直前の状態 (predictor states) へのポインタの集合である. 決定的な解析を行う場合, π は初期状態を除いてシングルトンであるが, ビームサーチ法と Huang と Sagae の動的計画法を併用する場合において, 複数のポインタを持つことがある.

図 1 は提案法の演繹論理システム (deductive logic system) を表している. 公理 (axiom) は仮想ルート $\mathbf{n}_0$ のみをスタックに積んだ状態を表しており, 最終状態はスタックの先頭に入力文の全ノードから成る依存構造木のみが積まれた状態にある. 提案手法は入力文の長さ n に対して, 公理から始まって $3n$ 回の遷移で最終状態へと至る. $3n$ 回のうち predict, scan, complete は各 n 回ずつ起こる. 各アクションの前提条件 $i < h$, $i = h$, $h < i$ は同時に満たされることはないので, アクション pred$_{\frown}$, scan, pred$_{\frown}$(または comp) が衝突 (conflict) を起こすことはない. しかし, pred$_{\frown}$ と comp は同じ前提条件 $h < i$ を持つため衝突を起こすことがある. 図 2 は具体的な解析の例である.

pred$_{\frown}$ における $i < h$ という前提条件はスタックの先頭にある木のルートノードのインデックス h より小さい位置にある入力文中のノードが未処理であることを意味している. pred$_{\frown}$ が起こると, i から $h - 1$ の間にあるノードを 1 つ選んで, スタックの先頭に積む操作が行われる. _ は任意とする. pred$_{\frown}$ における前提条件 $h < i$ は入力文の 1 から h までのノードが predict されて, 下で示す scan 操作がすでに終わっていることを意味している. pred$_{\frown}$ が起こると, h よりも右側にある i から j の間のノードを 1 つ選んで, スタックの先頭に積む操作を行う. scan はスタック先頭の木のルートノードのインデックス h と入力文を入れたキューの先頭のインデックス i が一致したとき, キューの先頭のノードを削除する操作を行っている. comp は現在の状態 p のスタック先頭における木のルートノードがこれ以上子供を持たないとき, スタックの 2 番目に積まれている木のルートノード (そのインデッ

遷移回数	状態名	スタック	キュー	アクション	状態の情報
0	p_0	$\$_0$	$\mathrm{I}_1\ \mathrm{saw}_2\ \mathrm{a}_3\ \mathrm{girl}_4$	–	$\langle 1,0,5\rangle : \emptyset$
1	p_1	$\$_0 \mid \mathrm{saw}_2$	$\mathrm{I}_1\ \mathrm{saw}_2\ \mathrm{a}_3\ \mathrm{girl}_4$	$\mathrm{pred}_\frown$	$\langle 1,2,5\rangle : \{p_0\}$
2	p_2	$\mathrm{saw}_2 \mid \mathrm{I}_1$	$\mathrm{I}_1\ \mathrm{saw}_2\ \mathrm{a}_3\ \mathrm{girl}_4$	$\mathrm{pred}_\frown$	$\langle 1,1,2\rangle : \{p_1\}$
3	p_3	$\mathrm{saw}_2 \mid \mathrm{I}_1$	$\mathrm{saw}_2\ \mathrm{a}_3\ \mathrm{girl}_4$	scan	$\langle 2,1,2\rangle : \{p_1\}$
4	p_4	$\$_0 \mid \mathrm{I}_1 \frown \mathrm{saw}_2$	$\mathrm{saw}_2\ \mathrm{a}_3\ \mathrm{girl}_4$	comp	$\langle 2,2,5\rangle : \{p_0\}$
5	p_5	$\$_0 \mid \mathrm{I}_1 \frown \mathrm{saw}_2$	$\mathrm{a}_3\ \mathrm{girl}_4$	scan	$\langle 3,2,5\rangle : \{p_0\}$
6	p_6	$\mathrm{I}_1 \frown \mathrm{saw}_2 \mid \mathrm{girl}_4$	$\mathrm{a}_3\ \mathrm{girl}_4$	$\mathrm{pred}_\frown$	$\langle 3,4,5\rangle : \{p_5\}$
7	p_7	$\mathrm{girl}_4 \mid \mathrm{a}_3$	$\mathrm{a}_3\ \mathrm{girl}_4$	$\mathrm{pred}_\frown$	$\langle 3,3,4\rangle : \{p_6\}$
8	p_8	$\mathrm{girl}_4 \mid \mathrm{a}_3$	girl_4	scan	$\langle 4,3,4\rangle : \{p_6\}$
9	p_9	$\mathrm{I}_1 \frown \mathrm{saw}_2 \mid \mathrm{a}_3 \frown \mathrm{girl}_4$	girl_4	comp	$\langle 4,4,5\rangle : \{p_5\}$
10	p_{10}	$\mathrm{I}_1 \frown \mathrm{saw}_2 \mid \mathrm{a}_3 \frown \mathrm{girl}_4$		scan	$\langle 5,4,5\rangle : \{p_5\}$
11	p_{11}	$\$_0 \mid \mathrm{I}_1 \frown \mathrm{saw}_2 \frown \mathrm{girl}_4$		comp	$\langle 5,2,5\rangle : \{p_0\}$
12	p_{12}	$\$_0 \frown \mathrm{saw}_2$		comp	$\langle 5,0,5\rangle : \emptyset$

図 2 例文“I saw a girl”に対する決定的な下降型依存構造解析の過程: ここでは慣習に従い，スタックの先頭は右端，キューの先頭は左端としている．また，この例では窓幅 d を 1 に設定しているため，s_0 と s_1 のみを各状態に示しており，さらに各スタック要素における木の子孫は深さ 1 までしか表記していない．

クスは h') との間に明示的なエッジ (h', h) を作る操作を行う．この操作では状態 p におけるスタックの要素 s_d よりも 1 つ深い場所にある要素や h' を知る必要があるので，predictor state となる状態 $q(\in \pi)$ を利用することでその情報を得ている．

4. Split-Head 文脈自由文法との関係

データ駆動方式の依存構造解析は明示的な文法規則を持たないが，2 分木化 (binarized)・語彙化 (lexicalized) された文脈自由文法に対する構文解析法と見なすこともできる．形式的な定義は省略するが，次のような 2 分木化・語彙化文脈自由文法を考える．

$$\begin{aligned}
\text{reduce}_\frown&: \quad \mathrm{S}[\$] \rightarrow \mathrm{X}[\mathbf{w1}]\\
\text{shift}&: \quad \mathrm{X}[\mathbf{w1}] \rightarrow \mathbf{w1}\\
\text{reduce}_\frown&: \quad \mathrm{X}[\mathbf{w1}] \rightarrow \mathrm{X}[\mathbf{w2}]\ \mathrm{X}[\mathbf{w1}]\\
\text{reduce}_\frown&: \quad \mathrm{X}[\mathbf{w1}] \rightarrow \mathrm{X}[\mathbf{w1}]\ \mathrm{X}[\mathbf{w2}]
\end{aligned}$$

ここで S は初期記号，X は非終端記号，**w1** と **w2** は終端記号であり，ここでは単語とする．また，上のように shift-reduce 法による依存構造解析[18] をこの文法に対応づけることができる．しかし，この文法はある依存構造木を作る導出 (derivation) が 1 つ以上考えられるという曖昧さ (spurious ambiguity) ⋆1 を持つ[10]．

この曖昧さの問題を解消するため，McAllester や Johnson は split-head 文脈自由文法 (split-head Context-Free Grammar) を提案している[10),14]．以下では McAllester の文法を示す．

$$\begin{aligned}
&\mathrm{S}[\$] \rightarrow \mathrm{X}[\mathbf{w1}]\\
&\mathrm{X}[\mathbf{w1}] \rightarrow \mathrm{L}[\mathbf{w1}]\ \mathbf{w1}\ \mathrm{R}[\mathbf{w1}]\\
&\mathrm{L}[\mathbf{w1}] \rightarrow \mathrm{X}[\mathbf{w2}]\ \mathrm{L}[\mathbf{w1}]\\
&\mathrm{L}[\mathbf{w1}] \rightarrow \epsilon\\
&\mathrm{R}[\mathbf{w1}] \rightarrow \mathrm{R}[\mathbf{w1}]\ \mathrm{X}[\mathbf{w2}]\\
&\mathrm{R}[\mathbf{w1}] \rightarrow \epsilon
\end{aligned}$$

ここで L と R は非終端記号である．提案法はこの文法と対応がとれる．例えば，図 2 の遷移回数 0 から 1 への遷移における $\mathrm{pred}_\frown$ では，**w1** が saw であり，1 つ目の規則から進んで，2 つ目の規則の L[**w1**] の手前まで予測を行う．次に 1 から 2 への遷移における $\mathrm{pred}_\frown$ では 3 つ目の規則の X[**w2**](**w2** =I) から再帰的に 1 つ目の規則へと行き，L[**w1**](**w1** =I) を 4 つ目の ϵ 規則で暗黙的に消化し⋆2，**w1** の手前まで進む．以降，2 から 3 の遷移では scan 操作によって **w1**(=I) を通過し，3 から 4 への comp 操作では R[**w1**](**w1** =I) を 6 つ目の ϵ 規則で消化して，3 つ目の規則へと戻り，その L[**w1**](**w1** =saw) を通過して，2 つ目の規則の **w1** の前まで進む．以下，同様の作業で上記した文法に提案法の解析過程を対応づけることができる．

Eisner と Satta は Alshawi によって提案された head automaton[2] が split-head 文脈自由文法へと等価変換できることを示しており[7]，この意味において，提案手法は head automaton の認識機械を構文解析として解釈したものであると言える．また，Kay によって提案された Head-corner 構文解析法[3),11] は split-head 文脈自由文法に Earley 法を適用した形に類似しており，提案法と最も近い性質を持った解析法であると言える．

5. 重み付き構文解析

5.1 状態遷移のための統計モデル

遷移ベースの依存構造解析では状態のスタックとキューから得られる情報を使って，次にとるべきアクションに重みを与えるモデルを設計する[9),19),23]．提案法でも同様なモデルを設計する．状態 $\ell : \langle i, h, j, S\rangle : \pi$ に対して適用できるアクションを act とすると，状態遷移ベースのモデルコスト $c_{\mathrm{s},act}(i, h, j, S)$ を

$$c_{\mathrm{s},act}(i, h, j, S) = \theta_{\mathrm{s}} \cdot \mathbf{f}_{\mathrm{s},act}(i, h, j, S). \qquad (1)$$

⋆1 shift-reduce 法による依存構造解析では一般に incremental な解析[18] が可能になるような木を標準形としている．

⋆2 ϵ 規則は対応するノードがない場合や子供をそれ以上とらない場合，アクション遷移を消費せず，暗黙的に使われるようにする．

Vol.2012-NL-206 No.4
Vol.2012-SLP-91 No.4
2012/5/10

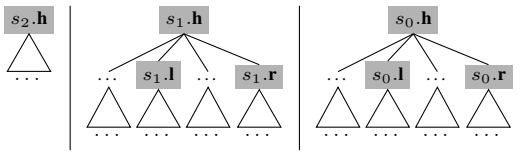

図 3 状態遷移ベースモデルの素性に用いるスタック S 上の情報: x.**h**，x.**l**，x.**r** は各々スタック要素 x のルートノード，左側の子供ノード，右側の子供ノードを表している．

Algorithm 1 Top-down parsing with Beam Search

```
1: input W = n_0, ..., n_n
2: start ← ⟨1, 0, n + 1, n_0⟩
3: buf[0] ← {start}
4: for ℓ ← 1 ... 3n do
5:   hypo ← {}
6:   for each state in buf[ℓ − 1] do
7:     for act ←applicableAct(state) do
8:       newstates ←actor(act, state)
9:       addAll newstates to hypo
10:    add top b states to buf[ℓ] from hypo
11: return best candidate from buf[3n]
```

として定義し，θ_{s} は状態遷移ベースの対数線形モデルに対する重みであり，$\mathbf{f}_{\mathrm{s}}$ は素性関数である．対数線形モデルのための素性テンプレートは[9] と同じものを実験では用いた． 図 3 に示すようにスタックの深さ d は 2 までで，s_0 と s_1 はルートの左右の子供ノードの情報を 1 つずつ素性に考慮している．ただし，shift-reduce 法の場合とは異なり，スタック要素 s_0 と s_1 の左側の子供ノード $s0$.**l**，$s1$.**l** は現在最も内側にあるものを使用している．これは提案手法では shift-reduce 法や CKY 法と異なり，左側の子供の構築が外側から内側に向かって行われるためである．

Algorithm1 はビームサーチ法を使った提案法の疑似コードである．ビームサーチ法では遷移回数 ℓ の状態でモデルコストが良い上位 b 個 (**ビーム幅**) の状態のみを残して，残りは枝刈りする (line10)．上位 b 個の状態はバッファ$buf[\ell]$ に残され，次の遷移候補となる．Huang と Sagae が提案した動的計画法では疑似コードの line10 において，次の条件を満たす状態を等価と見なして結合する．同じ遷移回数 ℓ を持つ 2 つの状態 $\langle i, h, j, S\rangle$ と $\langle i', h', j', S'\rangle$ が等価であるとは

$$\mathbf{f}_{\mathrm{s},act}(i, h, j, S) = \mathbf{f}_{\mathrm{s},act}(i', h', j', S'). \quad (2)$$

を満たす．これは状態遷移ベースのモデルコスト計算に使われる素性情報が同じになる状態を結合することになるため，モデル計算のオーバーヘッドを削減することができる．2 つの状態が結合されたとき，より良いコストを持つ状態のポインタ集合 π にもう一方の状態のポインタ集合 π を結合することで，過去の状態への参照を残すことができる．

5.2 重み付き予測モデル

predict の際に子供となる可能性の高いノードを効率良く選択するために，ここでは重み付きの予測モデルというものを提案する．本稿ではその重み付き予測モデルとして，グラフベースの依存構造解析[16] で用いられているモデルを採用した．グラフベースの依存構造解析モデルはノード間のエッジに重みを与えることができるため，親ノードと子ノードの関係を直接的にモデル化することができる．

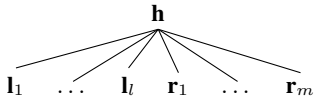

図 4 木構造の例: **h**，**l**_，**r**_ は各々主辞ノード，左の子供ノード，右の子供ノードを表している．

1 次のグラフベースモデルは子供ノード **c** と親ノード **h** の関係をモデル化し，そのモデルコスト $c_{\mathrm{p}}(\mathbf{h}, \mathbf{c})$ は次のように定義される．

$$c_{\mathrm{p}}(\mathbf{h}, \mathbf{c}) = \theta_{\mathrm{p}} \cdot \mathbf{f}_{\mathrm{p}}(\mathbf{h}, \mathbf{c}). \quad (3)$$

ここで θ_{p} は予測モデルの重みベクトル，$\mathbf{f}_{\mathrm{p}}$ はその素性関数である．本稿の実験では予測モデルとして 1 次のグラフベースモデルではなく，2 次のグラフベースモデルを採用している．2 次のグラフベースモデルは **c**，**h** と兄弟ノード **sib** の関係をモデル化し，そのモデルコスト $c_{\mathrm{p}}(\mathbf{h}, \mathbf{sib}, \mathbf{c})$ は次のように定義される．

$$c_{\mathrm{p}}(\mathbf{h}, \mathbf{sib}, \mathbf{c}) = \theta_{\mathrm{p}} \cdot \mathbf{f}_{\mathrm{p}}(\mathbf{h}, \mathbf{sib}, \mathbf{c}). \quad (4)$$

これら 1 次と兄弟ノードを用いた 2 次のグラフベースモデルは[16] の定義とほぼ同じである．しかし，兄弟ノードを用いた 2 次のグラフベースモデルの計算過程は[16] とは異なる． 例として図 4 で示すような木構造に対する 2 次モデルの計算方法を次のように定義する．

$$c_{\mathrm{p}}(\mathbf{h}, -, \mathbf{l}_1) + \sum_{y=1}^{l-1} c_{\mathrm{p}}(\mathbf{h}, \mathbf{l}_y, \mathbf{l}_{y+1}) + c_{\mathrm{p}}(\mathbf{h}, -, \mathbf{r}_1) + \sum_{y=1}^{m-1} c_{\mathrm{p}}(\mathbf{h}, \mathbf{r}_y, \mathbf{r}_{y+1}).$$

これは左側の子供に関するコスト計算が左から右に向かって行われている点で[16] の定義とは異なる．これは提案法では左側の子供が外側から内側に向かって生成されていくためである．このモデルは状態遷移モデルで用いられるスタックの情報のみで逐次計算できる．また，スタック情報は変わらないため，式 (2) で記述した状態の結合条件は変わらない．

ビームサーチ法によって節 3 で示した predict の推論規則における $\exists$ を $\forall$ に置き換えることができる．これより 1 つの状態で複数の子供ノードを予測して，複数の状態へと遷移することが可能となる．しかし，ここでビームサーチ法で用いるバッファが predict で作り出された状態のみで満たされ，他の状態候補が全く考慮できなくなるという問題が生じる．この問題を防ぐため，本稿では 1 つの状態から predict できる数に制限を設け，その制限する数を**予測幅** (prediction size) として定義した．実装では Algorithm1 の line10 で 1 つの状態から predict で作り出された候補を予測幅の

上限までに制限して確保している．

5.3 重み付き演繹論理システム

前向きコスト (forward cost) と**内側コスト** (inside cost) と呼ばれるコスト[9),20)] を導入して，先に定義した重みなしの論理システムを重み付き論理システムへと拡張する．前向きコストとは公理から現在の状態に至るまでのコストの総計である．内側コストとはスタック S の先頭要素 s_0 に関するコストの総計である．

ここでは状態遷移モデルと予測モデルの和を用いて，前向きコストと内側コストを定義する．その結合モデルにおける前向きコストは

$$c^{\text{fw}} = c_{\text{s}}^{\text{fw}} + c_{\text{p}}^{\text{fw}} \tag{5}$$

として定義され，ここで c_{s}^{fw} は状態遷移モデルの前向きコスト，c_{p}^{fw} は予測モデルの前向きコストを表している．結合モデルの内側コストも同様にして

$$c^{\text{in}} = c_{\text{s}}^{\text{in}} + c_{\text{p}}^{\text{in}} \tag{6}$$

として定義し，ここで c_{s}^{in} は状態遷移モデルの内側コスト，c_{p}^{in} は予測モデルの内側コストを表している．

各状態には次の 4 つ組のコストを付与する．

$$(c_{\text{s}}^{\text{fw}}, c_{\text{s}}^{\text{in}}, c_{\text{p}}^{\text{fw}}, c_{\text{p}}^{\text{in}}).$$

Stolcke はこの内側コストと前向きコストを確率的 Earley 法で効率良く計算するための手法を提案している[20)]．本稿では Stolcke の手法とほぼ同様の計算方法をとる．表現を簡潔にするため，以下では予測モデルに 1 次のグラフベースモデルを使って説明を行う．pred$_\frown$ と pred$_\curvearrowright$ の場合，コスト計算は

$$\text{pred}_\frown: \quad \frac{(c_{\text{s}}^{\text{fw}}, \_, c_{\text{p}}^{\text{fw}}, \_)}{(c_{\text{s}}^{\text{fw}} + \lambda, 0, c_{\text{p}}^{\text{fw}} + c_{\text{p}}(s_0.\mathbf{h}, \mathbf{n}_k), 0)}$$

$$\text{pred}_\curvearrowright: \quad \frac{(c_{\text{s}}^{\text{fw}}, \_, c_{\text{p}}^{\text{fw}}, \_)}{(c_{\text{s}}^{\text{fw}} + \rho, 0, c_{\text{p}}^{\text{fw}} + c_{\text{p}}(s_0.\mathbf{h}, \mathbf{n}_k), 0)}$$

となり，ここで

$$\lambda = \theta_{\text{s}} \cdot \mathbf{f}_{\text{s,pred}_\frown}(i, h, j, S) \tag{7}$$

$$\rho = \theta_{\text{s}} \cdot \mathbf{f}_{\text{s,pred}_\curvearrowright}(i, h, j, S) \tag{8}$$

と定義する．predict では各内側コストは 0 であり，各前向きコストにそれぞれのモデルのコストが加算される．scan の場合，コスト計算は

$$\frac{(c_{\text{s}}^{\text{fw}}, c_{\text{s}}^{\text{in}}, c_{\text{p}}^{\text{fw}}, c_{\text{p}}^{\text{in}})}{(c_{\text{s}}^{\text{fw}} + \xi, c_{\text{s}}^{\text{in}} + \xi, c_{\text{p}}^{\text{fw}}, c_{\text{p}}^{\text{in}})}$$

となり，ここで

$$\xi = \theta_{\text{s}} \cdot \mathbf{f}_{\text{s,scan}}(i, h, j, S). \tag{9}$$

と定義する．scan では遷移モデルのコストのみが前向きコストと内側コストに加算される．comp の場合，コスト計算は

$$\frac{(c'^{\text{fw}}_{\text{s}}, c'^{\text{in}}_{\text{s}}, c'^{\text{fw}}_{\text{p}}, c'^{\text{in}}_{\text{p}}) \qquad (c_{\text{s}}^{\text{fw}}, c_{\text{s}}^{\text{in}}, c_{\text{p}}^{\text{fw}}, c_{\text{p}}^{\text{in}})}{\begin{array}{l}(c'^{\text{fw}}_{\text{s}} + c_{\text{s}}^{\text{in}} + \mu, c'^{\text{in}}_{\text{s}} + c_{\text{s}}^{\text{in}} + \mu, \\ \quad c'^{\text{fw}}_{\text{p}} + c_{\text{p}}^{\text{in}} + c_{\text{p}}(s'_0.\mathbf{h}, s_0.\mathbf{h}), \\ \quad c'^{\text{in}}_{\text{p}} + c_{\text{p}}^{\text{in}} + c_{\text{p}}(s'_0.\mathbf{h}, s_0.\mathbf{h}))\end{array}}$$

となり，ここで

$$\mu = \theta_{\text{s}} \cdot \mathbf{f}_{\text{s,comp}}(i, h, j, S) + \theta_{\text{s}} \cdot \mathbf{f}_{\text{s,pred}_\_}(\_, h', j', S'). \tag{10}$$

と定義する．pred$__$ は pred$_\frown$ か pred$_\curvearrowright$ のどちらかをとる．comp では現在スタックの先頭にある木のルートノードを予測した際のコストを μ の右辺第 2 項と $c_{\text{p}}(s'_0.\mathbf{h}, s_0.\mathbf{h})$ によって，状態遷移コストと予測コストに繰り越して加算している．

ここで同じ遷移回数にある 2 つの状態 p と p' があり，それぞれ結合モデルの前向きコスト，内側コストの 2 つ組 $(c^{\text{fw}}, c^{\text{in}})$ と $(c'^{\text{fw}}, c'^{\text{in}})$ を持つとする．ビームサーチ法におけるこれら 2 つの状態の線形順序は

$$p \succ p' \text{ iff } c^{\text{fw}} < c'^{\text{fw}} \text{ or } c^{\text{fw}} = c'^{\text{fw}} \wedge c^{\text{in}} < c'^{\text{in}}. \tag{11}$$

によって決まる．ここでは内側コストよりも前向きコストの方に優先度をもたせているが，これはより長い系列のコストを表す前向きコストの方が適切に仮説の善し悪しを判断できるからである[17)]．

5.4 FIRST 関数による先読み

下降型構文解析ではバックトラックを軽減するために，FIRST(·) 関数を予め用意しておくことがある[1)]．本稿では考案した下降型依存構造解析法に対して，この FIRST(·) 関数を以下のように定義する．

$$\text{FIRST(t')} = \{\mathbf{ld}.\text{t} | \mathbf{ld} \in \text{lmdescendant(Tree, t')}, \text{Tree} \in \text{Corpus}\}$$

ここで t' は品詞タグ，Tree は構文解析済みコーパスに含まれる正解の依存構造木，lmdescendant(Tree, t') 関数は引数に Tree と t' をとって，Tree 上の t' を持つ各ノードの最も左側 (最も小さいインデックス) にある子孫ノード **ld** の品詞タグ **ld.**t の集合を返す．

提案法はバックトラックを行わないが，predict で適切な子供を選択するためのフィルタリングとして FIRST(·) 関数を使うことができる．pred$_\frown$ の場合，

$$\frac{\overbrace{\ell : \langle i, h, j, s_d | ... | s_0 \rangle : \_}^{\text{state } p}}{\ell + 1 : \langle i, k, h, s_{d-1} | ... | s_0 | \mathbf{n}_k \rangle : \{p\}} \quad \forall k : i \leq k < h \ \wedge \ \mathbf{n}_i.\text{t} \in \text{FIRST}(\mathbf{n}_k.\text{t})$$

として FIRST(·) 関数をフィルタリングに使うことができる．ここで $\mathbf{n}_i$.t はキューの先頭ノードの品詞タグ，$\mathbf{n}_k$.t は予測されたノードの品詞タグを表す．pred$_\curvearrowright$ の場合も全く同様の使い方となる[⋆1]．ただし，FIRST(·) 関数によって候補となる単語が 1 つもなくなる場合，predict できる全ての単語からできる状態を全て候補としている．FIRST(·) 関数のアイディアは決定的な LL(1) 構文解析法で用いられるテクニックである[1)] が，ここでは不要なエッジを除去するために使われる arc filtering と同じ性質のものである．

6. 提案法の分析

6.1 時間計算量

提案法は predict，scan，comp の 3 種類のアクションを持つ．入力文長 n に対して，これらはそれぞれ n 回ずつ起こる．ただし，predict では各回ごとに子供ノードを選択する必要が生じる．これには次のような

⋆1 提案手法では pred$_\frown$，pred$_\curvearrowright$ どちらの場合でも，キューの先頭にあるノードは予測されたノードの最も左の子孫ノードとなる．

回数の計算を必要とする．

$$n + (n-1) + \cdots + 1 = \sum_{i}^{n} i = \frac{n(n+1)}{2}. \quad (12)$$

この n^2 は最も大きな計算項となるので，提案法の計算量は $O(n^2)$ となり，ビーム幅 b のビームサーチ法を併用すると $O(n^2 * b)$ の計算量となる．

また，1 次のグラフベースモデル上[6),15)] で提案法の全解探索の計算量を考える．まず，素性を取り出すために最小限必要な情報の集合である**カーネル素性集合** (kernel features set) を $\{i, h, j, s_1.\mathbf{h}, s_0.\mathbf{h}\}$ (窓幅 $d = 1$) として定義する．このとき，動的計画法を使った提案法の計算量は $O(n^7)$ となる．これは complete のステップにおいて，6 つの自由変数 _, h', j', i, h, j と 1 つの主辞に関する変数 $s'_1.\mathbf{h}$ を必要とするからである[⋆1]．Huang と Sagae は同様の分析により，動的計画法を使った shift-reduce 法の計算量が $O(n^7)$ であることを示した[9)]．全解探索の観点では Kuhlmann らによって $O(n^5)$ 遷移ベースの依存構造解析法が提案され[13)]，これはさらに *hook trick*[7)] で $O(n^4)$ とできる．

6.2 演繹論理システムの正当性

定理 6.1 (正当性) 提案法の演繹論理システムは **well-formed** で**非交差**の依存構造グラフ (森) に対して正当 (correct) である．

証明 6.1 (正当性の証明) 正当な依存構造解析論理システムは健全性 (soundness) と完全性 (completeness) を有する．健全性を示すには公理の状態 p_0 に対する依存構造グラフ $G_{p_0} = (V_W, \emptyset)$ が well-formed で非交差の依存構造グラフであり，全ての遷移がこの性質を保持することを示せばよい．

- **root**: インデックス 0 のノードは G_{p_0} のルートであり，このノードは p_0 のスタックの先頭に積まれているので，以後，それが親を持つことはない．
- **single-head**: G_{p_0} における全てのノード $x \in V_W$ は親を持たず，comp アクションによって辺が作られた後，その辺の修飾語はスタックにもキューにも存在しない．よって，新たな辺が作られるとき，その修飾語はまだ 1 つも親を持っていない．
- **acyclicity**: G_{p_0} は閉路を持たない．新たな辺 (x, y) を作ったとき，依存構造グラフに閉路 (cycle) ができるのはノード y からノード x への有向なパス (path) が存在するときであるが，その場合，x はすでにスタックにもキューにも存在しない．
- **非交差**: G_{p_0} は交差を持たない．ある辺 (x, y) の関係を $\text{pred}_\frown(x > y)$ か $\text{pred}_\frown(x < y)$ によって作ることでグラフに交差が生まれるのは，$y < z < x$ または $x < z < y$ のノード z に対して，$x \rightarrow^* z$ または $y \rightarrow^* z$ となるパスが作られないときである．$\text{pred}_\frown$ によって状態 $\langle x+1, x, j\rangle$ から状態 $\langle x+1, y, j\rangle$ へ遷移したとすると，$x < z < y$ にある z のうち少なくとも 1 つは y から辺が作られる．$\text{pred}_\frown$ によって遷移した先の状態を $\langle i, y, x\rangle$ とすると，$y < z < x$ にあるノード z は y が scan されるまで ($\langle y+1, y, x\rangle$ の状態になるまで) predict されないので，$y < z < x$ にある z のうち少なくとも 1 つは y または x から辺が作られる．これらの性質は辺 (x, z) または (y, z) にも適用されるので，辺 (x, y) ができたとき，$y < z < x$ または $x < z < y$ にある全てのノード z には $x \rightarrow^* z$ または $y \rightarrow^* z$ となるパスができる．

よって，提案法の演繹論理システムは健全である．

完全性を示すため，どんな入力文 W と依存構造森 $G_W = (V_W, A_W)$ に対しても，$G_{p_m} = G_W$ となるような状態列 $C_{0,m} = \{p_0, \ldots, p_m\}$ が存在することを入力文の長さ $|W|$ に対して帰納法で示す．

- $|W| = 1$ のとき，W に対する唯一の依存構造森は $G_W = (\{0\}, \emptyset)$ であり，$G_{p_0} = G_W$ である．
- $|W| \leq t$ $(t > 1)$ のとき，主張が成立するとして，$|W| = t + 1$ で $G_W = (V_W, A_W)(V_W = \{0, \ldots, t\})$ であると仮定する．その部分グラフ $G_{W'}$ を $(V_W - \{t\}, A^{-t})$ $(A^{-t} = A_W - \{(x, y)|x = t \vee y = t\})$ とする．ここで G_W が依存構造森である場合，$G_{W'}$ も依存構造森である．また，G_W のノード t に関与する箇所以外の構造を作る状態列が存在することは明らかである[⋆2]．$0 \leq x < i < t + 1$ となる x と i に対して，状態 $p_q = \langle i, x, t + 1\rangle$ とする．ここで G_W において x が t の親となる場合，$\text{pred}_\frown$ アクションを使い，状態 $p_{q+1} = \langle i, t, t + 1\rangle$ へと遷移する．さらに $i < z < t$ の間の少なくとも 1 つのノードには $\text{pred}_\frown$ で遷移し，そこから再び comp で t に戻るまでの状態列は明らかなので，この操作を子供を持つだけ繰り返すことで $i < z < t$ の構造が構築できる．次に t を scan し，comp で x へと戻ると，残りの遷移列は明らかである．よって，$G_{p_m} = G_W$ となる遷移列 $C_{0,m}$ は $G_{W'}$ の部分構造を構築する全ての状態列と t に関与するアクションを経て作ることができる．

以上より，提案法の演繹論理システムは健全かつ完全である．よって，その正当性が証明された．

7. 実　　験

実験は英語の Penn Treebank データと中国語の CoNLL−06 データを用いて行った．英語データは WSJ 部分の 02 – 21 を訓練データ，22 を開発データ，23 をテストデータとして用いた．句構造から依存構造への変換は Yamada と Matsumoto の変換ルール[21)] を使っ

⋆1 $s'_0.\mathbf{h}$, $s_1.\mathbf{h}$ and $s_0.\mathbf{h}$ は $s'_0.\mathbf{h} = s_1.\mathbf{h} = \mathbf{n}_{h'}$ かつ $s_0.\mathbf{h} = \mathbf{n}_h$ なので計算量に考慮する必要はない

⋆2 これらの状態列は $|W'|$ に対する定義であるが，t に関与しない箇所では $|W|$ による定義としても特に問題はない．

Vol.2012-NL-206 No.4
Vol.2012-SLP-91 No.4
2012/5/10

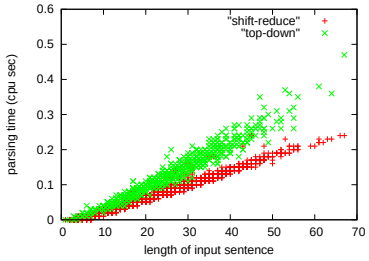

図 5 英語テストデータの文長ごとの解析速度: 提案法と shift-reduce 法ともにビーム幅を 8，提案法の予測幅を 5 とした．

表 2 英語テストデータに対する oracle 精度: 提案法 (ビーム幅 8, 予測幅 5) と shift-reduce 法 (ビーム幅 8) とした．

	ラベルなし精度	文正解率	ルート正解率
提案法	91.9	45.0	95.1
shift-reduce 法	92.3	43.5	96.0
oracle	94.2	51.4	97.8

表 3 中国語データ (CoNLL-06) での実験結果: 提案法と shift-reduce 法ともにビーム幅を 8，提案法の予測幅を 5 とした．

	ラベルなし精度	文正解率	ルート正解率
提案法	90.9	80.4	93.0
shift-reduce 法	90.8	77.6	93.5
oracle	93.8	84.0	95.6

た．中国語のデータに対しては，単語と fine-grained な品詞タグを素性に使った．また，比較実験のために，Huang と Sagae の shift-reduce 法の依存構造解析器[9] を実装して用意した．

shift-reduce 法と提案法のモデル学習には Collins と Roark の early update を使った平均化パーセプトロン[4] を用いた．shift-reduce 法と提案法の状態遷移モデルの素性テンプレートは Huang と Sagae と同じものを使用した．また，提案法の予測モデルには McDonald の 2 次のグラフベースモデル素性[16] を使用した．提案法の両モデルは同時学習している．

7.1 英語データに対する実験結果

開発データを使った予備実験から，訓練時における提案法の予測幅とビーム幅はそれぞれ 5 と 16 に設定した．パーセプトロンのイテレーションを 25 回行った後，開発データに対する提案法の解析精度 (ラベルなし精度) は 92.94 となった．テスト時には予測幅とビーム幅を 5 と 8 に設定した上で解析を行っている．一方，shift-reduce 法では訓練時のビーム幅を 16，開発データの解析時は 8 に設定して解析を行った結果，93.01 の解析精度が得られた．

次に，テストデータを用いた実験により解析精度を調べた．表 1 にはその結果と，現在，高い解析精度が報告されている様々な解析器との比較を示した．提案法は文正解率の観点で他の解析器に比べ，良い結果が得られ，ラベルなし精度でもほぼ同等の精度となった．文正解率が高い要因の 1 つとして，提案法では文全体を探索しながら解析を行っていることが考えられる．一方，提案法は shift-reduce 法と比較して，ルート正解率が低くなっている．この理由として，提案法は遷移回数 0 のときにルートとなるノードを選択する必要があるが，この時点では部分的な解析結果はまだなく，ルート選択に使える情報が少ないことが挙げられる．ルート選択の精度を向上させるような素性や仕組みを考えることは今後の重要な課題と言える．

図 5 はテストデータの文長に対する解析速度を示している．提案法は理論上の計算時間では shift-reduce 法よりも遅く，この結果はそれを反映したものとなっている．解析にかかる時間の多くは素性計算によるところが大きい[8] ため，提案法のモデルに対する適切なモデル設計や素性選択も今後の課題である．図 2 はテストデータに対する提案法の解析結果と shift-reduce 法の解析結果を比較して，精度の良い方を選んで，解析精度を出した結果である (oracle 精度)．oracle 精度が高いことは提案法と shift-reduce 法の解析結果が異なる傾向にあることを示唆している．

7.2 中国語データ (CoNLL-06) に対する実験結果

中国データでは shift-reduce 法とのみ比較を行った．各種パラメータ設定は英語データにおける実験を踏まえて，提案法と shift-reduce 法ともにビーム幅を 16，提案法の予測幅を 5 に設定して訓練を行った．表 3 はビーム幅を 8，予測幅を 5 に設定して，テストデータを解析した結果を示している．文正解率，ラベルなし精度，ルート正解率の傾向は英語における結果とほぼ同じである．ラベルなし精度 90.9 は CoNLL-06 の中国語データで，3 番目に高い数値である．

7.3 英語テストデータ解析結果の分析

表 2 で示した解析結果ではテストデータ 1439 文のうち，490 文で提案法が shift-reduce 法のラベルなし精度を上回った (shift-reduce 法: 487，同じ精度: 1439)．提案法がラベルなし精度で shift-reduce 法を上回った事例のうち，興味深いものを表 4 に提示した．これらの例では主動詞と主語の間にある副詞節や関係節のため，長距離の依存関係が生じている．提案法ではこのような現象を比較的上手く捉えられるのに対し，shift-reduce 法では局所的な探索を行うため，正しい解析を行うことが難しくなる．

参 考 文 献

1) A.V. Aho and J.D. Ullman. *The Theory of Parsing, Translation and Compiling*, volume 1: Parsing. Prentice-Hall, 1972.
2) H.Alshawi. Head automata for speech translation. In *Proc. the ICSLP*, 1996.
3) G.Bouma and G.V. Noorld. Head-driven parsing for lexicalist grammars: Experimental results. In *Proc. the EACL*, pages 71–80, 1993.
4) M.Collins and B.Roark. Incremental parsing with the perceptron algorithm. In *Proc. the 42nd ACL*,

表 1 英語データ (section23) に対する解析精度: 解析時間 (毎文秒)，ラベルなし精度，文正解率，ルート正解率を表示している．sh は shift-reduce 法，gh はグラフベース法，DP は動的計画法，FIRST は FIRST 関数を表している．∗ は著者らの実装によるものである．

	解析時間	ラベルなし精度	文正解率	ルート正解率
McDonald05，06 (gh 2nd)	0.15，–	90.9，91.5	37.5，42.1	–，–
Goldberg10[8]	–	89.7	37.5	91.5
Kitagawa10[12] (sh)	–	91.3	41.7	–
Zhang08 (sh，sh+gh ビーム幅 64)	–，–	91.4，92.1	41.8，45.4	–，–
Huang10 (sh+DP ビーム幅 8)	0.04	92.1	–	–
Huang10∗ (sh+DP ビーム幅 8，16，32)	0.07，0.13，0.26	92.3，92.27，92.26	43.5，43.7，43.8	96.0，96.0，96.1
提案法 ∗ (DP ビーム幅 8，16，32，予測幅 5)	0.10，0.19，0.33	91.7，92.3，92.5	45.0，45.7，**45.9**	94.5，95.7，96.2
提案法 ∗ (DP+FIRST ビーム幅 8，16，32，予測幅 5)	0.10，0.19，0.33	91.9，92.4，**92.6**	45.0，45.3，45.5	95.1，96.2，**96.6**

表 4 提案法が shift-reduce 法よりも良い解析であった事例: 四角部分は下線部分の主辞とする．

No.717	Little Lily , as Ms. Cunningham calls$_7$ herself in the book , really was$_{14}$ n't ordinary .																	
shift-reduce 法	2	7	2	2	6	4	14	7	7	11	9	7	14	0	14	14	14	
提案法	2	14	2	2	6	7	4	7	7	11	9	2	14	0	14	14	14	
正解	2	14	2	2	6	7	4	7	7	11	9	2	14	0	14	14	14	

No.1541	A provision that would have made . . . was$_{14}$ excised .									
shift-reduce 法	2	0	2	3	4	5	. . .	6	14	2
提案法	2	14	2	3	4	5	. . .	0	14	14
正解	2	14	2	3	4	5	. . .	0	14	14

2004.

5) J. Earley. An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery*, 13(2):94–102, 1970.

6) J.M. Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proc. the 16th COLING*, pages 340–345, 1996.

7) J. M. Eisner and G. Satta. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proc. the 37th ACL*, pages 457–464, 1999.

8) Y.Goldberg and M.Elhadad. An efficient algorithm for easy-first non-directional dependency parsing. In *Proc. the HLT-NAACL*, pages 742–750, 2010.

9) L.Huang and K.Sagae. Dynamic programming for linear-time incremental parsing. In *Proc. the 48th ACL*, pages 1077–1086, 2010.

10) M.Johnson. Transforming projective bilexical dependency grammars into efficiently-parsable cfgs with unfold-fold. In *Proc. the 45th ACL*, pages 168–175, 2007.

11) M.Kay. Head driven parsing. In *Proc. the IWPT*, 1989.

12) K. Kitagawa and K. Tanaka-Ishii. Tree-based deterministic dependency parsing — an application to nivre's method —. In *Proc. the 48th ACL 2010 Short Papers*, pages 189–193, July 2010.

13) M. Kuhlmann, C. Gómez-Rodríguez, and G. Satta. Dynamic programming algorithms for transition-based dependency parsers. In *Proc. the 49th ACL*, pages 673–682, 2011.

14) D.McAllester. A reformulation of eisner and satta's cubic time parser for split head automata grammars. 1999. http://ttic.uchicago.edu/dmcallester/.

15) R. McDonald, K. Crammer, and F. Pereira. Online large-margin training of dependency parsers. In *Proc. the 43rd ACL*, pages 91–98, 2005.

16) R.McDonald and F.Pereira. Online learning of approximate dependency parsing algorithms. In *Proc. EACL*, pages 81–88, 2006.

17) M.-J. Nederhof. Weighted deductive parsing and knuth's algorithm. *Computational Linguistics*, 29:135–143, 2003.

18) J. Nivre. Incrementality in deterministic dependency parsing. In *Proc. the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, 2004.

19) J.Nivre. *Inductive Dependency Parsing*. Springer, 2006.

20) A. Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201, 1995.

21) H. Yamada and Y. Matsumoto. Statistical dependency analysis with support vector machines. In *Proc. the IWPT*, pages 195–206, 2003.

22) Y.Zhang and S.Clark. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proc. EMNLP*, pages 562–571, 2008.

23) Y.Zhang and J.Nivre. Transition-based dependency parsing with rich non-local features. In *Proc. the 49th ACL*, pages 188–193, 2011.