日本語係り受け解析の線形時間アルゴリズム

颯々野

学^{†,††}

日本語係り受け解析を行なう新しいアルゴリズムを述べる。このアルゴリズムによれば、トップレベルの精度を落とすことなく線形時間で係り受け解析が行なえる。本論文では、アルゴリズムの形式的な記述を行ない、その時間計算量を理論的に議論する。加えて、その効率と精度を京大コーパス Version 2 を使って実験的にも評価する。改良された係り関係のモデルと提案手法を組み合わせると、京大コーパス Version 2 に対して従来手法よりもよい精度が得られた。

キーワード：日本語係り受け解析，構文解析，依存構造解析

A Linear-Time Algorithm for Dependency Analysis of Japanese

MANABU SASSANO^{†,††}

We present a novel algorithm for Japanese dependency analysis. The algorithm allows us to analyze dependency structures of a sentence in linear-time while keeping a state-of-the-art accuracy. In this paper, we show a formal description of the algorithm and discuss it theoretically with respect to time complexity. In addition, we evaluate its efficiency and performance empirically against the Kyoto University Corpus Version 2. The proposed algorithm with improved models for dependency yields the best accuracy in the previously published results on the Kyoto University Corpus Version 2.

Key Words: *dependency analysis, parsing*

1 はじめに

構文解析において、精度と同様、計算効率も、自然言語処理の重要な問題の一つである。構文解析の研究では、精度に議論の重点を置くことが多いが、効率についての研究もまた重要である。特に実用的な自然言語処理のアプリケーションにとっては、そうである。精度を落とすことなく効率を改善することは、とても大きな課題である。

本研究の目的は、日本語の係り受け解析 (依存構造解析) を行なう効率のよいアルゴリズムを提案し、その効率の良さを理論的、実験的の両面から示すことである。本論文では、日本語係り受け解析の線形時間アルゴリズムを示す。このアルゴリズムの形式的な記述を示し、その時間

[†] 株式会社富士通研究所, Fujitsu Laboratories Ltd.

^{††} 現在, ヤフー株式会社, Now at Yahoo Japan Corporation

計算量 (time complexity) を理論的に論じる．加えて，その効率と精度を京大コーパス Version 2 (Kurohashi and Nagao 1998) を使って実験的に評価する．

本論文の構成は以下の通りである．第 2 節では，日本語の構文的な特徴と典型的な日本語文の解析処理について述べる．第 3 節では，英語や日本語の依存構造解析の従来研究について簡単に述べる．その後，第 4 節で我々の提案手法を述べる．次に，第 5 節で，二つの文節の依存関係を推定するための改良したモデルを述べる．第 6 節では実験結果とその考察を記す．最後に，第 7 節で本論文での我々の貢献をまとめる．

2 日本語文の解析

本節では，日本語の構文的特徴と典型的な日本語文の解析の手順について整理する．

2.1 日本語の構文的特徴

日本語は基本的には SOV 言語である．語順は比較的自由である．英語では，文中の語の構文的機能は語順で表される．一方，日本語では，後置詞 (postpositions) によって表される．この点では，名詞の後に置かれる日本語の格助詞はドイツ語名詞の格変化と類似の役割を持っている．ドイツ語名詞は格変化することによって，文法的な格を表している．

文節の概念¹は上記の日本語の性質と親和性があり，日本語文を構文的に分析するときに使われてきた．文節は，1 個以上の内容語 (content words) とそれに続く 0 個以上の機能語 (function words) から構成される．文節をこのように定義することによって，ドイツ語のような屈折言語において文中の語の文法的役割を分析するときと似た手法を日本語文を分析するときにも使うことができる．

それゆえ，厳密なことを言えば，日本語の場合，語順が自由なのではなく，文節の順序が自由である．ただし，文の主動詞を含む文節は文の末尾に置かれなければならない．例えば，以下の 2 文は同じ意味を持つ：(1) 健が彼女に本を あげた．(2) 健が本を 彼女に あげた．この 2 例文で，最も右の文節「あげた」(動詞の語幹と，過去や完了を表すマーカで構成されている) は文の末尾に置かれていることに注意されたい．

ここで，上に述べたものも含めて，通常書き言葉の日本語で仮定される係り受けの制約条件をまとめておく．

- C1. 最も右の文節を除いて，全ての文節は必ず一つの係り先を持つ．
- C2. 係り先となる文節は，必ず係り元の文節の右側に位置する．

¹ 韓国語の *eojeol* も日本語の文節と似た概念である (Yoon, Choi, and Song 1999)．文献 (Abney 1991) で定義される英語のチャンク (chunk) も文節に近いといえる．

C3. 係り関係は交差しない².

これらの特徴は、基本的には韓国語やモンゴル語でも共通である。

2.2 日本語文解析の典型的な手順

日本語には前節のような特徴があるので、日本語文の解析では次のような手順が非常に一般的である：

- (1) 文を形態素に分割する (つまり形態素解析する)
- (2) それらを文節にまとめ上げる
- (3) 文節間の係り関係を解析する
- (4) それぞれの係り関係に agent, object, location などの意味的役割のラベルを付ける

我々は (3) における係り受け解析に焦点を置く。

3 関連研究

ここでは、主に時間計算量に重点を置いて関連研究を述べる。日本語はもちろん英語でも依存構造解析 (dependency analysis) は研究されている (Lafferty, Sleator, and Temperley 1992; Collins 1996; Eisner 1996)。これらの論文の解析アルゴリズムでは、 $O(n^3)$ の時間がかかる。ここで n は単語数である^{3,4}。

日本語の係り受け解析では、文中の二つの文節の係り受けの確率を使うことが非常に多かった。Haruno ら (Haruno et al. 1998) は、決定木を用いて係り受けの確率を推定した。Fujio と Matsumoto (Fujio and Matsumoto 1998) は Collins のモデル (Collins 1996) の修正版を日本語の係り受け解析に適用した。Haruno らと、Fujio と Matsumoto の両グループとも CYK 法を用いている。これは $O(n^3)$ の時間がかかる。ここで n は文の長さ、つまり文節数を表している。Sekine ら (Sekine et al. 2000) は最大エントロピー法 (Maximum Entropy Modeling; ME) を係り受けの確率の推定に使い、後方ビームサーチ (文末から文頭に向かうビームサーチ) で最もよい解析結果を見つける。このビームサーチのアルゴリズムは $O(n^2)$ の時間がかかる。Kudo と Matsumoto (Kudo and Matsumoto 2000) らも同じ後方ビームサーチを ME ではなくサポート

² 「これが僕は正しいと思う」という例文のようにこの制約条件が成立しないこともあるが、書き言葉では殆どの場合成り立つ (長尾真, 佐藤理史, 黒橋禎夫, 角田達彦 1996, 187 ページ)。

³ Nivre (Nivre 2003) は、projective dependency parsing の決定的なアルゴリズムを提案している。このアルゴリズムの時間計算量の上限は $O(n)$ である。このアルゴリズムをスウェーデン語のテキストで評価している。

⁴ Ratnaparkhi は実際の処理時間が $O(n)$ となる句構造を返す英語のパパーザについて述べている (Ratnaparkhi 1997)。これは依存構造解析について述べたものではない。このアルゴリズムでは、解析途中の各ノードに対して、いくつかの手続きを行ない、その中で確率値の高い K 個の導出 (derivation) を残して解析を進める (幅優先探索)。そのため、時間計算量の上限は $O(n^2)$ と考えられる。

ベクタマシン (SVMs) とともに用いている⁵.

二つの文節間の係り受けの確率を使わない統計的手法も少ないながらある. 一つは Sekine の決定的有限状態変換器を用いる手法 (Sekine 2000) である. Sekine は, 係り先の場所の 97% は文中の五つの候補でカバーされると報告している. 似た現象は Maruyama と Ogino (Maruyama and Ogino 1992) も観測している. これらの調査にもとづき, Sekine は, 決定的有限状態変換器を用いる効率のよい解析アルゴリズムを提案している. このアルゴリズムは, 考慮する係り先の文節数を制限することでしらみつぶしに探索することを避け, $O(n)$ の時間計算量となっている. しかしながら, 彼のパーザは京大コーパスに対して 77.97% の係り受け正解率 (定義は第 6.3 節で述べる) しか得られていない. これは, 89% を超える現在の最高精度よりもかなり低い.

2 文節間の係り受けの確率を用いない別の興味深い手法は, Kudo と Matsumoto (Kudo and Matsumoto 2002) による Cascaded Chunking Model である. このモデルは (Abney 1991; Ratnaparkhi 1997) のアイデアにもとづく. 彼らはこのモデルと SVMs を用いて, 89.29% を得ている. 彼らの手法では, 解析時に評価される係り関係の数は CYK 法や後方ビームサーチよりも相当少ないが, それでも時間計算量の上限は $O(n^2)$ である.

以上見たように, 高い精度を保ちつつ線形時間の処理を保証して, どのように日本語係り受け解析を行なうかは, まだ解決されていない問題である. 以下に記述するアルゴリズムがこの問題に対する答えとなろう.

4 アルゴリズム

本節では, 提案アルゴリズムを解析時に使うものと, 学習時に使うものとに分けて記す. 解析時のアルゴリズム, その時間計算量, 学習時のアルゴリズムを順に述べ, 最後に提案アルゴリズムの特徴のまとめと関連研究との理論的な比較を述べる.

4.1 文を解析するアルゴリズム

我々の提案する係り受け解析のアルゴリズムの擬似コードを図 1 に示す. このアルゴリズムは, ある文節が別の文節に係るかどうかを決定する推定器 (estimator) とともに用いる. 推定器の典型的なものとして, SVM や決定木などの訓練できる分類器が考えられる. ここでは, 文中の二つの文節の係り関係を推定できる, つまり係るか否かを決定できる何らかの分類器があり, その分類器の時間計算量は文の長さに影響されないと仮定しておく.

係り関係の推定器を別にすれば, このアルゴリズムで使うデータ構造はわずか二つである.

⁵ 彼らは, 係り先候補間の係りやすさの相対的な大小関係をモデル化する手法も報告している (工藤, 松本 2005). 係り受けのモデルは異なるが, 同じく後方ビームサーチを用いている.

```

// 入力: N: 一文中の文節の数
//   w[ ]: 処理対象の文の文節列を保持する配列
// 出力: outdep[ ]: 解析結果, つまり文節間の係り関係を格納する整数の配列.
//   例えば, j 番目の文節の係り先 ID は outdep[j] で表現される.
//
// stack: 係り元文節の ID を保持する. もし空なら, このスタックに対する
//   pop メソッドは, EMPTY (-1) を返す.
// function estimate_dependency(j, i, w[ ]):
//   j 番目の文節が i 番目の文節に係ると判断するとき, 非ゼロを返す.
//   それ以外るとき, ゼロを返す.
function analyze(w[ ], N, outdep[ ])
stack.push(0); // ID 0 をスタックに積む.
for (int i = 1; i < N; i++) { // 変数 i は係り先文節, 変数 j は係り元文節を指すのに使う.
  int j = stack.pop(); // スタックから値を降ろし, 変数 j にセットする.
  while (j != EMPTY && (i == N - 1 || estimate_dependency(j, i, w))) {
    outdep[j] = i; // j 番目の文節が i 番目の文節に係る.
    j = stack.pop(); // 次にチェックすべき係り元文節の ID をスタックから降ろす.
  }
  if (j != EMPTY)
    stack.push(j);
  stack.push(i);
}

```

図 1 提案手法の擬似コード (解析時). ここで “ $i == N - 1$ ” は i 番目の文節が文末の文節であることを意味している.

一つは入力に関するもので, もう一つは出力に関するものである. 前者は, チェックすべき係り元の文節の ID を保持するためのスタックである. 後者は, 既に解析された係り先文節の ID を保持する整数の配列である.

以下では, 例を使いながら先に示したアルゴリズムの動作を説明する. 図 1 の擬似コードに沿って, 図 3 にある例文を解析してみよう. 説明のため, 図 1 の *estimate_dependency()* として完璧な分類器があるとする. この分類器は図 3 の例文に対して必ず正しい結果を返すとする.

まず始めに 0 (健が) をスタックに積む. 0 は文の先頭の文節の ID である. この初期化の後, for ループの各繰り返し (iteration) の中で解析がどのように進むかを見る. 最初の iteration では, 0 番目の文節と 1 番目の文節 (彼女に) の係り関係をチェックする. 0 番目の文節は 1 番目の文節に係らないから, 0 をスタックに積み, 次に 1 を積む. ここで, スタックの底は 0 であって 1 ではないことに注意されたい. より小さい ID が必ずスタックの底のほうに保持される. この性質のおかげで, 非交差の制約 (第 2.1 の C3) を破らずに解析を進めることができる.

```
// indep[ ]: 訓練事例から与えられる正しい係り受け関係を保持する整数の配列.
//
// function estimate_dependency(j, i, w[ ], indep[ ]):
//     indep[j] == i が満たされるとき, 非ゼロを返す. それ以外るとき, ゼロを返す.
//     同時に, j 番目の文節が i 番目の文節に係るか係らないかに応じて, 1 あるいは -1 の
//     ラベル付きで, 素性ベクタ (エンコードされた事例) を出力する.
function generate_examples(w[ ], N, indep[ ])
stack.push(0);
for (int i = 1; i < N; i++) {
    int j = stack.pop();
    while (j != EMPTY && (i == N - 1 || estimate_dependency(j, i, w, indep))) {
        j = stack.pop();
    }
    if (j != EMPTY)
        stack.push(j);
    stack.push(i);
}
```

図 2 訓練事例を作るための擬似コード. 変数 $w[]$, N と $stack$ は図 1 のものと同じである.

	健が	彼女に	あの	本を	あげた.
文節 ID	0	1	2	3	4
係り先	4	4	3	4	-

図 3 例文

2 回目の iteration では, 1 をスタックから降ろし, 1 番目の文節と 2 番目の文節 (あの) の係り関係をチェックする. 1 番目の文節は 2 番目には係らないので, 再び 1 と 2 をスタックに積む.

3 回目の iteration では, 2 をスタックから降ろし, 2 番目の文節と 3 番目の文節 (本を) の係り関係をチェックする. 2 番目の文節は 3 番目に係るので, その関係を $outdep[]$ に格納する. $outdep[j]$ の値は, 第 j 番目の文節の係り先を表す. 例えば $outdep[2]=3$ は 2 番目の文節の係り先は 3 番目の文節であることを意味している.

次に, 1 をスタックから降ろし, 1 番目の文節と 3 番目の文節の係り関係をチェックする. 1 番目の文節は 3 番目に係らないので 1 を再びスタックに積む. その後, 3 をスタックに積む. この段階で, スタックには頭から底に向けて 3, 1, 0 が格納されている.

3 回目の iteration では, 3 をスタックから降ろす. 3 番目の文節と 4 番目の係り関係はチェックする必要がない. 4 番目の文節は文中の最も末尾の文節であり, 3 番目の文節は必ず 4 番

目にかかるからである．そこで $outdep[3] = 4$ とする．次に 1 をスタックから降ろす．この場合も，1 番目の文節と 4 番目との係り関係のチェックはする必要がない．同様に，0 番目の文節も 4 番目に係る．結果として $outdep[1] = 4$ と $outdep[0] = 4$ となる．この時点でスタックは空となり，この解析の関数 $analyze()$ は終了する．解析結果である係り受け関係は，配列 $outdep[]$ に得られている．

4.2 時間計算量

一見したところ，提案したアルゴリズムの時間計算量の上限は，2 重ループを含むため $O(n^2)$ と思える．しかしそうではない．時間計算量の上限が $O(n)$ であることを，図 1 における while ループの条件部が何回実行されるかを考えることによって示す．条件部が失敗する回数と成功する回数とに分けて考える．

while ループの条件部は $N-2$ 回失敗する．何故なら外側の for ループが 1 から $N-1$ へ，つまり $N-2$ 回実行されるからである．もちろん，while ループが無限ループになることはない．while ループの内部で stack に値を新たに積むことなく降ろしているのです，いつか $j == \text{EMPTY}$ になる．つまり $j != \text{EMPTY}$ を満たさず，while ループを抜けることになる．

一方，この条件部は $N-1$ 回成功する．何故なら $outdep[j] = i$ が $N-1$ 回実行されるからである．文節 j それぞれについて， $outdep[j] = i$ は必ず一度だけ実行される． $j = \text{stack.pop}()$ を実行すると， j に格納されている値は失われ，その値は二度と再びスタックに積まれることはないからである．つまり，while ループは，高々 $N-1$ 回実行される．これは末尾の文節を除く文節数に等しい．

結局，while ループの条件部の実行回数は，失敗回数 $N-2$ と 成功回数 $N-1$ を合計し $2N-3$ となる．これは時間計算量の上限が $O(n)$ となることを意味している．

4.3 訓練事例を作り出すアルゴリズム

前節のアルゴリズムで用いる分類器のための訓練事例を作り出すには，図 2 に示すアルゴリズムを使う．図 1 にある解析用のアルゴリズムと殆ど同じである．違いは， $indep[]$ を使って $estimate_dependency()$ が正しい係り関係の判定を返す点と， $outdep[]$ に係り先の ID を格納する必要がない点である．

4.4 特徴のまとめと関連研究との理論的な比較

我々の提案アルゴリズムは次のような特徴を持つ：

- F1. 特定の機械学習の方法に依存しない．訓練できる分類器ならどれでも使える．
- F2. 左から右へ文を一度だけスキャンする．

F3. 時間計算量の上限は $O(n)$ である．アルゴリズム中，最も時間を消費する部分である分類器の呼び出しの回数は，高々 $2N - 3$ 回である．

F4. アルゴリズムの流れとデータ構造は非常に簡単である．そのため，実装も易しい．

我々のアルゴリズムと最も関連が深いモデルの一つは，(Kudo and Matsumoto 2002) の Cascaded Chunking Model である．彼らのモデルと我々のアルゴリズムは F1 を始め多くの共通点がある⁶．彼らのモデルと我々のアルゴリズムの大きな違いは，入力文を何回スキャンするかにある (F2)．彼らのモデルでは，入力文を何回かスキャンする必要がある，これは計算上の非効率につながっている．最悪の場合では $O(n^2)$ の計算が必要になる．我々の解析アルゴリズムは，左から右に一度だけしかスキャンせず，実時間の音声認識などのような実用的なアプリケーションに対しても，より好適であろう．それに加えて，アルゴリズムの流れと利用するデータ構造は，Cascaded Chunking Model で使われるものよりもずっと簡単である (F4)．彼らのモデルでは，チャンクタグを保持する配列が必要となり，入力文を何度もスキャンする間，この配列は正しく更新されなければならない．

Nivre による Projective Dependency Parsing の手法 (Nivre 2003) も，我々のアルゴリズムと深い関係がある⁷．彼のアルゴリズムも，スタックを用いており，時間計算量の上限も $O(n)$ である．ただし，我々のアルゴリズムが日本語を対象とし，係り先が必ず右にあることを前提にしているのに対し，Nivre のアルゴリズムは依存関係の向きはどちらでもよい．その点では，彼のアルゴリズムは我々の手法をより一般的にしたものと考えることができる．一方，(Nivre 2003) では，単語間の依存関係を決めるルールを用意しておき，ある一定の優先度で選ぶとしている⁸．我々は，依存関係が一方方向である日本語に対して，機械学習を用いる方法を提示し，実際に検証している．

我々の解析アルゴリズムは，shift-reduce 法の最も簡単な形の一つと考えられる．典型的な shift-reduce 法との違いは，アクションの型を複数持つ必要がなく，スタックの先頭のみ調べればよいという点である．これらの簡潔さの理由は，日本語が制約 C2 (第 2.1 節参照) を仮定できること，文脈自由文法の解析ではなく，係り受け関係のみの解析であることの二つによる．

⁶ 文献 (Uchimoto, Sekine, and Isahara 1999) に代表される 2 文節間の係り受け確率を考える確率モデルと，Cascaded Chunking Model との比較は文献 (Kudo and Matsumoto 2002) で詳細になされている．ほぼ全ての議論が確率モデルと我々の提案手法との比較にも当てはまる．

⁷ Nivre のいう “projective” とは，依存関係 (係り関係) が交差しないことと同じである．

⁸ スウェーデン語を対象に 126 のルールを人手で記述している．ルールを，「右向きに係る」「左向きに係る」「対象語をスタックから降ろす」「対象語をスタックに積む」の四つのタイプに分け (詳細はここでは略す)，タイプ間の優先度は，事前に人手で決める場合のみ実験で検証されている．ここでいう対象語とは，アルゴリズム中で係り関係をチェックする対象となっている語を指す．

5 係り関係を推定するためのモデル

2 文節間の係り関係を推定するために、2 文節に関係する形態的、文法的情報を素性のベクタとして表現し、それを入力として分類器に係るか否かを判断させる。

その分類器として、サポートベクタマシン (SVMs) (Vapnik 1995) を用いた。SVMs は優れた特徴を持っている。その一つは、多項式カーネルを用いると、ある事例の持つ素性の組合せが自動的に考慮される点である。現在まで多数の分類タスクに対して、非常に優れた性能が報告されている。SVMs の形式的な記述については、文献 (Vapnik 1995) を参照されたい。

素性として、第 5.1 節以降で述べるものを用いた。実際には 2 文節間の係り関係の推定の処理は、図 1 の *estimate_dependency()* の中で行なう。推定しようとする 2 文節の形態的、文法的情報を素性のベクタとして表現し、SVMs に係るか否かを判定させることになる。以下では、まず基本となる標準素性を述べ、次にそれに追加して用いる付加的な素性について述べる。

5.1 標準素性

ここで「標準素性」といっているものは、(Uchimoto et al. 1999; Sekine et al. 2000; Kudo and Matsumoto 2000, 2002) でほぼ共通に使われている素性セットを指す。それぞれの文節について以下の素性を使った：

- (1) 主辞品詞, 主辞品詞細分類, 主辞活用型, 主辞活用形, 主辞表層形
- (2) 語形品詞, 語形品詞細分類, 語形活用型, 語形活用形, 語形表層形
- (3) 句読点
- (4) 開き括弧, 閉じ括弧
- (5) 位置一文の先頭か文の末尾か

ここで主辞とは、概ね文節内の最も右の内容語に相当する。品詞が特殊、助詞、接尾辞を除き、最も文末に近い形態素を指す。語形とは、概ね文節内の最も右の機能語に相当する。品詞が特殊となるものを除き、最も文末に近い形態素を指す。

これらに加えて、2 文節間のギャップに関する素性も用いた。距離 (1, 2-5, 6 以上) と、助詞、括弧、句読点である。

5.2 注目文節の前後の文節

注目している係り元文節、係り先文節の前後の文節も有用である。それらが固定的な表現や格フレーム、その他の連語を表すことがあるからである。第 j 番目の文節が係り元文節で、第 i 番目の文節が係り先文節の候補だとする。第 j 番目の文節と第 i 番目の文節の前後にある文節のうち、次の三つを素性として考慮する：第 $j-1$ 番目の文節 (j に係るときのみ) と、第 $i-1$ 番目の文節、第 $i+1$ 番目の文節の三つである。我々のアルゴリズムでは、 $j < i-1$ を満たし第 j 番目の文

節が i 番目の文節に係るかチェックしているとき、 $i-1$ 番目の文節は必ず i 番目の文節に係っていることに注意されたい。提案手法におけるデータ構造を簡単しておくために、 j 番目、 i 番目の文節からさらに遠い文節については考慮しなかった。なお、 $j-1$ 番目の文節が j 番目の文節に係るかどうかは `outdep[]` を見れば簡単にチェックできる。注目している文節の前後を使うのは、(Kudo and Matsumoto 2002) における動的素性⁹と似ている。

5.3 文節内の追加素性

「標準素性」では、文節内に二つ以上の機能語を含むとき格助詞の情報を見落とすことがある。ある文節が格助詞と提題助詞¹⁰を持つとする。このとき格助詞の後ろに提題助詞が来る。それゆえ、格助詞の情報を見落としてしまう。「標準素性」では文節内の最も右の機能語しか素性として扱われないからである¹¹。こういった情報を見落とさないように、文節内の全ての格助詞を素性として扱う。

「標準素性」で見落とされる重要な情報は他にもある。それは、係り先候補の文節の最も左の語の情報である。この語は係り元の文節の最も右の語と慣用表現のような強い相関関係を持つことも多い。

これに加えて、係り先候補文節の直後の文節の表層形も素性として使う。これは、第 5.2 節の素性とともに行う。

5.4 並列句のための素性

並列構造を正しく認識することは、長い文を正しく解析する際に最も難しいことの一つである。Kurohashi と Nagao は、二つの文節列の類似度を計算することによって並列句を認識する手法を提案している (Kurohashi and Nagao 1994)。

現在までのところ、機械学習を使うシステムの中で並列構造を認識するための素性はあまり研究されていない。我々は最初のステップとして、並列構造を認識するための基本的な二つの素性を試した。注目している文節がキー文節 (*distinctive key bunsetsu*) (Kurohashi and Nagao 1994, page 510) であるとき、この二つの素性は使われる。一つ目の素性は、係り元文節がキー文節であるときアクティブになるものである。もう一つの素性は、係り元文節がキー文節で、その係り元文節と係り先候補の文節の主辞表層形が一致していればアクティブになるものである。単純さを保つため、対象とする主辞品詞は名詞のみとした。

⁹ Kudo らのモデルは、以下の三つから動的素性を作る: j 番目の文節に係るもの (タイプ B), i 番目の文節に係るもの (タイプ A), i 番目の文節の係り先 (タイプ C)。我々の提案手法では解析が左から右に進むので、タイプ C の素性を取り入れるためには、スタッキング (Wolpert 1992) やその他の手法を用いる必要がある。

¹⁰ 提題助詞とは、主題を提示する助詞である (益岡隆志, 田窪行則 1992, page 50)。代表的な提題助詞は「は」である。

¹¹ 例えば、係り元文節が「本-に-は」の場合、第 5.1 の語形の素性として「は」に関する素性が採用される。「に」に関する素性は使われない。

6 実験と考察

提案アルゴリズムを利用したパーザを C++ で実装し、その時間計算量の振る舞いや解析精度を実験的に評価した。

6.1 コーパス

提案アルゴリズムを評価するために、京大コーパス Version 2 (Kurohashi and Nagao 1998) を使った。新聞記事の 1 月 1 日から 1 月 8 日分 (7,958 文) を訓練事例とし、1 月 9 日分 (1,246 文) をテスト事例とした。1 月 10 日分を開発用に用いた。これらの記事の使い方は (Uchimoto et al. 1999; Sekine et al. 2000; Kudo and Matsumoto 2002) と同じである。

6.2 SVM の設定

独自に C++ で実装した SVMs のツールを用いた。カーネルとして、3 次の多項式カーネルを用いた。特に記述がない限り誤分類のコストは 1 に設定した。

6.3 実験結果

解析精度 テスト事例に対する我々のパーザの性能を表 1 に示す。従来研究との比較のために、性能評価には京大コーパスで標準的に使われる尺度である係り受け正解率と文正解率の二つを用いる。係り受け正解率とは、正しく解析された係り受けの割合であり (他の多くの文献と同様、文末の一文節を除く)、文正解率とは、全ての係り関係が正しく解析された文の割合である。

「標準素性」を用いた場合の精度は比較的よい。実際、この係り受け正解率は動的素性を用いないときの Cascaded Chunking Model (Kudo and Matsumoto 2002) とほぼ同じである。第 5 節で述べた全ての素性を用いた場合、我々のパーザは 89.56% の係り受け正解率を得た。これは京大コーパス Version 2 に対して公表されている精度の中で最もよいものである。

時間計算量の漸近的な振る舞い 図 4 に、我々のパーザのテスト事例に対する実行時間を示

表 1 テスト事例に対する精度

	係り受け正解率 (%)	文正解率 (%)
標準素性	88.72	45.88
全て	89.56	48.35
前後文節素性 (5.2 節) なし	88.91	46.33
文節内追加素性 (5.3 節) なし	89.19	47.05
並列句素性 (5.4 節) なし	89.41	47.86

図 4 一文あたりの処理時間

す。これはワークステーション (Ultra SPARC II 450 MHz, 1GB メモリ) を用いて計測した。図 4 より実行時間の上限が文の長さ按比例しているのが分かる。これは、第 4.2 節で行なった理論的な分析と一致している。

この実験結果を見て、確かに従来研究よりも時間計算量の上限は低く抑えられているが、我々のパーザの実際の処理時間はそれほど速くないと思われるかもしれない。パーザのこの遅さの主たる原因は、SVMs におけるカーネル評価での膨大な計算のせいである。我々の実験では、SVM の分類器は 4 万個以上のサポートベクタを持っている。それゆえ、係り関係を判定するたびに膨大な内積計算が必要となる。幸い、この問題に対する解決策は既に Kudo と Matsumoto (Kudo and Matsumoto 2003) によって与えられている。彼らは高次の多項式カーネルを線形カーネルに変換する手法を提案し、変換された線形カーネルでは、精度を保ったまま元の多項式カーネルよりもおよそ 30 から 300 倍高速だったと報告している。彼らの手法を我々のパーザに適用すれば、処理時間も十分高速化されるだろう。

彼らの手法を用いればどのくらい我々のパーザの速度が改善されるか粗く見積もるために、線形カーネルを用い、同じテスト事例に対してパーザを走らせてみた。図 5 に、線形カーネルを用いたパーザの処理時間を示す。なお計測には多項式カーネルを用いた場合と同じマシンを使った。3 次の多項式カーネルを使う場合に比べて相当に高速である。非常に長い文であっても 0.02 秒以内で解析が行なえている。加えて、このパーザのスピードばかりでなく精度も我々

図 5 線形カーネルを使った場合の一文あたりの処理時間．誤分類のコストは 0.0056.

が期待した以上だった．係り受け正解率は 87.36%，文正解率は 40.60% に達した．これらの精度は，素性の組合せを人手で選択して追加しているパーザ (Uchimoto et al. 1999) よりもわずかに良い．

6.4 関連研究との比較

我々のパーザと関連研究におけるパーザとを時間計算量と精度の点から比較する．比較のサマリを表 2 に示す．我々のアルゴリズムと SVMs と組み合わせたものが時間計算量の点から優れた性質を持ち，加えてトップレベルの精度が得られている．

文献 (Kudo and Matsumoto 2002) との比較は，第 4.4 の記述にゆずる．Uchimoto ら (Uchimoto et al. 1999) は最大エントロピー法と後方ビームサーチを用いている．文献 (Sekine et al. 2000) によれば，解析時間は n^2 に比例するとのことである．これに対し，我々のパーザは線形時間で文を解析し，精度もよい．工藤と松本 (工藤，松本 2005) の「相対モデル」のパーザ¹²も，後方ビームサーチを用いているので，時間計算量という点では Uchimoto らのパーザと同様である．文献 (工藤，松本 2005) では，「相対モデル」のパーザは，京大コーパス Version 3.0 に対して，係り受け正解率 91.37% を得，Cascaded Chunking Model は 91.23% を得たと報告されて

¹² 京大コーパス Version 2 に対する精度が不明なため，表 2 にはあげていない．

表 2 関連研究との性能の比較. KM02 = Kudo and Matsumoto 2002, KM00 = Kudo and Matsumoto 2000, USI99 = Uchimoto et al. 1999, and Seki00 = Sekine 2000. 提案アルゴリズムを Stack Dependency Analysis と記述.

	アルゴリズム/モデル	時間計算量	係り受け正解率 (%)
本論文	Stack Dependency Analysis (SVMs)	n	89.56
	Stack Dependency Analysis (linear SVMs)	n	87.36
KM02	Cascaded Chunking Model (SVMs)	n^2	89.29
KM00	後方ビームサーチ (SVMs)	n^2	89.09
USI99	後方ビームサーチ (ME)	n^2	87.14
Seki00	決定的有限状態変換器	n	77.97

いる. 我々のパーザは, 京大コーパス Version 2 において, Cascaded Chunking Model の精度を 0.27 ポイント上回っていることを考えると, 我々のパーザと「相対モデル」パーザとの差も大きなものではないと判断できる¹³. また, 我々と同様 Sekine (Sekine 2000) も線形時間で処理が進む非常に高速なパーザを提案している. 彼の手法は, 後方から係り先を決定していく. 係り元の文節の語形の情報と, 係り先候補 (5つまで) の主辞の情報から, 係り先を一つに決める決定的有限状態変換器を用いている. 係り元の語形や 5つの係り先候補の主辞の情報を細かく区別すると, 状態 (state) の数が多くなりすぎ, 大量のメモリを消費するため, 係り元の語形の状態を 40 に, 係り先の主辞の状態を 18 に限定している. 品詞や活用形の情報のみ利用している. このため, 精度が大きく犠牲になっている.

7 おわりに

我々は線形時間で処理を行なう日本語係り受け解析のアルゴリズムを提案した. 京大コーパス Version 2 に対して実験を行ない, 時間計算量と解析精度を調べた. 時間計算量の上限は $O(n)$ であることが確認でき, 解析精度も従来報告されているものを上回った.

精度の差は従来研究で報告されているものと大きくないため, 精度面からの優位性は結論できないが, 本研究で (1) 提案アルゴリズムが理論的にも実験的にも時間計算量の上限が $O(n)$ で抑えられていることと, (2) 時間計算量を抑え, 左から右へ一度しかスキャンしないにも関わらずトップレベルの精度が得られることの二つを示せた意義は大きいと考える.

後方の文節を直接考慮しない提案アルゴリズムに一定の限界があることは明らかであるが,

¹³ 「相対モデル」が Cascaded Chunking Model よりも, 長距離の文節間の係り受け F 値が上回っている (工藤, 松本 2005) ことは注目に値する. 我々の提案手法も Cascaded Chunking Model と同様, 直後に係りやすいという性質を利用しているため, 「相対モデル」のほうが提案手法よりも長距離の文節間の係り受け F 値がよい可能性が高い.

係り先として考慮する文節の数を増やしても精度が向上するとは限らず、その解決は単純ではない。我々はスタッキングにより精度が向上しないか検討したいと考えている。また、並列構造の認識についてもよりよいモデルを提案したい。

参考文献

- Abney, S. P. (1991). “Parsing by Chunks.” In Berwick, R. C., Abney, S. P., and Tenny, C. (Eds.), *Principle-Based Parsing: Computation and Psycholinguistics*, pp. 257–278. Kluwer Academic Publishers.
- Collins, M. (1996). “A New Statistical Parser Based on Bigram Lexical Dependencies.” In *Proc. of ACL-96*, pp. 184–191.
- Eisner, J. M. (1996). “Three New Probabilistic Models for Dependency Parsing: An Exploration.” In *Proc. of COLING-96*, pp. 340–345.
- Fujio, M. and Matsumoto, Y. (1998). “Japanese Dependency Structure Analysis based on Lexicalized Statistics.” In *Proc. of EMNLP-1998*, pp. 88–96.
- Haruno, M., Shirai, S., and Ooyama, Y. (1998). “Using Decision Trees to Construct a Practical Parser.” In *Proc. of COLING/ACL-98*, pp. 505–511.
- Kudo, T. and Matsumoto, Y. (2000). “Japanese Dependency Structure Analysis Based on Support Vector Machines.” In *Proc. of EMNLP/VLC 2000*, pp. 18–25.
- Kudo, T. and Matsumoto, Y. (2002). “Japanese Dependency Analysis using Cascaded Chunking.” In *Proc. of CoNLL-2002*, pp. 63–69.
- Kudo, T. and Matsumoto, Y. (2003). “Fast Methods for Kernel-based Text Analysis.” In *Proc. of ACL-03*, pp. 24–31.
- 工藤拓, 松本裕治 (2005). “相対的な係りやすさを考慮した日本語係り受け解析モデル.” *情報処理学会論文誌*, **46** (4), pp. 1082–1092.
- Kurohashi, S. and Nagao, M. (1994). “A Syntactic Analysis Method of Long Japanese Sentences Based on the Detection of Conjunctive Structures.” *Computational Linguistics*, **20** (4), pp. 507–534.
- Kurohashi, S. and Nagao, M. (1998). “Building a Japanese Parsed Corpus while Improving the Parsing System.” In *Proc. of the 1st LREC*, pp. 719–724.
- Lafferty, J., Sleator, D., and Temperley, D. (1992). “Grammatical Trigrams: A Probabilistic Model of Link Grammar.” In *Proc. of the AAAI Fall Symp. on Probabilistic Approaches to Natural Language*, pp. 89–97.
- Maruyama, H. and Ogino, S. (1992). “A Statistical Property of Japanese Phrase-to-Phrase

- Modifications.” *Mathematical Linguistics*, **18** (7), pp. 348–352.
- 益岡隆志, 田窪行則 (1992). 基礎日本語文法 —改訂版—. くろしお出版.
- 長尾真, 佐藤理史, 黒橋禎夫, 角田達彦 (1996). 自然言語処理. 岩波書店.
- Nivre, J. (2003). “An Efficient Algorithm for Projective Dependency Parsing.” In *Proc. of IWPT-03*, pp. 149–160.
- Ratnaparkhi, A. (1997). “A Linear Observed Time Statistical Parser Based on Maximum Entropy Models.” In *Proc. of EMNLP-1997*, pp. 1–10.
- Sekine, S. (2000). “Japanese Dependency Analysis using a Deterministic Finite State Transducer.” In *Proc. of COLING-00*, pp. 761–767.
- Sekine, S., Uchimoto, K., and Isahara, H. (2000). “Backward Beam Search Algorithm for Dependency Analysis of Japanese.” In *Proc. of COLING-00*, pp. 754–760.
- Uchimoto, K., Sekine, S., and Isahara, H. (1999). “Japanese Dependency Structure Analysis Based on Maximum Entropy Models.” In *Proc. of EACL-99*, pp. 196–203.
- Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag.
- Wolpert, D. H. (1992). “Stacked Generalization.” *Neural Networks*, **5**, pp. 241–259.
- Yoon, J., Choi, K., and Song, M. (1999). “Three Types of Chunking in Korean and Dependency Analysis Based on Lexical Association.” In *Proc. of the 18th Int. Conf. on Computer Processing of Oriental Languages*, pp. 59–65.

略歴

颯々野 学：1991 年京都大学工学部電気工学第二学科卒業。同年より富士通研究所研究員。1999 年 9 月より 1 年間、米国ジョンズ・ホプキンス大学客員研究員。2006 年 3 月よりヤフー株式会社勤務。自然言語処理の研究に従事。2006 年 10 月より京都大学大学院情報学研究科知能情報学専攻博士後期課程在学。言語処理学会、情報処理学会、ACL 各会員。

(2005 年 12 月 5 日 受付)

(2006 年 7 月 25 日 再受付)

(2006 年 8 月 26 日 採録)