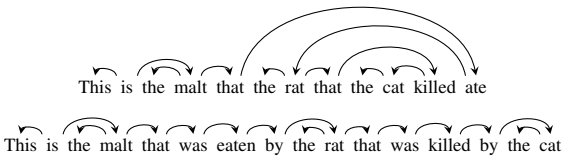
言語の逐次性は係り受け構造に影響を与えるか

能地 宏　　宮尾 祐介
総合研究大学院大学 情報学専攻 / 国立情報学研究所
{noji, yusuke}@nii.ac.jp

1　はじめに

言語は、その進化の過程で様々な要因を受けて形成されてきたと考えられる。古典的には、Zipf (1949) は単語の長さと使用頻度の間に相関を見いだしており、これはコミュニケーションをより効率化させるように単語が形成されていった結果であると考えられる。このような性質は語彙レベルを超えた統語レベルでも成り立つだろうか。

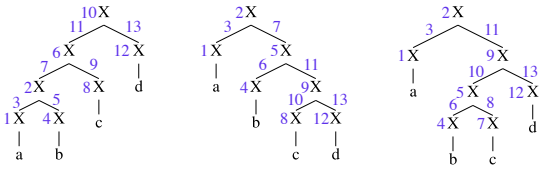
Gildea and Temperley (2010) は、文の単語間の係り関係の総和を小さくするようなバイアスが語順に影響を与えたと仮定し、Treebank から抽出した係り受け構造を用いてこれを検証している。Gildea and Temperley (2010) のバイアスは dependency length minimization (DLM) と呼ばれる。これが生まれた背景としては例えば、心理言語学における次の二つの文の比較があげられる (Jaeger and Tily, 2011)。



これらは同じ意味をもつが、後者のほうが易しい。Gibson (1998) は心理言語学の立場から、前者の難しさは that → ate などの長い係り関係に起因すると説明する。この考えを突き詰めると、通常の文にも、係り関係は短い方が望ましいというバイアスがかかっている可能性が考えられ、これを定量的に評価したのが Gildea and Temperley (2010) である。

本稿では、DLM とは異なる種類のバイアスを取り上げ、そのバイアスが文法の形成に与えた影響を評価する。本稿で着目するバイアスは言語の逐次性であり、我々はこれを文を解析するために必要な記憶容量と結びつけて定義する。しかしながら、ある文に対してどのように必要な記憶容量を定義するかは一意ではない。本稿ではまず次節で、句構造木に対して知られる left-corner parser と、これが言語学的に意味のある記憶容量を見積もる能力があることを述べる。その後、この left-corner parser の考えを導入した新しい決定的な係り受け解析アルゴリズムを提案し、この parser で必要な記憶容量の小さな文を、逐次性の高い文と定義する。Penn Treebank から抽出した係り受け構造を対象に分析を行ったところ、もとのグラフ構造を保持したまま並び替えを行った文と比較して逐次性に大きな差が観測された。これは、英語に関しては逐次性が言語進化に影響を与えたことを示唆するものである。

2　中央埋め込みと left-corner



上に示した句構造は、それぞれ左枝分かれ、右枝分かれ、中央埋め込みと呼ばれる。左枝分かれは日本語などの head-final な言語でよく見られる構造であり、右枝分かれは英語などの head-initial な言語に典型的である。心理言語学によると、人間はこれらのうち、中央埋め込みの構造のみに理解の困難を示す (Abney and Johnson, 1991)。前節であげた This is the malt ... の文も、この中央埋め込みの構造をもつ。

文を処理するための記憶容量とは、特定の構文解析アルゴリズムを用いた際に必要なスタックの深さ、と言い換えられる。このとき、どのアルゴリズムを用いるかという点が重要となる。Bottom-up parser と top-down parser はよく知られる解析アルゴリズムであるが、これらの性質は上記の観測と合致しない。Bottom-up parser は右枝分かれの構造に、top-down parser は左枝分かれの構造に対し、それぞれ中央埋め込みよりも大きな記憶容量を必要とすることが知られている。中央埋め込み以外の構造に対する記憶容量を定数で抑える解析器として、Abney and Johnson (1991) は arc-eager left-corner parser を導入した[1]。上記の句構造のノードとエッジに振られた番号は、この left-corner parser に

[1] ここでの arc-eager は、Nivre (2004) のアルゴリズムとは関係がなく、従来の left-corner parsing と比較し、エッジを認識する条件を変更したという意味で使われている。

おいてそれらが認識される順番を表している。Left-corner は bottom-up と top-down の中間の戦略である。ある部分木の親ノードを認識する際、まずその左の部分木を完成させる。そして、両者間のエッジを認識する。上図の中央埋め込みでは、b を読み込んだ時点で手順 6 まで進むが、このとき二つの部分木は接続されていない。これにより必要な記憶容量が増加する。それに対し、左枝分かれと右枝分かれでは各文字を読み込む手前で全ての部分木は接続されており、記憶容量は 1 で抑えられる。

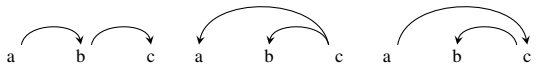
以上は句構造に関する理論である。本稿の興味は係り受け構造であるが、係り受けに関して left-corner のような分析は我々の知る限り行われていない。次節ではまず、shift-reduce 操作に基づく基本的な係り受け解析アルゴリズムである arc-standard と arc-eager の動作を確認し、これが中央埋め込み構造の難しさを正しく見積もることができないことを指摘する。その後、係り受けにおける中央埋め込み構造の難しさを正しく見積もる新しい係り受け解析アルゴリズムを提案する。

3 Left-corner dependency parser

3.1 従来法の問題点

句構造と係り受け構造との関係について簡単にまとめると、句構造の係り受け構造への変換には、各非終端記号について、どの子が head かという情報を必要とするが、係り受け構造から Chomsky normal form の句構造への変換は、特定の場合を除くと一意に行える。一意に定まらないのは、a⌢b⌢c の形の構造であり、この場合 a⌢b で先に句を作るか、b⌢c で先に句をつくるかの曖昧性が生じる。

決定的な係り受け解析器と、それが持つ逐次性については Nivre (2004) が議論している。彼は、3 つの語の並び a b c と、その間の係り関係を列挙し、arc-standard でその構造を処理する際にスタックがどう増えるかを検証している。これによると、係り受け構造を句構造に変換したものが右枝分かれの構造をもつ場合に必要なスタックが増加し、次の構造が該当する。



(1) (2) (3)

Arc-standard でこれらの構造を処理する際、まず a b c の全ての語を shift 操作で stack 上に移す必要が生じる。直感的には、arc-standard は bottom-up、すなわちある head に dependent を接続する際、まずその dependent を head とする木を完成させる必要があるため、句構造の bottom-up parser と同じく右枝分かれに対してコストが発生すると理解される。

以上をふまえ我々の目標を確認すると、句構造に変換した際に中央埋め込みを持つ係り受け構造のみに対してスタックが深くなるような解析器を得ることであるといえる。これまで見たように、arc-standard はこの性質を満たさない。Nivre (2004) は上の問題を一部解決するために、arc-eager でのスタック上の接続された部分木の数によって逐次性を提案している。この定義によると、上の例の (1) に対してコストを 1 で抑えることができる。これは、arc-eager では右側の dependent をつくる際 rightArc 操作によってスタック上で部分木を構築するからである。しかしながら、この定義では常に dependent が右側である構造に対してコストが増加しない。例えば次のような構造を考えると、これは句構造に変換した際には中央埋め込みの構造を持つが、arc-eager の部分木の数ではコストが増加しない。

3.2 提案アルゴリズム

提案アルゴリズムの核となるアイデアは、スタック上で定義される部分木が、実際にはまだ観測されていないノードを表すダミーノードを持つというものである。この概念の必要性を示す例として、ここでは上の (2) の係り関係を取り上げる。ここで定義するアルゴリズムは、スタックを増やさずにこの構造を処理できなければならない。そのためには、a b を読み込んだ時点で、これらが繋がることを予測し、一つの木を構成する必要がある。提案アルゴリズムはこの構造に対して次のように動作する。まず a を shift 動作でスタックに移動した後、これが続く語の左側の dependent になることを予測する。これを head のダミーノード ∘ を用いて a⌢∘ と表現する。この動作は一種の reduce である。続いて b を shift した後、この語も同じくダミーノードの dependent として、b⌢∘ のリンクが作られる。最後に c がこのダミーノードをうめ解析が完了する。

本節ではこれ以降、提案アルゴリズムの動作を形式的に定義する。なお紙面の都合上、left-corner parsing との等価性の証明は省略するが、これは部分木とそれに応じた分析法を列挙することで行うことができる。

提案アルゴリズムでは、shift-reduce の状態は $c = (\sigma, \beta)$ のペアで表される。σ は部分木を保持するスタックで、β は入力列を保持する。部分木は a_1, a_2 と表現し、a_1 はスタック先頭の部分木を表す。$j-1$ 単語まで

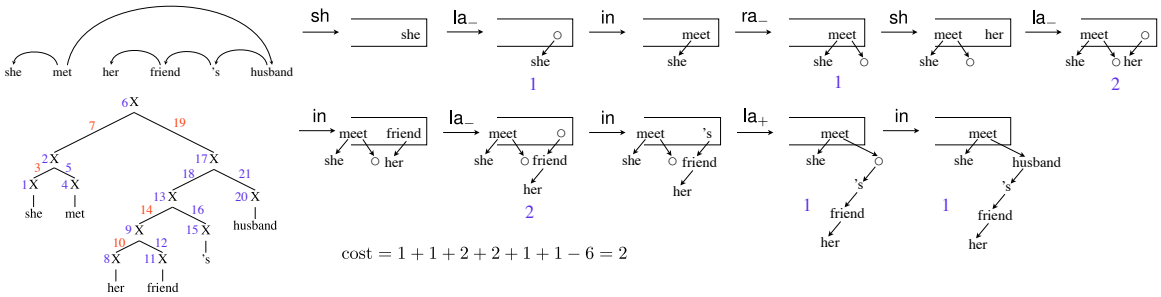


図 1: 提案アルゴリズムの解析例（右）と、対応する句構造木に対する left-corner parsing の解析順序（左下）。赤の数字は、次の単語を読み込む直前の番号を示し、このときの部分木の数と、右の解析の reduce 操作後のスタックの深さが一致する。

を処理したあとの状態を、$(\sigma|a_2|a_1, j|\beta)$ などと表記する。各部分木のノードを $n = \langle i, ch_l, ch_r \rangle$ の 3 つ組で表現し、i はそのノードのインデックス、ch_l は左の子のリスト、ch_r は右の子のリストを表す。さらに $n.i$ などと表記する。また部分木 a はスタック上で $a = \alpha|n_2|n_1$ と、別のリストを用いて表現される。このリストは、部分木 a の head からその右端の子を再帰的に列挙したものであり、例えば n_1 はノード n_2 の右端の子、すなわち $n_2.ch_r[len(n_2.ch_r)-1] = n_1$ が成り立つ。ここで $len(\cdot)$ はリストの長さを返す関数である。また部分木 a の head を $a.head$ と表す。ダミーノードは、そのインデックスが ϕ であることによって表現される。また、左右の子が空であるダミーノードを $\Phi = \langle \phi, [], [] \rangle$ と略記する。部分木 a のうち、ダミーノードを含まないものを complete、含むものを incomplete と呼んで区別する。

アクションは以下に列挙する 2 種類の shift 操作と 4 種類の reduce 操作からなる。

sh:	$(\sigma, j|\beta) \mapsto (\sigma|(\langle j, [], [] \rangle), \beta)$;
in:	$(\sigma|(\alpha|n_1), j|\beta) \mapsto (\sigma|(\alpha|n_1^{\mathsf{in}}), \beta)$ if $n_1.i = \phi$;
la$_-$:	$(\sigma|a_1, \beta) \mapsto (\sigma|(\langle \phi, [a_1.head], [] \rangle), \beta)$;
ra$_-$:	$(\sigma|a_1, \beta) \mapsto (\sigma|(\langle a_1.i, a_1.ch_l, a_1.ch_r|\Phi \rangle), \beta)$;
la$_+$:	$(\sigma|(\alpha|n_1)|a_1, \beta) \mapsto (\sigma|(\alpha|n_1^{\mathsf{la}+}), \beta)$ if $n_1.i = \phi$;
ra$_+$:	$(\sigma|(\alpha|n_1)|a_1, \beta) \mapsto (\sigma|(\alpha|n_1^{\mathsf{ra}+}|\Phi), \beta)$ if $n_1.i = \phi$

sh は単純な shift 操作である。in は Insert の略であり、スタック先頭にダミーノードがあるとき、それを単語 j でおきかえる。つまり $n_1^{\mathsf{in}} = \langle j, n_1.ch_l, n_1.ch_r \rangle$ が成り立つ。部分木を $a = (\alpha|n_1)$ と表したとき、これらのアクションは、ダミーノードが存在するとするなら必ず n_1 の位置にあることを保証する。またこの動作はダミーノードが a の head であるとき、つまり $a = (n_1); n_1.i = \phi$ の場合も有効である。

reduce 操作は、スタック先頭の部分木が complete であることを前提とする。la と ra はそれぞれ left-Arc、rightArc の略であるが、それぞれ二種類の動作が存在する。la$_-$ は、スタック先頭の部分木を左の子として持つダミーノードをつくり、スタック先頭をおきかえる。ra$_-$ は、スタック先頭の部分木の右の子に新しくダミーノードを追加する。la$_+$ と ra$_+$ は、スタック先頭の二つの部分木に対して作用する。la$_+$ は、スタックが $\sigma|a_2|a_1$ であるとき、a_1 を a_2 のダミーノードの左の子として追加する。従って、$n_1^{\mathsf{la}+} = \langle n_1.i, n_1.ch_l|a_1.head, n_1.ch_r \rangle$ が成立する。ra$_+$ は、まず a_2 のダミーノードの位置に a_1 の部分木を挿入し、ダミーノードをその右の子として追加する。$n_1^{\mathsf{ra}+} = \langle n_1.i, n_1.ch_l + a.head.ch_l, a.head.ch_r|\Phi \rangle$ となる。ただし $n_1.ch_l + a.head.ch_l$ は二つのリストの結合を意味する。

スタック先頭が a_1 であるとき、shift アクションは incomplete 状態の a_1 を complete にし、reduce アクションは complete 状態の a_1 を incomplete にする。従って shift と reduce は必ず交互におき、アクションの合計は文の長さ n に対して $2n-1$ となる。

Oracle　shift-reduce parser に対する oracle は、ある文 s_d が与えられたときに、それに対する正解のアクションの列を返す関数である。次に示すアルゴリズムによる解析は、各時点で、その文を句構造に変換した構造を left-corner parsing したときの接続されていない部分木の数と等しいスタックの深さをもつ性質をもつ。図 1 に、このアルゴリズムの動作の例を示す。以前に述べたように、係り受け解析から句構造への変換は一意ではないが、このアルゴリズムは、a⌒b⌒c の形に対して常に a⌒b から先に句をつくる変換を行った場合に対応している。

Algorithm 1 Gold action for state $c = (\sigma|a_1, \beta)$

```
if a_1 is incomplete then                    ▷ Select from shift actions
    If canInsert(c) then in else sh endif
else                                         ▷ Select from reduce actions
    If canLeftArc+(c) then la+
    elseif canRightArc+(c) then ra+
    elseif canRightArc-(c) then ra- else la-
end if
```

ここでの内部の関数は、そのアクションを行った際、その後正解の係り関係に辿り着けるかどうかを判定

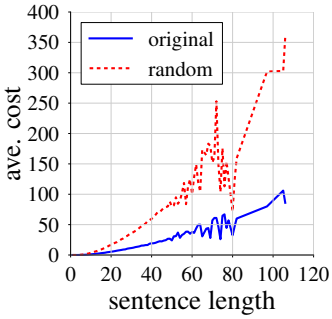


図 2: Penn Treebank での逐次性の評価。original は元の文、random はグラフ構造と projectivity を保持したまま、ランダムに並び替えた文である。

する。詳細は省略するが、例えば canRightArc+(c) は、$c = (\sigma|a_2|a_1, \beta)$ かつ $a_2 = (\alpha|n_2|n_1)$ のとき、$n_2.i \rightarrow a_1.head.i$ のリンクが存在し、かつ $a_1.head$ は β にあと一つの dependent を持てば true を返す。

4 実験

係り受け構造が与えられたとき、その構造を解釈するのに必要な記憶容量が心理言語学的に意味をもつアルゴリズムが定義できたので、次にコーパスを用いて、この記憶容量を抑えるようなバイアスが実際の文にどれほど存在するかを評価する。各単語を処理する際、スタックが深い状態が長く続くことは高負荷であると考えられるので、我々は文 s_d を処理する際のコストとして、各単語を処理した際のスタックの深さの合計を用いる。ただし、各状態のスタックは必ず一つの要素が存在しているため、文長を引いた数を最終的な値とする。図 1 に具体例を示す。

もしここで述べた逐次性が言語の語順の形成に影響を与えたのであれば、観測される文は必要なコストを小さく抑える傾向を持つはずである。図 2 に、Penn Treebank の各文を処理したときのコスト、およびグラフ構造を保持したままランダムに並び替えた文を処理したときのコストを、文の長さ毎に平均をとってプロットした。Penn Treebank の係り受け構造への変換には、Penn converter[2]を用いた。現在のアルゴリズムは non-projective な文を処理できないため、ランダムな並び替えは projectivity を満たすように行った。元の文は長い文でもコストを抑える傾向があるのに対し、ランダムに並べた文では平均として大きなコストがかかることが観測された。これは言語進化の過程で、left-corner parsing のコストを抑えるようなバイアスが、英語に関しては働いたことを示唆する。

[2] http://nlp.cs.lth.se/software/treebank_converter/

5 議論

心理言語学での、中央埋め込み構造の解釈が高コストであるという観測を受けて、それが通常の文の語順に影響を与えるかを調べたところ、英語においてはそのような傾向が観測された。Gildea and Temperley (2010) の DLM と本稿での基準（逐次性）の関係について考察すると、これらは等価ではないが、逐次性が低くなる文の多くは、長い係り関係を伴っていると考えられる。等価でないというのは、例えば the man in the park with the dog ran などの文を考えると、本稿の逐次性はこのように長い句を先に完成させることで生じる係り関係の長さとは無関係である。ドイツ語では DLM の影響が英語ほど顕著ではないことが観測されている (Gildea and Temperley, 2010)。もしこれらの言語で、本稿の基準でランダムな文との大きな差が観測されれば、それは DLM では説明できないバイアスを本稿の基準が捉えていることを示す。係り受け構造でのバイアスを調べることの一つの利点は、句構造に比べて利用できる言語の数が多いことであり、本稿で定義したアルゴリズムはそのためのフレームワークを提供する。今後は多言語での比較実験を通して、このような検証を行っていく予定である。

また本稿で導入した shift-reduce parser は、従来のアルゴリズムよりも認知的制約の点で望ましいモデルといえる。このモデルのパラメータを教師あり学習した際に、どれほどの性能を得ることができるかという点も興味深い。最後に、このような文法構造に内在するバイアスを調べることは、将来の教師なし構文解析の発展に繋がる可能性がある。Cohen et al. (2011) は shift-reduce 動作を生成モデル的に扱う方法を示した。本稿のアルゴリズムをこのように拡張することにより、言語の逐次性を考慮に入れた教師なし係り受け解析のモデルを得ることができると考えている。

参考文献

S. Abney and M. Johnson. Memory requirements and local ambiguities of parsing strategies. *Journal of Psycholinguistic Research*, 20(3):233–250, 1991.

S.B. Cohen, C. Gómez-Rodríguez, and G. Satta. Exact inference for generative probabilistic non-projective dependency parsing. In *Proc. of EMNLP*, pages 1234–1245, 2011.

E. Gibson. Linguistic complexity: Locality of syntactic dependencies. *Cognition*, 68(1):1–76, 1998.

D. Gildea and D. Temperley. Do grammars minimize dependency length? *Cognitive Science*, 34(2):286–310, 2010.

T.F. Jaeger and H. Tily. On language utility: Processing complexity and communicative efficiency. *Wiley Interdisciplinary Reviews: Cognitive Science*, 2(3):323–335, 2011.

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

G.K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley (Reading MA), 1949.