Vol.2014-NL-216 No.1
Vol.2014-SLP-101 No.1
2014/5/22

係り受け解析における Left-corner 型遷移

能地 宏[1,2,a)]　宮尾 祐介[1,2,b)]

概要：遷移型係り受け解析に、Left-corner 解析の考えを導入する。得られる遷移システムは、そのスタックサイズが中央埋め込みの構造のみに対し増加する性質を持つ。多言語ツリーバンクに対するオラクルの遷移を観察したところ、この提案システムが言語に依らず記憶容量を抑える性能をもつことが確認された。

1. はじめに

遷移型 (transition-based) の係り受け解析は、その処理の効率さによる工学的利点の他に、科学的な側面からも注目を集め、研究がなされてきた。これらは人間のように文を逐次的に処理するため、両者間の関係に着目した研究がいくつか存在する [4], [5], [22]。様々な文法理論のもとで、人間の認知的制約を満たすような構文解析器が注目を集めているが [9], [30]、係り受け構造を用いる利点は、その多言語データへの適応性である。このことから、Keller は認知的に望ましい性質を持つ係り受け解析器の重要性を説いている [17]。しかしながら、現在提案されている遷移型の係り受け解析アルゴリズムは全て、心理言語学的な観点から見ると適切とは言えない。これらは全て bottom-up 戦略に基づく[*1] [23] が、bottom-up もしくは top-down 戦略の解析は、人間にとって処理の難しい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という問題がある [1]。

本稿では left-corner 戦略に基づく係り受け解析のための遷移システムを提案する。句構造解析において、arc-eager left-corner 戦略は他の戦略と異なり中央埋め込みの難しさを正しく見積もることができることが知られている [1]。しかしながら、著者らの知る限り left-corner 戦略はこれまで係り受け解析には適用されてこなかった。提案システムはそのスタックサイズが中央埋め込み構造を処理するときにのみ増加する特性を持つ。

遷移システムを提示したのち、本稿では実際の文を解析する際に必要な記憶容量の観点からこのシステムの特徴づけを行う。特にこのシステムが実際に観測される文をスタックサイズを抑えたままで解析する性能を持つかどうかを確かめる。中央埋め込み構造が人間にとって処理が難しく、また人間が文を生成する際そのような解析の難しい文を避ける傾向があるのであれば、システムはそのような特性を持つはずである。本稿ではこの分析を、システムが与えられた文を解析する際のオラクルの遷移、すなわち正解の係り関係を導出するための動作列を観察することにより行う。英語では、Penn Treebank の各文に left-corner 変換を施して得られる句構造木が記憶容量を抑えた状態で解析が可能であることが示されている [32]。本稿での興味はこの特性の言語普遍性である。係り受け構造に対する left-corner 戦略による解析を可能にすることで、係り受け構造が付与された多言語コーパスを用いた言語横断的な分析が可能となる。この分析を行う動機は以下の二種類にまとめられる。一つは純粋な科学的な動機であり、人間が中央埋め込み構造を避ける傾向はどんな言語の話者にも共通して成り立つ性質かどうかを確かめることである。心理言語学における中央埋め込み構造の難しさに関する観察はほとんど英語で行われてきたこともあり、この問いは自明ではない。二つ目の動機は、このシステムに基づく係り受け解析器が、多言語に適応可能な人間の文処理の仕組みを理解するためのモデルとして価値があるかどうかを確かめることである。人間は文を処理する際、途中で三つもしくは四つの非常に小さな定数の要素しか保持できないことが報告されており [8]、これは異なる言語の話者においても成り立つものと考えられる。もし提案システムがこのような小さな定数の記憶容量のもとであらゆる言語を処理することができるのであれば、そのような主張が可能である。本稿では CoNLL-X, 2007 shared tasks [6], [25] の多言語ツリーバンクを用いてこれらの問いに答える。

本稿の貢献は以下のようにまとめられる。

(1) Left-corner 戦略に基づく、係り受け解析のための遷移

[1] 総合研究大学院大学
The Graduate University for Advanced Studies
[2] 国立情報学研究所
National Institute of Informatics
[a)] noji@nii.ac.jp
[b)] yusuke@nii.ac.jp

[*1] Hayashi らは top-down 戦略に基づく遷移システムを提案している [12] が、これは逐次的な処理を行わない。

Vol.2014-NL-216 No.1
Vol.2014-SLP-101 No.1
2014/5/22

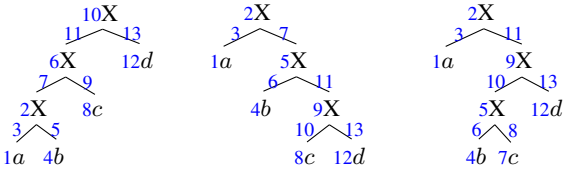
システムを構築する。ここでの技術的な貢献は、スタック上の要素として、これから来る head もしくは右側の dependent を予測するためのダミーノードを導入したことである。

(2) CoNLL shared tasks の多言語ツリーバンクを対象に、各文に対するオラクルの遷移を他の遷移システムと比較し、本システムの特徴づけを行う。これは left-corner 解析で必要な記憶容量に関する多言語を対象とした初めての分析である。

2. 構文解析と記憶容量

本稿の興味の中心は遷移システムに基づく係り受け解析で生じる記憶負荷である。心理言語学における観察から、人間は他の木構造に比べ中央埋め込み構造に対して理解の困難を示すことが知られている [1]。Left-corner 戦略[*2]はそのような性質を持つ。係り受け解析の遷移システム上での記憶負荷に関する議論は少ないが、本節では、既存のシステムはどれもこの性質を満たさないことを指摘する。そのための準備として以下ではまず、句構造解析の記憶負荷について知られている結果をまとめる。

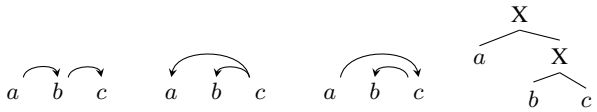
2.1 中央埋め込みと Left-corner



上に示した構造はそれぞれ左枝分かれ (left-branching)、右枝分かれ (right-branching)、中央埋め込み (center-embedding) と呼ばれる。人間は左枝分かれ及び右枝分かれの構造を容易に処理できるのに対して、中央埋め込みの構造には困難を示す。*The rat [the cat [the dog chased] bit] ate the cheese* は中央埋め込み文の例である。*The cheese was eaten [by the rat [that bit the cat [that chased the dog]]]* は同じ内容を右枝分かれの構造で言い換えた文であり、こちらのほうが容易に理解が可能となる。

Abney と Johnson は top-down 及び bottom-up 戦略はこの性質を満たさないことを指摘している [1]。例えば bottom-up 戦略で上記の右枝分かれを処理する際、c と d からなる部分木を構築するまで a 及び b が接続できないため $O(n)$ の負荷が生じるが、中央埋め込みに対する負荷はこれよりも小さくなる。それに対し、次の left-corner 戦略は中央埋め込みの難しさを正しく予測することが知られている。

[*2] 以下では構文解析の説明として戦略とアルゴリズムを区別して用いる。解析戦略は “構文木のノードとアークを認識する順番” を指定する [1]。それに対し解析アルゴリズムはこれらの戦略の具体的な実装方法を表し、典型的にプッシュダウンオートマトンを用いて定義される。以降では句構造に対する理論の説明として主に戦略を用いる。係り受け構造に対してはアルゴリズム、すなわち遷移システムによって各手法の違いを説明する。



(a) (b) (c) (d)

図 1: (a)–(c): 3 語からなる右枝分かれの係り受け構造。(d): 対応する CNF。

(1) 各ノードは、その最も左の子が認識された時点で認識される。

(2) 各アークは、それを繋ぐ二つのノードが認識された時点で認識される。

上記の各木に添えた数字はこの戦略での認識の順番を表している。これを見ると、left-corner 戦略は左枝及び右枝分かれ構造に対して生じる負荷が定数で抑えられることが分かる。例えば右枝分かれでは b を読み込んだ時点で 7 まで進むが、これはこの時点で a と b が接続されていることを示している。一方、中央埋め込み構造では b を読み込んだ時点で 6 まで進む。しかしこの時点で a と b は接続されず、余分な記憶領域が必要となる。

2.2 遷移型係り受け解析

次に、既存の遷移システムによる係り受け解析の問題点についてまとめる。これらは各文を遷移システムの上で解析する。遷移システムは状態 (configuration) の集合と状態間の遷移の集合によって規定される（詳細は [23] を参照）。各状態はその時点で構築された部分木を保持するスタックをもつ。本稿ではこのスタックの上で遷移システム毎に記憶負荷に対応する関数を定義する。

2.2.1 Arc-Standard

Arc-standard のスタック上の全ての部分木は互いに独立であるため、記憶負荷はスタックの深さとして定義するのが自然である。しかしながら、このとき arc-standard は右枝分かれの構造に対して問題を持つ。例えば $a \frown b \frown c \frown \cdots$ という構造を処理する場合を考える。Arc-standard でこの構造を処理する際、各アークを構築する前に全ての語を shift 動作でスタックに移す必要があり、$O(n)$ の負荷が生じる。Nivre はより詳細にこのシステムの問題点を論じている [22]。それによると、arc-standard は、処理する係り関係が文脈自由文法 (CFG) のチョムスキー標準形 (CNF) に変換した際に右枝分かれの構造を持つ場合に負荷が増加する性質を持つ。図 1 に 3 語からなるそのような係り関係を列挙した。これらを処理する際、システムは a を b もしくは c に接続する手前で b と c からなる部分木を構築する必要があり、余分な記憶領域が必要とされる。これはこのシステムの戦略が bottom-up、つまり各語がその head に接続される前にその語は全ての dependent を集めていなければならない性質をもつためである。実際 arc-standard は本質的に CNF に対する bottom-up 戦略に基づくプッシュダウン

	左枝分かれ	右枝分かれ	中央埋め込み
Arc-standard	$O(1)$	$O(n)$	$O(n)$
Arc-eager	$O(1)$	$O(1 \sim n)$	$O(1 \sim n)$
Left-corner	$O(1)$	$O(1)$	$O(n)$

表 1: 係り受け木の構造毎の各システムが処理に要する負荷のオーダー。$O(1 \sim n)$ は特定の構造のみ定数負荷で処理でき、他の構造には線形オーダーを要することを表す。

オートマトンと等価であり [22]、CFG に対する bottom-up アルゴリズムと同じ性質を持つ。

2.2.2 Arc-Eager

Arc-eager ではスタック中で隣接した語を接続することができるため、スタック中で独立な部分木の数によって負荷を定義する。このとき arc-eager は arc-standard での問題を一部解決する。図 1(a) 及び $a \frown b \frown c \frown \cdots$ の文を考えると、arc-eager はこれらの全ての語をスタック上で接続できるため、定数の負荷で処理を行える。これはもはや単純な bottom-up 戦略ではないため、木の形毎に生じる負荷を議論するのは困難である。しかしながら、arc-eager は全てのアークが右方向である限り負荷が増加しないという性質を持つため、ある種の中央埋め込み構造も定数の負荷で処理ができてしまう。$a \frown \overbrace{b \frown c}\ d$はそのような例であり、CNF に変換した際中央埋め込みを持つ。また arc-eager は 1(b–c) にあるような右枝分かれ構造や他の中央埋め込み構造に対して余分な負荷を生じるため、単純なオーダー記法で議論することは困難となる。これまでの結果を表 1 にまとめた。以降の目標は、中央埋め込み構造のみに対して線形のコストを要するこの表の最後の行に対応するシステムを構築することである。

2.2.3 他のシステム

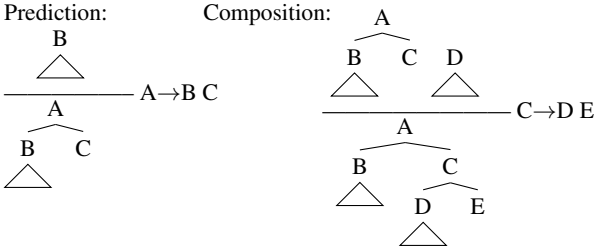
スタック上の要素を接続できない全てのシステム、例えば [20] のハイブリッドシステムなどは、その bottom-up の性質ゆえに arc-standard と同じ問題を持つ。興味深い例外として、スタック上の部分木にノードを接続する際の接続先を部分木の head に限定しない [18] や [31] が挙げられる。しかしながら、記憶負荷の観点からはこれらは arc-eager と性質を共有する。これらはいくつかの中央埋め込みを定数負荷で処理できるのに対して、図 1(b) のような右枝分かれに余分な負荷を生じてしまう。

3. Left-corner 遷移システム

本節では left-corner 戦略に基づく遷移システムを定義する。Resnik は、CFG に対してのプッシュダウンオートマトンを定義している [28] が、ここではそれをもとに、係り受け解析での対応する動作について議論する。Left-corner の核となるアイデアを係り受け解析に移植しやすくするため、以降ではまず Resnik のアルゴリズムを推論規則によって特徴づける。

3.1 Prediction と Composition

Resnik は、left-corner 解析の重要な特性は、その **composition** 動作であると述べている。ここでは CNF に対する推論規則を、別の動作である **prediction** とともに示す。



Prediction は、ある部分木が完成したときにその親と右側の兄弟ノードを予測する。**Composition** は、ノードの親と右側の兄弟を予測したのち、予測した親を直ちに他の部分木で予測された子ノードと一致させることで二つの木を結合する。この場合 C が一致したノードとなる。この動作は、完成した部分木の親となるノードが既に他の部分木の子として top-down 的に予測されている場合に使われる。

3.2 遷移システム

3.2.1 ダミーノード

ここから係り受け解析へ議論を移行する。提案システムの核となるアイデアは、さきほどの CFG における予測ノードに対応するダミーノードの導入である。解析動作についての直感を得るため、ここでは図 1(b) に対するシステムの遷移を示す。これは既存のどのシステムに対しても余分な負荷を生じさせた構造である。提案システムはまず a を shift 動作でスタックに移した後、一種の **prediction** 動作により $a \overset{x}{\leftarrow}$ という形の部分木をつくる。ここで x はダミーノードである。これは a が続く語の左側の dependent となることをこの時点で予測していることを意味する。次に b を shift 動作でスタックに移した後、**composition** 動作により二つの木を結合し $a \overset{x}{\leftarrow} b$ の木を得る。最後に c を x の位置に挿入し、解析を完了する。

3.2.2 状態

他の多くの遷移システムと同じく、提案システムは 3 つ組からなる状態 $c = (\sigma, \beta, A)$ を持つ。ここで σ はスタックであり、垂直線により要素の追加を表す。例えば $\sigma = \sigma'|\sigma_1$ は σ_1 が σ の先頭要素であることを表す。β はまだ処理していない入力単語のインデックスを保持するバッファーである。$\beta = j|\beta'$ は j が β の先頭であることを表す。最後に $A \subseteq V_w \times V_w$ は、文 w 上の単語のインデックスの集合 V_w が与えられたもとでこれまで構築されたアークの集合を保持する。

スタック上の各要素は、[18] や [31] のようにその部分木の右スピン (right spin) のみを保持する形で表現される。右スピン $\sigma_i = \langle \sigma_{i1}, \sigma_{i2}, \cdots, \sigma_{ik} \rangle$ とは、σ_i の head からその右側の子のうち最右のノードを再帰的に辿ってできるリス

SHIFT	$(\sigma, j\|\beta, A) \mapsto (\sigma\|\langle j\rangle, \beta, A)$
INSERT	$(\sigma\|\langle\sigma'_1\|i\|x(\lambda)\rangle), j\|\beta, A) \mapsto (\sigma\|\langle\sigma'_1\|i\|j\rangle, \beta, A \cup \{(i,j)\} \cup \{\cup_{k\in\lambda}(j,k)\})$
LEFT-PRED	$(\sigma\|\langle\sigma_{11}, \cdots\rangle, \beta, A) \mapsto (\sigma\|\langle x(\sigma_{11})\rangle, \beta, A)$
RIGHT-PRED	$(\sigma\|\langle\sigma_{11}, \cdots\rangle, \beta, A) \mapsto (\sigma\|\langle\sigma_{11}, x()\rangle, \beta, A)$
LEFT-COMP	$(\sigma\|\langle\sigma'_2\|x(\lambda)\rangle\|\langle\sigma_{11}, \cdots\rangle, \beta, A) \mapsto (\sigma\|\langle\sigma'_2\|x(\lambda \cup \sigma_{11})\rangle, \beta, A)$
RIGHT-COMP	$(\sigma\|\langle\sigma'_2\|x(\lambda)\rangle\|\langle\sigma_{11}, \cdots\rangle, \beta, A) \mapsto (\sigma\|\langle\sigma'_2\|\sigma_{11}\|x()\rangle, \beta, A \cup \{\cup_{k\in\lambda}(\sigma_{11}, k)\})$

図 2: Left-corner 遷移システムの各動作の定義。

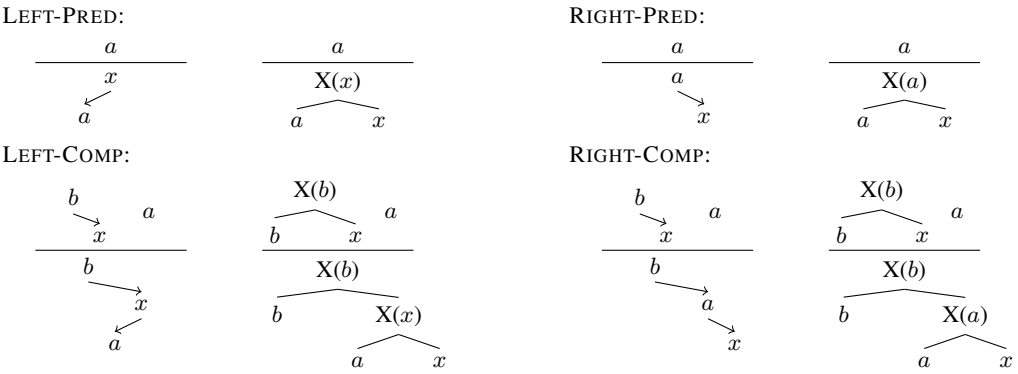
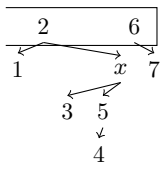


図 3: 係り受け構造と句構造での reduce 動作の対応。非終端記号 X(t) はその head となる語が t であることを表す。ここでは最も簡略化した右スピンのみを示すが、実際は x が右スピンの右端にある限り a, b 及び x はいくつでも子を持てる。

トである。$\sigma_i = \sigma'_i|\sigma_{ik}$ と書いた場合、これは σ_{ik} がスピン σ_i の右端であることを表す。σ_i の各要素は、文内の単語のインデックスもしくはダミーノード $x(\lambda)$ となる。ここで λ は x の左側の dependent の集合である。一つの状態の例を以下に示す。

スタック:



$$\sigma = [\langle 2, x(3,5)\rangle, \langle 6, 7\rangle]$$
$$\beta = [8, 9, \cdots, n]$$
$$A = \{(2,1), (5,4), (6,7)\}$$

以下では σ_i がダミーノードを持たなければ *complete*、ダミーノードを持てば *incomplete* と読んで区別する。この遷移システムは全部で六つの動作からなり、そのうち二つが shift 動作、四つが reduce 動作となる。全ての動作の定義を図 2 にまとめた。

3.2.3 Shift 動作

Shift 動作は SHIFT と INSERT[*3] の二種類が存在する。SHIFT はバッファーの先頭の語をスタック上に移動する。INSERT はこのシステム固有の動作であり、スタック先頭に存在するダミーノードをバッファーの先頭で置き換える。これはダミーノードと接続している語との間のアークを複数追加することを意味する。注意すべき点として、この動作は $x(\lambda)$ が σ_1 の head もしくは λ が空である場合も有効であり、その場合 (i,j) もしくは $\cup_{k\in\lambda}(j,k)$ のアークは追加されない。Resnik [28] はこの動作を定義していない代わりに、verification 動作を導入している (Rule 9)。提案システムの INSERT は SHIFT とこの verification との合成動作であると見なすことができる。これら shift の後、スタックの先頭要素は必ず complete になることが分かる。

[*3] SHIFT のように、small cap フォントは具体的な動作を指定する。それに対し、"shift" は動作の種類を表す。

3.2.4 Reduce 動作

Reduce 動作はスタック上の部分木に新しいアークを追加する。LEFT-PRED と RIGHT-PRED は CFG での prediction に対応している。図 3 に簡単な部分木での二つの対応を示す。LEFT-PRED はダミーノード x を a (σ_{11} に対応) の head として構築するのに対し、RIGHT-PRED では x が右側の dependent となる。動作後の部分木を CNF に変換すると、これらのアクションの違いは、a についての予測した親ノードの種類であることが分かる。LEFT-PRED は親を X(x) と予測するが、これはこの部分木の head は予測された兄弟ノード x の head であることを意味する。一方 RIGHT-PRED では head を a と予測する。CFG とは異なり、予測の際に具体的な兄弟ノードが何であるかを考える必要はないことに注意する。代わりにダミーノード x が全ての候補を抽象化する機能を持つ。似た対応関係を二つの composition 動作である RIGHT-COMP と LEFT-COMP にもとることができる。

このシステムにおいて有効な木を得るためには、shift 及び reduce 動作はそれぞれ交互に行われる必要がある。これは次のように証明できる。$c = (\sigma|\sigma_2|\sigma_1, \beta, A)$ とする。全ての reduce 動作は complete 状態の σ_1 を incomplete にするため、二度続けての reduce 動作は行えない。Shift 動作は σ_1 を complete にする。この後で、INSERT は σ_1 が incomplete であることを必要とするため行うことができず、もし SHIFT を行った場合スタック上の先頭二つが complete

となる。しかし、これらの木を繋ぐ方法は reduce 動作以外になく、reduce 動作は σ_2 が incomplete であることを必要とするため、有効な木を得ることはできない。

3.2.5 オラクルの性質

遷移システムに対するオラクルは、状態と正解のアークに対して正解の動作を返す関数である。典型的には解析器のモデルの訓練の際に用いられる [23] が、ここでは各システムの動作の違いを分析する目的で定義する。

まず、提案システムは疑似曖昧性 (spurious ambiguity) を持つことを示し、その意味を議論する。$a^\frown b^\frown c$ という文は、次の二つの動作列によって解析が可能である。

(1) SHIFT → LEFT-PRED → INSERT → RIGHT-PRED → INSERT
(2) SHIFT → LEFT-PRED → SHIFT → RIGHT-COMP → INSERT

前者はステップ 3 で b を INSERT し、その後、RIGHT-PRED により c を待つ。後者では、b は代わりに SHIFT され、続いて RIGHT-COMP により二つの部分木（$a^\frown x$ と b）が結合され、$a^\frown b^\frown x$ となる。この曖昧性は解析器の疑似曖昧性と呼ばれる。これらの解析動作の違いは、背後で認識される句構造木の形の違いに起因している。前者ではステップ 4 の RIGHT-PRED により対応する句構造として $((a\ b)\ c)$ ができあがるが、後者では RIGHT-COMP により $(a\ (b\ c))$ の句構造ができあがる。これはまた、係り受け構造から句構造への変換の際の曖昧性が原因であるとも言い換えられる。近年、これらの曖昧性を動的オラクルという手法で利用するシステムが提案されている [11], [13], [31]。本システムでも似たような分析が可能であると思われるが、これは今後の課題とし、以下では一つの静的オラクルを取り上げそれが持つ性質を議論する。

提案システムは shift と reduce 操作を交互に行うため、オラクルはそれぞれに対して定義する必要がある。$c = (\sigma|\sigma_2|\sigma_1, \beta, A)$ とする。次の shift 動作は、

- INSERT: $\sigma_1 = \langle \sigma_1'|i|x(\lambda)\rangle$ であり、$(i,j) \in A_g$ かつ j は右に dependent を持たない（i が存在する場合）または $\exists k \in \lambda; (j,k) \in A_g$（それ以外）。
- SHIFT: それ以外。

と決められる。次の reduce 動作も同じように決定できる：

- LEFTCOMP: $\sigma_2 = \langle \sigma_2'|i|x(\lambda)\rangle, \sigma_1 = \langle \sigma_{11}, \cdots\rangle$ かつ σ_{11} が β に dependent を持たず、σ_{11} が x の左の dependent となる、すなわち i の次の dependent と σ_{11} の head が一致する（i が存在する場合）もしくは $k \in \lambda$ と σ_{11} が同じ head を持つ（それ以外）。
- RIGHTCOMP: $\sigma_2 = \langle \sigma_2'|i|x(\lambda)\rangle, \sigma_1 = \langle \sigma_{11}, \cdots\rangle$ かつ σ_{11} が β にあと一つ dependent を持ち、σ_{11} が x に挿入可能、すなわち $(i, \sigma_{11}) \in A_g$（i が存在する場合）もしくは $\exists k \in \lambda; (\sigma_{11}, k) \in A_g$（それ以外）。
- RIGHTPRED: $\sigma_1 = \langle \sigma_{11}, \cdots\rangle$ かつ σ_{11} が β 一つ以上の dependent を持つ。
- LEFTPRED: それ以外。

各条件は、その動作を行った後に正解のアークを得られるかどうかを判定する。このオラクルは composition と insert を優先する戦略となっている。上で述べたように、INSERT と SHIFT はどちらも正解のアークを得るために有効な動作となることがあるが、本オラクルでは常に INSERT を選択する。Composition と prediction も同様である。

本システムの疑似曖昧性は、係り受け構造から句構造への変換が一意でない場合に発生する。本オラクルは、認識される句構造木が、ある係り受け木に対して各 head の左側の dependent から先に変換して得られるものとなる性質を持つ。例えば先ほど挙げた例では、オラクルは常に $((a\ b)\ c)$ の句構造を認識する。このことは、本オラクルは各 head に対して、左側の dependent を全て集めてから右側の dependent を集めることを示すことにより証明できる。INSERT もしくは composition が可能であるときにそれを行わないことは、ある head にまだ左向きアークが存在するのに右向きアークを構築することを意味する。また本オラクルによる遷移でスタック深さが 3 を超えるのは、係り受け木をこのように CNF に変換した際に中央埋め込みを持つ場合に限られることを確かめることができる。

4. 記憶負荷の分析

ここでは提案システムの性質を、様々な言語に対するオラクルの遷移を既存システムと比較することで特徴付ける。本分析を通じて、導入で述べた、実際に観測される文は left-corner 解析の際に小さな負荷しか必要としないという主張の言語普遍性を検証する。

4.1 設定

対象データとして、CoNLL-X 及び 2007 の shared task [6], [25] から 18 言語のツリーバンクを集めた。いくつかの言語はどちらの年度にも含まれるが、その場合は 2007 年 のデータのみを用いた。データの前処理として、まず non-projective な文もしくは root が一つもアノテートされていない文を取り除き、また各文にダミーのルート単語を追加した。このルート単語は、[3] に従い各文の最後に配置した。これは文の先頭に置く従来の方法と異なり、アノテートされたルートが一つしかない文に対する負荷がどのアルゴリズムでも変化しないためである。

ここでは arc-standard, arc-eager, left-corner の三つの遷移システムを比較する。各言語の全ての文に対して各システムでオラクルを走らせ、その記憶負荷を以下のように計測した。Arc-standard 及び left-corner はスタックの深さを負荷として用いた。この arc-standard は [22] で定義されている、reduce の際にスタック上の二つの語を繋ぐものである。Arc-eager の負荷はスタック上の接続された部分木の数とした。Arc-eager は バッファーの先頭にも部分木を構築するが、この場合は負荷に 1 を追加する。

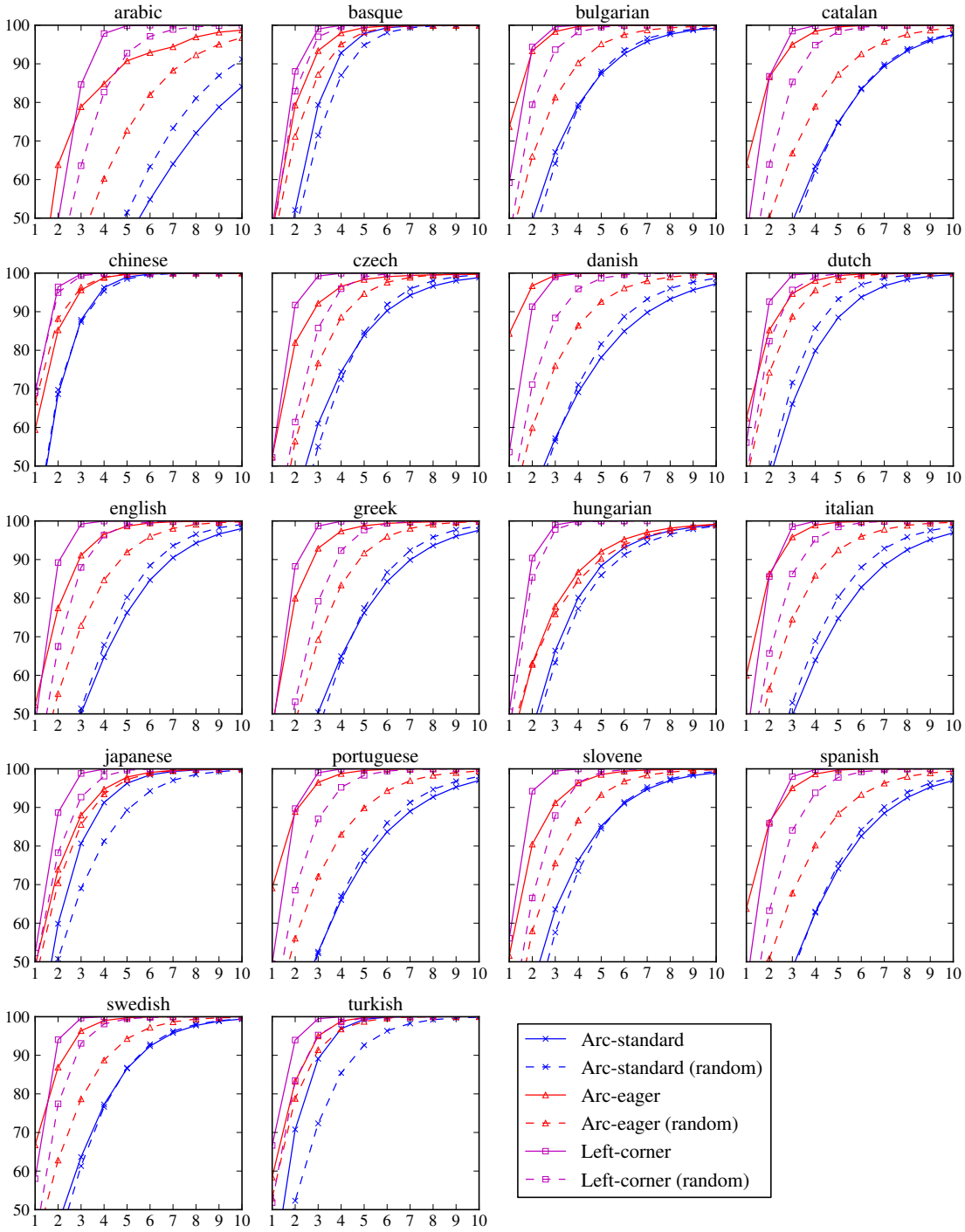


図 4: 各システムで全文に対してオラクルの状態遷移を行った際の、各記憶負荷の状態の累積頻度に関する言語間比較。例えば、Arabic では arc-eager で全文の正解の係り関係を復元する際にシステムが遷移する全状態の 9 割が、スタック上の負荷が 5 以下で抑えられることを表している。より早く 100% 付近に収束するシステムのほうが必要な負荷が小さいといえる。点線 (random) は、各文のグラフ構造と projectivity を保持したままランダムに並び替えた文での結果である。

4.2 実際の文に対する記憶負荷

オラクルを走らせる際、それぞれの記憶負荷の状態が何度現れたかを数えた。図 4 に各記憶負荷の累積頻度の言語間比較を示す。この図の実線を見ると、ある言語の全ての正解の係り関係を復元する際の全ての状態のうち、X% の状態をカバーするのに必要な負荷はどれほどか、といった問いに答えることができる。

各システムにより一つのアークを構築するのに必要な負荷が異なることから、絶対値を直接比較するのはあまり意味を持たない。例えば arc-standard はどのアークも必ずスタック上に二つ以上の要素を必要とするが、arc-eager では右向きのアークであれば一つの要素しか必要としない。そ

のためここでは、各システムの振る舞いが言語間で一致しているかどうかを中心に議論を進めることとする。

2.2 節で述べたように、arc-standard は左枝分かれの構造のみを定数の負荷で処理することができる。これらは Japanese や Turkish など head-final な言語に典型的であるが、実際に傾向が見てとれる。他の言語については、arc-standard は負荷を抑える傾向は見られない。

Arc-eager と left-corner は多くの言語で似たように振る舞う。しかしながら、left-corner は他と同じように振る舞うのに arc-eager では例外的に多くの負荷を必要とする言語がいくつか存在する（Arabic, Hungarian, Japanese など）。実際 Arabic を除いて、left-corner は 3 以下の負荷で 98% 以上の状態をカバーしている。これは、left-corner が負荷を抑えるという傾向は他のシステムに比べて言語普遍的であるという主張を支持するものである。

4.3 非文との比較

この結果は、left-corner それ自身が文の構造によらず負荷を抑えやすい傾向を持つためで、実際の文が持つバイアスを反映しているものではない、と思うかもしれない。この問いに部分的に回答するため、別の実験として非文に対してオラクルの遷移を観察し、正しい文との比較を行うことにした。この非文は [10] に習い、ツリーバンクの各係り受け木に対し (1) まずそれを有向グラフの形で取り出し、(2) 各ノードの子を projectivity を満たすようにランダムに並び替えることで作成した。図 4 の点線が、この並び替えた文での各システムの結果を表す。これを見ると、ほとんどの言語において、実際の文とランダムな文との間には大きな差が観測されている。これは、提案システムが負荷を抑えたまま解析する性質を持つのは、実際に観察される文に限られるということを示唆している。Chinese や Hungarian などの一部の言語で差は小さいが、これらの言語では他のシステムでも似た傾向が観察される。従ってこれらの言語はそのグラフ構造に何らかのバイアスが存在するのではないかと想定される。

5. 関連研究と議論

筆者らの知る限り、left-corner 戦略はこれまで句構造解析に対してしか適応されてこなかった。Roark は left-corner 文法変換 [16] を施した CFG の top-down 解析器を提案している [30]。これは本質的に left-corner 解析をしていることと等価であるが、他の種類の拡張を統一的に実現可能となっている。Roark らはこの解析器の心理学的な妥当性を検討しており、その動作が人間の reading time の観測とよく合致することを観察している [29]。Left-corner 戦略を用いた他のモデルとしては、Schuler らは、変換した英語の文の多くが限られた記憶容量で解析できることに着目し、階層型 HMM による有限の遷移モデルを提案している [32]。これらは、近年心理言語学と計算言語学の境界領域で近年注目を集めている、認知的妥当性を持つ高被覆 (broad-coverage) な解析モデル [9] の例として挙げられる。

これらとは異なり、本稿では係り受け解析が興味の対象である。認知的妥当性を持つ係り受け解析モデルは、特に多言語処理の観点から重要といえる。Keller は、現在のモデルの対象は基本的に英語、もしくは非常に少ないがドイツ語に限られており、人間の言語処理を理解するための多言語に適応可能なモデルの重要性を説いている [17]。これまでに遷移システムをもとにした人間の言語処理に関する研究はいくつか存在するため [4]、提案システムの挙動を他のシステム及び人間と比較して妥当性を検証することは今後取り組むべき課題である。

4 節で示した実験は、言語に存在する機能的バイアス、もしくは言語進化に関わる研究 [15] とも見なすことができる。計算言語学の分野では、Gildea と Temperley が似た種類の分析を行っており、彼らは *dependency length minimization* (DLM) というバイアスが、一般の文の語の並びに与えた影響を検証している [10]。これは、ある文の係り関係にはそのアークの長さの総和を小さくするようなバイアスがかかっていると主張するものである。彼らは英語とドイツ語のツリーバンクに対して、projective なランダムな並びと、長さを最小する並びとの二種類の並び替えを行い、それを元の文と比較したところ、英語は最適な並びによく合致するのに対して、ドイツ語はそれほど合致しなかったと報告している。本稿では、left-corner 解析の際の負荷を抑える傾向の言語普遍性を検証した。この分析では、Gildea と Temperley のように最適 な並びとの統計量を比較することは困難であるが、元の文の並びで、負荷が 4 以上の状態がほとんど存在しないという結果は、自然文に存在する中央埋め込みを避けるというバイアスは普遍的なものだと言えそうである。中央埋め込みの構造は典型的に長い係り関係を伴うことから、DLM と本稿の基準には何らかの相関があると考えられる。従ってこれら二つの基準をより詳細に論じることも興味深い。DLM が成り立つが、本稿の基準では説明できない、もしくはその逆となる言語は存在するだろうか。このような分析を行う際には、係り受けの定義まで含めて考えることが重要だろう。例えば、並列構造に対する係り受け関係の定義には様々なものが考えられる [27]。またこの種の機能的バイアスが、望ましい文法関係について何らかの示唆を与える可能性も考えられる。

6. まとめ

既存の係り受け解析のための遷移システムが必要とする記憶負荷が人間の観測と一致しないことを指摘し、代わりに left-corner 戦略に基づく遷移システムを提案した。多言語間の比較を通じて、人間が中央埋め込み文の生成を避ける傾向があることは言語の使用者によらず普遍的なもので

あることを確かめた。この結果はまた、提案システムが多言語を対象とした人間の言語処理の理解のために重要なモデルとなりうることを示唆する。

このような次の分析を行うためには、本システムによる解析器のモデルを構造化パーセプトロン [14], [33] などで学習する必要がある。本システムはまた、教師なし係り受け解析 [19] のためのモデルとしても興味深い。この分野では、近年様々な言語学的なバイアス、例えば reducibility [21] や音素の手がかかり [26] の有効性が示されている。近年、Cohen らが shift-reduce の動作を生成モデル的に扱う方法を示しており [7]、本システムをこのように解釈することができれば、文に存在する記憶負荷のバイアスを抽出したモデルを得ることが可能であると考えている。

最後に、係り受け構造は non-projective な構造の分析により適しているといえ、non-projective な構造を扱うための遷移システムの拡張もいくつか提案されている。本システムに対しても [2], [24] のように動作を追加することでそのような拡張が可能となれば、これまで projective な構造に閉じていた left-corner 分析の対象を、non-projective な構造にまで拡げられる可能性がある。

参考文献

[1] Abney, S. and Johnson, M.: Memory requirements and local ambiguities of parsing strategies, *Journal of Psycholinguistic Research*, Vol. 20(3), pp. 233–250 (1991).
[2] Attardi, G.: Experiments with a Multilanguage Non-Projective Dependency Parser, *CoNLL*, pp. 166–170 (2006).
[3] Ballesteros, M. and Nivre, J.: Going to the Roots of Dependency Parsing., *Computational Linguistics*, Vol. 39, No. 1, pp. 5–13 (2013).
[4] Boston, M. F. and Hale, J. T.: Garden-pathing in a statistical dependency parser, *the Midwest Computational Linguistics Colloquium* (2007).
[5] Boston, M. F., Hale, J. T., Patil, U., Kliegl, R. and Vasishth, S.: Parsing costs as predictors of reading difficulty: An evaluation using the Potsdam Sentence Corpus, *Journal of Eye Movement Research*, Vol. 2, No. 1, pp. 1–12 (2008).
[6] Buchholz, S. and Marsi, E.: CoNLL-X Shared Task on Multilingual Dependency Parsing, *CoNLL*, Association for Computational Linguistics, pp. 149–164 (2006).
[7] Cohen, S. B., Gómez-Rodríguez, C. and Satta, G.: Exact Inference for Generative Probabilistic Non-Projective Dependency Parsing, *EMNLP*, pp. 1234–1245 (2011).
[8] Cowan, N.: The Magical Number 4 in Short-term Memory: A Reconsideration of Mental Storage Capacity, *Behavioral and Brain Sciences*, Vol. 24, No. 1, pp. 87–114 (2001).
[9] Demberg, V., Keller, F. and Koller, A.: Incremental, Predictive Parsing with Psycholinguistically Motivated Tree-Adjoining Grammar., *Computational Linguistics*, Vol. 39, No. 4, pp. 1025–1066 (2013).
[10] Gildea, D. and Temperley, D.: Do Grammars Minimize Dependency Length?, *Cognitive Science*, Vol. 34, No. 2, pp. 286–310 (2010).
[11] Goldberg, Y. and Nivre, J.: Training Deterministic Parsers with Non-Deterministic Oracles., *TACL*, Vol. 1, pp. 403–414 (2013).
[12] Hayashi, K., Watanabe, T., Asahara, M. and Matsumoto, Y.: Head-driven Transition-based Parsing with Top-down Prediction, *ACL)*, pp. 657–665 (2012).
[13] Honnibal, M., Goldberg, Y. and Johnson, M.: A Non-Monotonic Arc-Eager Transition System for Dependency Parsing, *CoNLL*, pp. 163–172 (2013).
[14] Huang, L. and Sagae, K.: Dynamic Programming for Linear-Time Incremental Parsing, *ACL* (2010).
[15] Jaeger, T. and Tily, H.: On language utility: Processing complexity and communicative efficiency, *Wiley Interdisciplinary Reviews: Cognitive Science*, Vol. 2, No. 3, pp. 323–335 (2011).
[16] Johnson, M.: Finite-state Approximation of Constraint-based Grammars using Left-corner Grammar Transforms., *COLING-ACL*, pp. 619–623 (1998).
[17] Keller, F.: Cognitively Plausible Models of Human Language Processing, *ACL*, pp. 60–67 (2010).
[18] Kitagawa, K. and Tanaka-Ishii, K.: Tree-Based Deterministic Dependency Parsing — An Application to Nivre's Method —, *ACL*, pp. 189–193 (2010).
[19] Klein, D. and Manning, C.: Corpus-Based Induction of Syntactic Structure: Models of Dependency and Constituency, *ACL*, pp. 478–485 (2004).
[20] Kuhlmann, M., Gómez-Rodríguez, C. and Satta, G.: Dynamic Programming Algorithms for Transition-Based Dependency Parsers, *ACL*, pp. 673–682 (2011).
[21] Mareček, D. and Žabokrtský, Z.: Exploiting Reducibility in Unsupervised Dependency Parsing, *EMNLP*, pp. 297–307 (2012).
[22] Nivre, J.: Incrementality in Deterministic Dependency Parsing, *the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together* (2004).
[23] Nivre, J.: Algorithms for Deterministic Incremental Dependency Parsing, *Computational Linguistics*, Vol. 34, No. 4, pp. 513–553 (2008).
[24] Nivre, J.: Non-Projective Dependency Parsing in Expected Linear Time, *ACL-IJCNLP*, pp. 351–359 (2009).
[25] Nivre, J., Hall, J., Kübler, S., McDonald, R., Nilsson, J., Riedel, S. and Yuret, D.: The CoNLL 2007 Shared Task on Dependency Parsing, *the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pp. 915–932 (2007).
[26] Pate, J. K. and Goldwater, S.: Unsupervised Dependency Parsing with Acoustic Cues., *TACL*, Vol. 1, pp. 63–74 (2013).
[27] Popel, M., Mareček, D., Štěpánek, J., Zeman, D. and Žabokrtský, Z.: Coordination Structures in Dependency Treebanks, *ACL*, pp. 517–527 (2013).
[28] Resnik, P.: Left-Corner Parsing And Psychological Plausibility., *COLING*, pp. 191–197 (1992).
[29] Roark, B., Bachrach, A., Cardenas, C. and Pallier, C.: Deriving lexical and syntactic expectation-based measures for psycholinguistic modeling via incremental top-down parsing, *EMNLP*, pp. 324–333 (2009).
[30] Roark, B. E.: Robust Probabilistic Predictive Syntactic Processing: Motivations, Models, and Applications, PhD Thesis (2001).
[31] Sartorio, F., Satta, G. and Nivre, J.: A Transition-Based Dependency Parser Using a Dynamic Parsing Strategy, *ACL*, pp. 135–144 (2013).
[32] Schuler, W., AbdelRahman, S., Miller, T. and Schwartz, L.: Broad-Coverage Parsing Using Human-Like Memory Constraints., *Computational Linguistics*, Vol. 36, No. 1, pp. 1–30 (2010).
[33] Zhang, Y. and Nivre, J.: Transition-based Dependency Parsing with Rich Non-local Features, *ACL*, pp. 188–193 (2011).