2007－NL－182 (2)
2007／11／19

GLRをベースにした自然言語処理用MSLRパーザの改良

田中穂積[†1] 野呂智哉[†2] 植木正裕[†3]

東工大で開発した自然言語処理用のパーズ・システムは隣接する品詞間の接続関係を表す接続制約とCFGで書かれた日本語の文法規則の制約とを同時に組み込んだLR表を用いてGLRベースの構文解析を行うシステムである。両者の制約をLR表に組み込むことにより、制約に違反するアクションを削除する。しかしMSLRパーザでは削除しきれないアクションが存在することが知られていた。本稿ではその問題点を解消し、改良したMSLRシステム用いた構文解析を行った評価結果について述べる。

Improvement of MSLR Parser Based on GLR for Japanese Language Processing

Hozumi Tanaka ,[†1] Tomoya Noro [†2] and Masahiro Ueki[†3]

Adjacent symbol connection constraints (ASCCs) are very useful for not only the morphological analysis of non-segmenting language such as Japanese and Thai language, but also for continuous speech recognition of any language. This paper propose an algorithm to incorporate both CFG constraint and ASCCs into an LR parse table, which can be used to perform morpho-syntactic analysis through a conventional GLR parser which is called MSLR parser but is incomplete since a few of problems remained unsolved. This paper described an improved version of MSLR along with demonstrating its effectiveness by way of a concrete example.

1. はじめに

インターネットワークが社会に普及するにつれて、電子的な情報が社会の至るところに、蓄積され溢れている。我々はインターネットを介して、時と場所を選ばず、即座にそれらの情報にアクセスすることができる。情報には、画像・図形データ、音声、動画、文書などが含まれ、膨大な量に達している。「情報爆発」というキーワードのもと、わが国でも文部科学省、経済産業省が新しいプロジェクトを立ち上げ、新技術の開発に取り組み始めている。この膨大な量の情報を人手で処理することは、不可能であるといってよい。

文書情報は、その中で最も重要なものの一つであるが、文書は自然言語で書かれた部分が実質的な情報である。文書情報を機械的に処理する技術の研究、言い換えると自然言語処理技術の研究が極めて重要になっているのはそのためである。第2章では自然言語処理の2つの代表的な方法、コーパスベースとルールベースの方法を比較して論じ、ルールベース法の利点を生かした自然言語処理技術の研究を行うことが必要な時期に来ていることを説明する。そして、筆者らが東京工業大学で開発したMSLRパーザを利用する方法を説明する。これは、構文解析結果の順序付けにPGLR[2),3),5),6)]を用いるので、ルールベース法にコーパスベース法を融合したハイブリッドな方法であるといえる。

第3章では、これまでのMSLRシステムの問題点を述べ、それを改良した新しいMSLRパーザについて説明する。また、改善したMSLRパーザを用いた日本語文書処理実験とその評価結果について述べる。第4章では、結論と今後の課題について説明する。

2. コーパスベースとルールベース[3),6),7)]

自然言語処理技術は、2つに大別される。コーパス(文例を集めたもの)ベースの方法とルールベースの方法である。以下では、特に断らない限り、ルールとは文法規則のことであると考えて欲しい。最近では、コー

†1 中京大学情報理工学部
School of Information Science and Technology, Chukyo University
†2 東京工業大学大学院情報理工学研究科
Department of Computer Science, Tokyo Institute of Technology
†3 国立国語研究所
The National Institute for Japanese Language

パスベースの方法が自然言語処理技術の世界を席巻している。これは網羅性のある文法規則を開発することが困難であったことが主な要因としてあげられる[8]。これに対して、コーパスベースの方法では、そこから得られた統計データに文法規則性が反映されていると考える。コーパスの量を増やすことで、文法規則性を正確に反映させることができるはずだと考える。ところが統計データからは陽に文法規則が取り出されるわけではない。文法規則を取り出して、これをどう改良すべきかが分からない。このことは、自然言語処理の分野に関心を持っていた言語学者の多くがこの分野から離れていった理由でもあった。

文法規則は機械 (コンピュータ) で扱うことができる規則でなければならない。多種多様な分野の日本語の文書処理を行う文法規則の数は、およそ数千の規模になると言われている。ところがこのような日本語の文法規則を言語学者ですら作成したという話をまだ聞かない。これに対して、コーパスベースの方法により日本語文の文節間係り受け関係を機械的に抽出する精度 (文節係り受け精度) は 90%弱に達する。これがルールベースの方法が自然言語処理技術の中心ではなくなってきた大きな理由である。

一方ルールベースの問題は網羅性のある文法規則の開発である。このことが障害となってルールベースの方法が省みられなくなったといえよう。しかし大規模文法規則を一度構築しておけば、それを用いた構文解析を行う手法については過去の研究の蓄積があり、それが利用できる。構築した文法を修正し、より精密な文法にすれば解析精度が向上する可能性がある。

最近、コーパスベースの自然言語処理法にも解析精度に飽和現象が見られる。精度をさらに向上させようとすれば、現存するコーパスの量を 1 桁以上増やさなくてはならないといわれている。これは容易なことではない。この限界を越える技術として、ルールベースの方法を再考すべき時期に来ていると考えている。幸いにして大規模な日本語文法規則が野呂により開発されている。この文法規則を利用してルールベースの自然言語処理を試みることができる。

3. MSLR パーザ

新聞記事を対象にした大規模な日本語文法 (日本語 CFG) が野呂により開発され[8]、それにより、大規模な日本文法の存在を前提にしたルールベースの自然言語処理を行うことが可能になった。この日本語 CFG を前提にして東京工業大学で開発した、MSLR パーザ (とよぶ) 構文解析システムを用いる。MSLR パーザでは形態素解析と構文解析とを同時並行的に実行できることが特徴である。MSLR を用いた野呂文法の評価実験によれば、係り受け精度ではなく、句構造の構文解析結果の精度が 90%弱であった[*1]。

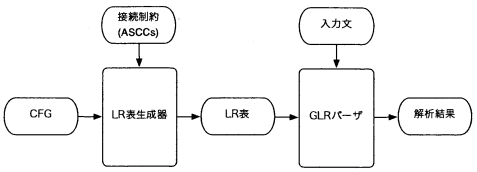

図 1 MSLR パーザ概要

第 3.1 節では、MSLR パーザのアルゴリズムの概要と問題点を述べる。

3.1 MSLR の原理

GLR 法で構文解析するとき用いる LR 表に、隣接する前終端記号 (preterminal) 間の接続制約 (隣接接続制約 (adjacent connection constraint; ASCC)) を組み込む新しい方法、MSLR 法を提案している。ASCC は形態素解析でよく使われる制約である。したがって、ASCC を組み込んだ LR 表を用いた GLR 法による構文解析を行うことにより[1),4),11)]、形態素解析と構文解析とを同時並行的に進めることができる。これを Morpho-Syntactic LR Analysis (MSLR) とよぶ[9),10)]。図 1 に MSLR パーザの処理の流れを示す。LR 表に ASCC を組み込むことにより、組み込む以前の LR 表に存在していたアクションを取り除くことが可能である。不必要なアクションは次のようにして取り除く。

構文解析は LR 表の状態 0 から出発する。状態 0 で実行可能なアクションから出発する。状態 0 で複数の先読み語があれば、それに対応した複数のアクションから出発する。アクションを 3 つ組 ⟨ 状態, アクション, 先読み語 ⟩ で表すとしよう。状態 0 の複数の先読み語を $la_{01}, \ldots la_{0i}$、それらに対応したアクションを $ACT_{01}, \ldots ACT_{0i}$ であるとする。こ

⋆1 1 文中の 10 文節すべての係り先が正しく解析できた場合の確率は p^8 となる。文節の係り先の数は 9 であり、しかも最後尾の文節は、無条件に最後の文節に掛かるので、解析すべき係り先の数は一切には 8 となる。したがって文節係り受け精度が 90%なら、$0.9^8 = 0.43$ である。構文解析の精度とは、MSLR パーザにより複数個得られる構文解析結果 (句構造) のうち、PGLR による統計的なランク付け 1 位の句構造が、正しいと判定される割合である。係り受け解析の精度と単純に比較できない。しかし、CFG の句構造の細部に至るまで完全に一致しなくてはならないので、係り受け精度以上に厳しい評価基準になっていることに注意してほしい。

こで実行するアクションがshiftアクションsh_mであれば、次に遷移する状態がmで、そこで実行するアクションが決まる。実行すべきアクションがreduceアクションre_nであれば、それを実行してLR表のgoto部を参照することにより、次に推移する状態と実行すべきアクションが決まる。

このようにして、状態0から実行すべきアクションを次々に決めてゆくと実行すべきアクションのチェインができる。このアクションのチェインが、受理(acc)アクションに至れば、解析が成功することになる。一方、実行すべきアクションがLR表から決まらないときには解析が失敗することになる。このアクションチェインは有向グラフとして表現できる。

開始状態0から受理に至るチェインは成功パスとよぶ。成功パス上のアクションは必要なアクションとしてLR表にのこす。ここで隣接するアクションが$\langle k, ACT_{0k}, la_{0k}\rangle\langle l, ACT_{0l}, la_{0l}\rangle$であるとする。$\mathrm{connect}(la_{0k}, \mathrm{la}_{0l})$が成り立たない場合は、後続するアクション$\langle k, ACT_{0k}, la_{0k}\rangle$は削除される。このアクション削除法の欠点は、成功アクション上のアクションを取り出すときにチェインがループする可能性があることである。これを除いた改良版のアルゴリズムは次節で説明する。

3.2 改良アルゴリズム

成功パスを取り出すときに縦型探索アルゴリズムを採用してはならない。有向グラフに沿ってグラフを辿るときにループに嵌って先に進めなくなり最終状態のaccに到達することができないことがあるからである。したがって成功パスの探索(チェインの探索)には横型探索を使う。新しいMSLRに組み込んだLRパーズLR表からのアクション削除アルゴリズムでは、チェインの最終状態accからチェインを有向グラフとは逆向きにたどる。こうすることで初期状態からチェインを辿った場合に中途でチェインが途切れて、それが無駄な探索に終わるケースを避けることができて効率的な成功パスの検出ができるのではないかと考えられる。

改良アルゴリズムの概要を図2に示す。図中の記法については、以下のとおりである。

ReduceChain($zeros, rule, la, status$): 状態 $zeros$、先読み la において規則 $rule$ でreduceするチェイン。$status$はチェインの処理状態を表す。チェインの処理状態には、setup(作成直後)、start(展開前)、set(展開後)、pass(通過後)、end(最終状態)がある。

ShiftChain($zeros, la, status$): 状態 $zeros$、先読み la においてshiftするチェイン。

PrevChain(c): reduceチェインまたはshiftチェイン c に先行するチェインの集合。

Rule(A): 非終端記号 A を左辺に持つ規則の集合。

Length($rule$): 規則 $rule$ の右辺の長さ。

RHS($rule, i$): 規則 $rule$ の右辺の i 番目の(非)終端記号。$1 \leq i \leq \mathrm{Length}(rule)$

State($zeros, rule, i$): 状態 $zeros$ から 規則 $rule$ について、$\mathrm{RHS}(rule, 1), \ldots \mathrm{RHS}(rule, i-1)$ を遷移した後の状態。

La($state, rule$): 状態 $state$ から規則 $rule$ の右辺を遷移した状態でのreduceアイテムの先読みの集合。

PrevSym($state$): 状態 $state$ への遷移記号の集合。

PrevState($state, sym$): 記号 sym によって状態 $state$ に遷移する状態の集合。

図3に示すCFGと接続制約に対し、上述のアルゴリズムを適用すると、以下のような手順で処理が進行する。

(1) $\mathrm{ReduceChain}(0, 0, \$, \text{“start”})$ を作成。

(2) $\mathrm{ReduceChain}(0, 0, \$, \text{“start”})$ について、
$\mathrm{ReduceChain}(0, 1, \$, \text{“start”})$ を作成。

(3) $\mathrm{ReduceChain}(0, 1, \$, \text{“start”})$ について、
$\mathrm{ReduceChain}(0, 2, b, \text{“setup”})$、
$\mathrm{ReduceChain}(0, 2, c, \text{“setup”})$、
$\mathrm{ReduceChain}(2, 5, \$, \text{“start”})$、
$\mathrm{ReduceChain}(2, 6, \$, \text{“start”})$ を作成。

(4) $\mathrm{ReduceChain}(2, 5, \$, \text{“start”})$ について、
$\mathrm{connect}(b, \$) = 1$ より
$\mathrm{ShiftChain}(2, b, \text{“start”})$ を作成。

(5) $\mathrm{ReduceChain}(2, 6, \$, \text{“start”})$ について、
$\mathrm{connect}(c, \$) = 0$

(6) $\mathrm{ShiftChain}(2, b, \text{“start”})$ について、
$\mathrm{ReduceChain}(0, 2, b, \text{“setup”})$ の処理状態をstartに変更。

(7) $\mathrm{ReduceChain}(0, 2, b, \text{“start”})$ について、
$\mathrm{ReduceChain}(0, 3, b, \text{“start”})$、
$\mathrm{ReduceChain}(0, 4, b, \text{“start”})$ を作成。

(8) $\mathrm{ReduceChain}(0, 3, b, \text{“start”})$ について、
$\mathrm{ReduceChain}(0, 3, c, \text{“setup”})$、
$\mathrm{ReduceChain}(0, 4, c, \text{“setup”})$、
$\mathrm{ReduceChain}(3, 5, b, \text{“start”})$、
$\mathrm{ReduceChain}(3, 6, b, \text{“start”})$ を作成。

(9) $\mathrm{ReduceChain}(0, 4, b, \text{“start”})$ について、
$\mathrm{connect}(a, b) = 1$ より
$\mathrm{ShiftChain}(0, 1, \text{“start”})$ を作成

(1) ReduceChain$(0, 0, \$, \text{start})$ を作成し (状態 0 は開始状態、規則 0 は最初に展開される規則を表す)、PrevChain(ReduceChain$(0, 0, \$, \text{start})) = \emptyset$ とする。

(2) 処理状態が start の reduce チェインまたは shift チェインがあれば、その中から 1 つを横型探索で選択し (これを c_{start} とする)、処理状態を set として以下の処理を行う。

- $c_{\text{start}} = \text{ReduceChain}(zeros, rule, la, \text{“set”})$ の場合
  - $1 \leq i \leq \text{Length}(rule) - 1$ である各 i について
    - RHS$(rule, i)$ が終端記号の場合、
      (a) $c = \text{ShiftChain}(\text{State}(zeros, rule, i), \text{RHS}(rule, i), \text{“setup”})$ を生成。
      (b) PrevChain$(c) = \{c_{\text{start}}\}$ とする。
    - RHS$(rule, i)$ が非終端記号の場合、各 $r \in \text{Rule}(\text{RHS}(rule, i))$、各 $l \in \text{La}(\text{State}(zeros, rule, i), r)$ について、
      (a) $c = \text{ReduceChain}(\text{State}(zeros, rule, i), r, l, \text{“setup”})$ を生成。
      (b) PrevChain$(c) = \{c_{\text{start}}\}$ とする。
  - $i = \text{Length}(rule)$ の場合
    - RHS$(rule, i)$ が終端記号の場合、connect$(\text{RHS}(rule, i), la) = 1$ ならば、
      (a) $c = \text{ShiftChain}(\text{State}(zeros, rule, i), \text{RHS}(rule, i), \text{“start”})$ を生成。
      (b) PrevChain$(c) = \{c_{\text{start}}\}$ とする。
    - RHS$(rule, i)$ が非終端記号の場合、各 $r \in \text{Rule}(\text{RHS}(rule, i))$ について、
      (a) $c = \text{ReduceChain}(\text{State}(zeros, rule, i), r, la, \text{“start”})$ を生成。
      (b) PrevChain$(c) = \{c_{\text{start}}\}$ とする。
- $c_{\text{start}} = \text{ShiftChain}(zeros, la, \text{“set”})$ の場合
  - $zeros = 0$ の場合、処理状態を pass とする。
  - $zeros \neq 0$ の場合、各 $sym \in \text{PrevSym}(zeros)$ について
    - sym が終端記号の場合、connect$(sym, la) = 1$ ならば、
      各 $curs \in \text{PrevState}(zeros, sym)$ について、
      (a) $c = \text{ShiftChain}(curs, sym, \text{“start”})$ を作成。
      (b) PrevChain$(c) = \{c_{\text{start}}\}$ とする。
    - sym が非終端記号の場合、
      各 $curs \in \text{PrevState}(zeros, sym)$、各 $r \in \text{Rule}(sym)$ について、
      $c = \text{ReduceChain}(curs, r, la, \text{“setup”})$ があれば、
      (a) 処理状態を start とする。
      (b) PrevChain$(c) := \text{PrevChain}(c) \cup \{c_{\text{start}}\}$ とする。

(3) 処理状態が start の reduce チェインまたは shift チェインがあれば、(2) へ戻る。

(4) 処理状態が pass の reduce チェインまたは shift チェインがあれば、その中から 1 つを選択し (これを c_{pass} とする)、以下の処理を行う。
(a) c_{pass} の処理状態を end とする。
(b) 各 $c \in \text{PrevChain}(c_{\text{pass}})$ について、以下の処理を行う。

- $c = \text{ReduceChain}(zeros, rule, la, \text{“set”})$ の場合、$i = Length(rule)$ について、
  - RHS$(rule, i)$ が終端記号の場合、connect$(\text{RHS}(rule, i), la) = 1$、
    $\text{ShiftChain}(\text{State}(zeros, rule, i), \text{RHS}(rule, i), \text{“end”})$ があれば、
    c の処理状態を pass にする。
  - RHS$(rule, i)$ が非終端記号の場合、各 $r \in \text{Rule}(\text{RHS}(rule, i))$ について、
    $\text{ReduceChain}(\text{State}(zeros, rule, i), r, la, \text{“end”})$ があれば、
    c の処理状態を pass にする。
- $c = \text{ShiftChain}(zeros, la, \text{“set”})$ の場合、
  各 $sym \in \text{PrevSym}(zeros)$ について
  - sym が終端記号の場合、connect$(sym, la) = 1$ ならば、
    各 $curs \in \text{PrevState}(zeros, sym)$ について、
    $\text{ShiftChain}(curs, sym, \text{“end”})$ があれば、c の処理状態を pass にする。
  - sym が非終端記号の場合、
    各 $curs \in \text{PrevState}(zeros, sym)$、各 $r \in \text{Rule}(Sym)$ について、
    $\text{ReduceChain}(curs, r, la, \text{“end”})$ があれば、c の処理状態を pass にする。

(5) 処理状態が pass の reduce チェインまたは shift チェインがあれば、(4) へ戻る。

図 2 改良アルゴリズム概略

(10) ReduceChain$(3, 5, b, \text{“start”})$ について、
connect$(b, b) = 0$

(11) ReduceChain$(3, 6, b, \text{“start”})$ について、
connect$(c, b) = 0$

(12) ShiftChain$(0, 1, \text{“start”})$ について、
処理状態を pass に変更。

(13) ShiftChain$(0, 1, \text{“pass”})$ について、
ReduceChain$(0, 4, b, \text{“setup”})$ の処理状態を pass に変更。

(14) ReduceChain$(0, 4, b, \text{“pass”})$ について、

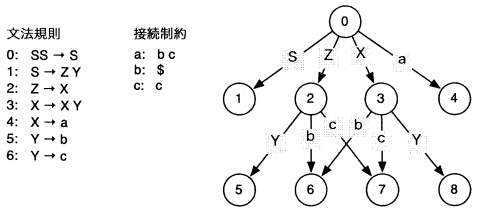

図 3 CFG、接続制約と goto グラフ

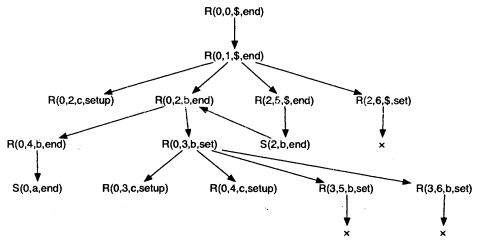

図 4 終了時のチェインのグラフ

ReduceChain$(0, 2, b,$ "setup") の処理状態を pass に変更。

(15) ReduceChain$(0, 2, b,$ "pass") について、ShiftChain$(2, b,$ "setup") の処理状態を pass に変更。

(16) ShiftChain$(2, b,$ "pass") について、ReduceChain$(2, 5, \$,$ "setup") の処理状態を pass に変更。

(17) ReduceChain$(2, 5, \$,$ "pass") について、ReduceChain$(0, 1, \$,$ "setup") の処理状態を pass に変更。

(18) ReduceChain$(0, 1, \$,$ "pass") について、ReduceChain$(0, 0, \$,$ "setup") の処理状態を pass に変更。

終了時のチェインのグラフを図 4 に、作成される LR 表を表 1 に示す。ただし、図 4 において、"R" は ReduceChain を、"S" は ShiftChain を表す。また、表 1 において、括弧で囲まれたアクションは、提案手法により削除されるものである。

3.3 実験と評価

提案手法の効果を調べるため、従来手法との比較実験を行った。コーパスは東工大コーパス 20,190 文[8] を利用する。CFG は 20,190 文すべてから抽出し、MSLR パーザで構文解析を行う。ただし、入力は品詞列とする。解析結果の順位付けは確率一般化 LR (PGLR) モデルにより行う。比較は、提案手法により生成される

表 1 作成される LR 表

	a	b	c	$	S	X	Y	Z
0	sh4				1	3		2
1				acc				
2		sh6	(sh7)				5	
3		(sh6)/ re2	(sh7)/ (re2)				(8)	
4		re4	(re4)					
5				re1				
6		(re5)	(re5)	re5				
7		(re6)	(re6)	(re6)				
8				(re3)				

表 2 各 LR 表中の各アクション数

	提案手法	従来手法	
		接続制約有	接続制約無
shift	38,020	38,355	77,900
reduce	213,163	230,683	747,580
goto	41,208	41,379	50,799
accept	1	1	1
合計	292,392	310,418	876,281

LR 表と、MSLR パーザが持つ LR 表生成機能により生成される LR 表で行う。MSLR パーザが持つ LR 表生成機能は、一般的な LALR 表を生成するだけでなく、品詞間の接続制約を反映させることもできる。今回の実験では、接続制約を反映させた LR 表と反映させなかった LR 表の両方を比較対象とする。

人手で抽出した CFG 規則数は 2,722 規則 (非終端記号 294 個、終端記号 412 個) である。各手法により生成された LR 表中の各アクションの数を表 1 に示す。この表より、接続制約を組み込まない場合と比較して約 67%、組み込んだ場合と比較して約 6%、アクション数を削減できていることが分かる。

次に、PGLR モデルによる順位付けの結果を見る。モデルの学習には、文法抽出に利用した 20,190 文すべてを利用する。解析精度は、文正解率により比較する。文正解率は以下のように定義する。

$$\text{文正解率} = \frac{\text{上位 } n \text{ 位までに正解が含まれる文の数}}{\text{解析した文の総数}}$$

ここで「正解」とは、出力された解析木が正解とすべき構文木と完全に一致する場合を指す。結果を表 3 に示す。これより、各順位における文正解率はほとんど変わらないことが分かる。PGLR モデルによる順位が 1 位の解析木のみを見た場合で 0.04%向上しただけであり、順位によっては逆に文正解率が悪化しているところもある。実際、接続制約を組み込んだ場合の結果と比較して、順位に変動があった文は 20,190 文中

表 3　各順位における文正解率

順位 (n)	1	2	3	4	5
提案手法	58.66	72.55	78.40	82.13	84.68
制約有	58.62	72.60	78.42	82.13	84.69
制約無	58.62	72.60	78.42	82.14	84.69
順位 (n)	6	7	8	9	10
提案手法	86.39	87.78	88.94	89.62	90.46
制約有	86.43	87.84	88.85	89.64	90.49
制約無	86.43	87.85	88.86	89.65	90.50
順位 (n)	20	30	40	50	60
提案手法	93.77	95.22	96.13	96.69	97.12
制約有	93.78	95.22	96.14	96.69	97.12
制約無	93.78	95.22	96.14	96.69	97.12
順位 (n)	70	80	90	100	
提案手法	97.48	97.71	97.87	98.05	
制約有	97.48	97.72	97.88	98.05	
制約無	97.48	97.72	97.88	98.05	

表 4　構文解析所要時間 (ユーザ CPU 時間)

	提案手法	従来手法	
		接続制約無	接続制約有
所要時間 (秒)	102.4	113.4	119.2

421 文だけで、そのうち 343 文は順位が 1 つ上昇、または下降しただけであった (2 位以下から 1 位へ上昇した文は 47 文、1 位から 2 位以下へ下降した文は 40 文であった)。

最後に、全 20,190 文を構文解析する際の所要時間 (ユーザ CPU 時間) を計測した。結果を表 4 に示す。ただし、計測は Dual-Core Intel Xeon 3GHz、メモリ 4GB の環境で行った。結果より、接続制約を組み込まない場合と比較して 14%程度、組み込んだ場合と比較して 10%程度、時間が短縮されている。提案手法による解析精度の向上は見られなかったが、解析速度が向上することが分かる。

4. 結論と今後の課題

改良した MSLR パーザによる構文解析の速度は表 4 によると確かに向上している。LR 表に残されたアクションの数も表 2 によると減少している。LR 表のサイズも接続制約を組み込んだ場合よりさらに 6%程アクション数が減少し、LR 表に付与される確率が変わるので、PGLR の構文解析結果の順位付けが変わる。上位 10 位までに正解が含まれる確率は 90%程度であり高確率である。この結果に共起データの情報を加えればコーパスベースの自然言語処理と同程度の正解率が得られるものと期待される。筆者らはルールベースの自然言語処理にまだまだ検討の余地があると考えている。

ルールベースの自然言語処理の問題は大規模日本語 CFG の開発である。本稿では野呂の開発した日本語文法を用いた。これは人手で開発された労力とコストを要するものである。これに対して筆者らは大規模日本語 CFG の開発に取り組んでいる。3 万の新聞記事データに対して人手で依存構造が付与された京大コーパスから、機械的に CFG を抽出する実験を行っている。その結果はいずれどこかの研究会で発表する機会を持ちたいと考えている。

参 考 文 献

1) Aho, A.V., Sethi, R. and Ullman, J.D.: *Compilers: Principles, Techniques, and Tools*, Addison Wesley (1986).
2) Briscoe, T. and Carroll, J.: Generalized Probabilistic LR Parsing of Natural Language (Corpora) with Unification-Based Grammars, *Computational Linguistics*, Vol.19, No.1, pp.25–59 (1993).
3) Charniak, E.: *Statistical Language Learning*, The MIT Press (1996).
4) DeRemer, F. and Pennello, T.: Efficient Computation of LALR(1) Look-Ahead Sets, *ACM Transactions on Programming Languages and Systems*, Vol.4, No.4, pp.615–649 (1982).
5) Inui, K., Sornlertlamvanich, V., Tanaka, H. and Tokunaga, T.: Probabilistic GLR Parsing, *Advances in Probabilistic and Other Parsing Technologies* (Bunt, H. and Nijholt, A., eds.), Kluwer Academic Publishers, chapter5, pp.85–104 (2000).
6) Jelinek, F.: *Statistical Methods for Speech Recognition*, The MIT Press (1998).
7) Manning, C.D. and Schütze, H.: *Foundations of Statistical Natural Language Processing*, The MIT Press (1999).
8) Noro, T., Koike, C., Hashimoto, T., Tokunaga, T. and Tanaka, H.: Evaluation for a Japanese CFG Grammar Derived from Syntactically Annotated Corpus with Respect to Dependency Measures, *the 5th Workshop on Asian Language Resources*, pp.9–16 (2005).
9) 白井清昭，植木正裕，橋本泰一，徳永健伸，田中穂積：自然言語解析のための MSLR パーザ・ツールキット，自然言語処理， Vol.7, No.5, pp. 93–112 (2000).
10) Tanaka, H., Tokunaga, T. and Aizawa, M.: Integration of Morphological and Syntactic Analysis Based on LR Parsing Algorithm, *the 3rd International Workshop on Parsing Technologies*, pp.101–109 (1993).
11) Tomita, M.: *Efficient Parsing for Natural Language*, Kluwer Academic Publishers (1986).