

YAGLR法: Yet Another Generalized LR Parser

田中穂積, K. G. スレッシュ

東京工業大学・工学部

プログラム言語などの人工言語のコンパイラでは、Knuthの考案したLR(k)法とよぶ統語解析アルゴリズムが使われることがある。これは先読み情報を用いて決定的に統語解析を進め、無駄なく効率的に統語解析を行うことができる。ただしそこで使われる文法は、LR文法に限られており、自然言語の解析に用いる一般の文脈自由文法(CFG)を扱うことができない。

富田はLR(k)法の持つ利点をそのまま保持し、しかも一般のCFGが扱えるようにLR(k)法を拡張している。これは一般化LR法(generalized LR parsing algorithm; 拡張LR法)の1つであり、富田法とよばれている。経験的に富田法はアーリー法と比べて高速な統語解析が可能であるが、富田法に対して、アーリー法以上のオーダの解析時間を要す特殊なCFGの存在が知られている。

我々は、アーリー法の利点と富田法の利点をそのまま引き継ぐ新しい一般化LR法(これをYAGLR法とよぶ)を提案する。それによれば、先に述べた特殊なCFGに対する解析時間のオーダを n^3 に抑えることができる。また、富田法ではグラフ構造化スタック(GSS)を用いて、使用記憶空間の節約を図っているが、GSSの構造は必ずしも単純でなく、実装が容易ではない。これに対してYAGLR法では木構造化スタックを用いており、経験的に、富田法のGSSと比較して同程度の使用記憶空間であり、構造が単純であるため扱い易い。これは、YAGLR法では富田法以上にスタックのマージが可能になるためである。本論文では、YAGLR法による統語解析アルゴリズムを与え、富田法との比較を具体例により説明し、YAGLR法が高速な統語解析法であることを実証する。

YAGLR: Yet Another Generalized LR Parser

Hozumi TANAKA, K. G. SURESH

Department of Computer Science
Tokyo Institute of Technology

2-12-1 Oookayama Meguro-ku Tokyo 152, Japan

We have developed a new generalized LR parsing algorithm called YAGLR. The parsing algorithm uses a set of tree-structured stacks which is compact as well as graph-structured stack of Tomita's algorithm. The reason is that YAGLR does effective merge operations on the tree-structured stack. We also present the experimental results from which we can reveal that the parsing time of YAGLR is faster than that of Tomita's. Even for the special CFGs for which Tomita's algorithm does not fare well, YAGLR's parsing time is in the order of $O(n^3)$. We conclude that YAGLR has both advantages of Earley's and Tomita's algorithm.

1. はじめに

プログラム言語などの人工言語のコンパイラでは、Knuthの考案したLR(k)法とよぶ統語解析アルゴリズム[Knuth 65]が使われることがある。これは先読み情報を用いて決定的に統語解析を進め、無駄なく効率的に統語解析を行うことができる。ただしそこで使われる文法は、LR文法に限られており、自然言語の解析に用いる一般の文脈自由文法(CFG)を扱うことができない。

富田はLR(k)法の持つ利点をそのまま保持し、しかも一般のCFGが扱えるようにLR(k)法を拡張している[Tomita 86, 87]。これは一般化LR法(generalized LR parsing algorithm; 拡張LR法)の1つであり、富田法とよばれている。経験的に富田法はアークリー法と比べて高速な統語解析が可能であるが[Tomita 86]、富田法に対して、解析すべき文の長さを n としたとき n^3 以上のオーダの解析時間を要す特殊なCFGの存在が知られている[Johnson 89][Kipps 89]。一方、アークリー法は、いかなるCFGに対しても、解析時間のオーダが n^3 であるから、この特殊なCFGに対して、理論的に、富田法はアークリー法以上の解析時間を要することになる。

本稿で提案する新しい一般化LR法では、アークリー法の利点と富田法の利点をそのまま引き継ぐ(付録B参照)。この新しい一般化LR法をYAGLR法(Yet Another Generalized LR parsing)とよんでいる。YAGLR法によれば、先に述べた特殊なCFGに対する解析時間のオーダを n^3 に抑えることができる(付録A参照)。また、富田法ではグラフ構造化スタック(GSS)を用いて、使用記憶空間の節約を図っているが、GSSの構造は必ずしも単純でなく、実装が容易ではない。これに対してYAGLR法では木構造化スタック(trss)を用いるが、経験的に、富田法のGSSと比較して使用記憶空間は同程度であり、構造が単純であるため扱い易い。これは、YAGLR法では富田法以上にtrssのマージが可能になるためである。

YAGLR法では、解析中に統語解析木は作らず、統語解析木を構成する部品のみを作成する。この部品を逆ドット項と呼ぶが、これはアークリー法で作成するドット項(アークリー項)と対称な構造をしている。2章では、アークリー項と逆ドット項を作成するための木構造化スタックを説明し、逆ドット項を作成することにより、木構造化スタックのより深いマージが可能になることを説明する。逆ドット項を作成するその他の利点についても述べる。3章では、逆ドット項の形式的な定義を与える。4章ではYAGLR法の鍵となるtrssのマージ操作(統合化操作)、シフト操作、レデュース操作を説明し、最後にYAGLR法の統語解析アルゴリズムを説明する。5章では、富田法とYAGLR法による解析過程をトレースする。6章では、実験結果を示しYAGLR法の高速度を実証する。7章ではまとめと今後の研究課題について述べる。

2. アークリー項と逆ドット項

本章では、スタックの節点に位置番号をもたせることにより、レデュース操作時にアークリー項と、逆ドット項を作ることができることを説明する。逆ドット項はYAGLR法で用いる項で、アークリー項と対称な形をしている。アークリー項を作る場合は、富田法以上のスタックのマージが可能になるが、逆ドット項を作る場合は、アークリー項を作る以上のスタックのマージが可能なることを説明する。

さて、入力文 W が $W=W_1W_2\ldots W_n$ であるとしよう。ここで単語 W_i と W_{i+1} の間に番号 i をふり、これを位置番号と呼ぶことにする。スタックの節点に、部分解析木の代わりにこの位置番号をもたせることにより、レデュース操作時にアークリー項と逆ドット項を作ることができる。(a)に示すスタックを考えよう。

(a) $\cdots - [\{3, s_3\} - [\{5, s_2\} - [\{6, s_1\}]]]$ (top) re, x

ここで s_1, s_2, s_3 は状態を、 $\{6\}, \{5\}, \{3\}$ は位置番号の集合を表す。これらの位置番号は、これらを含む節点の直接右に振られていると考える。従って、節点 $\{6\}, s_1$ には入力文の w_6 が対応し、節点 $\{5\}, s_2$ には w_4w_5 が対応する。以上のことから、

(a)のレデュース操作"re, x"で用いる規則"x"が、 $A \rightarrow B \ C$ であるとすると、このレデュース操作によりトップから二つの節点をポップし、(b)に示すアークリー項を作ることができることが分かる。

(b) アークリー項: $1_5 \ni [A \rightarrow B \cdot C, 3], 1_6 \ni [A \rightarrow B \ C \cdot, 3]$

最初のアークリー項の内部にある数字3と・記号は、位置番号3から5の間にある入力文の部分(w_4w_5)がBとして認識されたことを表す。同様に、二番目のアークリー項は位置番号3から6の間にある入力文の部分($w_4w_5w_6$)が"B C"として認識され、それがAにまとめられることを意味している。一方、(a)から(c)に示す逆ドット項を作ることでもできる。

(c) 逆ドット項: $1_5 \ni [A \rightarrow B \cdot C, 6], 1_3 \ni [A \rightarrow \cdot B \ C, 6]$

(c)の項中の数字の6は、スタックトップの節点の位置番号である。最初の逆ドット項は、アークリー項とは逆に、位置番号6から5の間にある入力文の部分(w_6)がCとして認識されていることを、また二番目の逆ドット項は、位置番号6から3の間にある入力文の部分($w_4w_5w_6$)が"B C"として認識され、それがAにまとめられることを意味している。この様に、逆ドット項では、アークリー項とは逆にCFGの右辺の右端からドットの位置までが解析済みであると見なす。逆ドット項はアークリー項と対称な項なのである。

ここで、(d)に示す木構造化スタックを考えよう。これは富田法と同様、スタックトップの節点の状態が s_1 で同一であるので、以後の処理を共通して行うためにトップを一本化している。

(d) $\cdots - [\{3, s_3\} - [\{5, s_2\} - [\{6, s_1\}]]]$ (top) re, x
 $=== [\{2, s_4\} - [\{4, s_2\} - [\{6, s_1\}]]]$

先の規則"x"により、"re, x"を行い、(e)と(f)に示すアークリー項と逆ドット項を作ることができる。

(e) アークリー項: $1_5 \ni [A \rightarrow B \cdot C, 3], 1_6 \ni [A \rightarrow B \ C \cdot, 3], 1_4 \ni [A \rightarrow B \cdot C, 2], 1_8 \ni [A \rightarrow B \ C \cdot, 2]$

(f) 逆ドット項: $1_5 \ni [A \rightarrow B \cdot C, 6], 1_3 \ni [A \rightarrow \cdot B \ C, 6], 1_4 \ni [A \rightarrow B \cdot C, 6], 1_2 \ni [A \rightarrow \cdot B \ C, 6]$

(d)の代わりに、状態 s_1 を持つ二つのトップの節点を一つにマージした(g)を考えてみよう。

(g) $\cdots - [\{3, s_3\} - [\{5, s_2\} - [\{6, s_1\}]]]$ (top) re, x
 $=== [\{2, s_4\} - [\{4, s_2\}]]$

(g)に示すスタック節点のマージは富田法では許されないことに注意しよう。一段深いマージを行った(g)からも、(e)と(f)に示すアークリー項と逆ドット項が作れることは明かである。スタックトップの次の節点も同一の状態 s_2 を持つので、更に一段深くマージを進めた(h)を考えてみよう。ただし、マージ後の節点の位置番号の集合は、マージ前の節点の持つ位置番号の集合の和とする。

(h) $\dots - [\{3, s_3\}] - [\{4, 5, s_2\}] - [\{6, s_1\}] \text{ (top) } re, x$
 $==== [\{2, s_4\}]$

すると、(h) に "re, x" を施した結果得られるアーリー項には (e) に示すもの以外に、次の二つの余分な項が含まれる：
 $l_4 \ni [A \rightarrow B \cdot C, 3]$ と $l_5 \ni [A \rightarrow B C \cdot, 2]$ である。しかし幸いなことに、(h) から得られる逆ドット項は (f) と全く同一である。これは LR 法が最右導出をベースにしていることによるのだが、ここではそれを指摘しておくに留める。いずれにしても逆ドット項を作成することにより、木構造化スタックの深いマージが可能になり、スタックの構造が簡単になり記憶の節約になるだけでなく、スタック操作がグラフ構造化スタックと比べて簡便になるという利点がある。YAGLR 法で、アーリー項の代わりに逆ドット項を作成する最大の理由はここにある。言うまでもなく、このようなスタックの深いマージは、富田法では不可能である。

最後に (e) に示す逆ドット項の内部の数字は全て 6 であることに注意したい。これはスタックトップの節点の位置番号であり、次のシフト操作が行われるまで変わらないので (4. 1. 4 参照)。重複した逆ドット項の検査は、現在のシフト操作から次のシフト操作が行われるまでの間のレデュース操作中に作られた逆ドット項を調べるだけでよい。言い換えると、調べる範囲を局所化することが可能になる。これが逆ドット項作成の第 2 の利点である。

実際木構造化スタックを用いた富田法と比べて、解析結果の曖昧性が増えるにつれて急激に高速な統語解析が可能であることが実証されている (6 章)。

3. 逆ドット項(Dot Reverse Item)

本章では、YAGLR 法による統語解析で重要な役割を果たす逆ドット項を説明する。富田法がレデュース操作時に部分統語解析結果を圧縮統語森として作成するのに対して、YAGLR 法ではアーリー項と対称な逆ドット項の集合を作成する。逆ドット項の定義を以下に与える。

- (1) 解析対象文 $w = w_1 w_2 \dots w_n \in T^*$ であるとする。ここに T^* は、終端記号の集合 T の要素を 0 個以上並べたものを表す。終端記号 (語と考えてよい) w_i と w_{i+1} の間に番号 i を振り、これを位置番号とよぶ (i を、語 w_i の位置番号とよぶ)。ただし w_1 の左には位置番号 0 を、また w_n の右には位置番号 n を振る。
- (2) CFG 規則 $A \rightarrow \alpha$ があり、 α が $\beta \gamma$ と書けるとき、 $0 \leq j \leq n$ なる j に対して、 $[A \rightarrow \beta \cdot \gamma, j]$ は、 w に対する逆ドット項である。特に、 $[A \rightarrow \cdot \alpha, j]$ または $[A \rightarrow \alpha \cdot, j]$ も w に対する逆ドット項である。
- (3) 逆ドット項の集合 I_i を次のように定義する。 $0 \leq i \leq j \leq n$ なる i と j に対して、
 $[A \rightarrow \alpha \cdot \beta, j] \in I_i$ iff $S \Rightarrow \gamma A \delta$, $\beta = w_{i+1} w_{i+2} \dots w_j$, and $\delta = w_{j+1} w_{j+2} \dots w_n$

アーリー項も逆ドット項も、ドットの位置は項の集合 I の添字 i として表す。アーリー項と逆ドット項の違いは、項中の位置番号 j の解釈にある。上記した (3) の定義から明らかなように、逆ドット項では、ドット記号の右の部分が β として解析済みであるとみなす。項内の CFG 規則の右辺の直後の位置番号が j であり、そこから先頭方向に i まで逆戻りした部分 $w_{i+1} w_{i+2} \dots w_j$ が β として解析済みであると解釈する。ちなみにアーリー項では解析対象文の α が解析済みであるとされていた。

4 YAGLR 法

本章では木構造化スタック (以下では $trss$ と呼ぶ) の構造と、 $trss$ 同士のマージ操作、 $trss$ に対するシフト操作とレデュース操作を説明し、YAGLR 法の手続きを示す。

4. 1 マージ, シフト, レデュース

4. 1. 1 識別子付き位置番号

YAGLR 法で用いる $trss$ の各節点には、レデュース操作時に逆ドット項、 $[A \rightarrow \alpha \cdot \beta, j] \in I_i$ を作成するために、状態の他に、 j と i を得る情報が含まれていなければならないことを 2 章で説明した。2 章では説明の簡単のために、 $trss$ の各節点には位置番号の集合が付加されていた。しかし、マージを効率よく行うために、識別子付き位置番号を導入し、これを付加することにしよう。識別子付き位置番号は全解析過程でユニークな番号であり、後述のマージ操作、シフト操作、レデュース操作により、スタックトップの節点が新たに作られる度に作り出され、このスタックトップの節点の位置番号に付加される。識別子が付加された位置番号のことを識別子付き位置番号と呼ぶ。位置番号が i で識別子が x なら、この識別子付き位置番号を x_i と書くことにする。識別子はマージの歴史を保存しておくためのもので、それにより重複した統合操作を避けることができる (4. 1. 2, 4. 1. 3)。レデュース操作時の逆ドット項の生成には、識別子付き位置番号の内、位置番号のみが使われることに注意したい (4. 1. 5)。YAGLR 法で用いる $trss$ の節点の構造は以下の通りである：

[識別子付き位置番号の集合, 状態]

たとえば、 $\{^2 2, ^4 3\}$, $\{^1 2, ^5 4, ^8 6\}$ は識別子付き位置番号の集合で、 $\{^3 2, ^4 3\}, 2]$, $[\{^1 2, ^5 4, ^8 6\}, 4]$ は $trss$ の節点である。

4. 1. 2 節点のマージ操作

YAGLR 法は富田法以上にスタックのマージを進めることができることを 2 章で説明した。 $trss$ の構造は富田法で用いる GSS の構造ほど複雑にはならない。スタックトップの節点中の識別子付き位置番号の集合には、最後にシフトした語に対する識別子付き位置番号が一つ含まれるだけである。以下では状態が同一の二つの節点のマージ操作 (M1) と (M2) について説明する。これを基本に、三つ以上の節点のマージも行うことができる。

(M1) スタックトップの節点 $[x_i, s]$ と $[y_i, s]$ をマージして、一つの統合節点 $[z_i, s]$ にすることができる。但し、 z は新たに作り出されたユニークな識別子である。また統合前の二つの節点の持つ全ての子節点を、統合節点の子節点とし、その後のマージの対象とする。

(M2) スタックトップ以外の節点 $[M, s]$ と $[N, s]$ をマージして、一つの統合節点 $[M \cup N, s]$ にすることができる。ただし、 M が N の部分集合なら、節点 $[N, s]$ の子節点を統合節点の子節点とする。さもなければ、統合前の二つの節点の持つ全ての子節点を、統合節点の子節点とし、その後のマージの対象とする。前者の場合、統合節点の子節点は節点 $[N, s]$ の子節点に等しく、それらの状態は全て異なるので、子節点同士のマージを更に進める必要がない。この事実とは、 $trss$ 同士のマ

ジ操作を途中で中断して良いことを意味しているので重要である。
 なお(M2)の妥当性については、レデュース操作が関係するので、レデュース操作を説明した後で与える。trss同士のマージ操作アルゴリズムのPASCAL風の定義を以下に与える。

4. 1. 3 マージ操作のアルゴリズム

```

procedure merge(current_set: set of trsses, flag: char)
var new_current_set: set of trsses;
{after this merge, current_set will be changed into the result of merge}
begin
  if all the top nodes in current_set have distinct states
  then return
  else begin
    while current_set is not empty do
    begin
      from current_set, pickup and retract all the trsses with the same state
      in the top node and name them TRSS';
      case flag of "top": apply (M1) repeatedly to merge all root nodes of TRSS' into one;
                    "otherwise": apply (M2) repeatedly to merge all root nodes of TRSS' into one
      end;
    end;
    new_current_set := a set of all the immediate decendent subtrees of the newly merged
                      root node of TRSS';
    merge(new_current_set, "otherwise")
  end
end merge;
  
```

trss同士のマージ例:

(a) $\cdots - [\{ 2, 3 \}, 8] - \cdots - [\{ 2, 4, 5 \}, 9] - [\{ 6 \}, 1] (\uparrow 7)$
 (b) $\cdots - [\{ 2, 3, 4 \}, 8] - \cdots - [\{ 4, 5 \}, 9] - [\{ 6 \}, 1] (\uparrow 7)$
 (c) $\cdots - [\{ 2, 3, 4, 5 \}, 8] - \cdots - [\{ 6 \}, 1] (\uparrow 7)$

上記したマージ操作のアルゴリズムに従い、(a)と(b)のマージを行うと(c)を得る。スタックトップの節点同士の統合から始め、(M1)による統合節点を作り、新しい識別子付き位置番号¹⁰⁶を導入する。次に子節点間のマージを行うが、両者の状態が9で等しいから(M2)によるマージをさらに進める。次の孫節点同士も同一の状態8を持つから、(M2)によるマージを行うが、この場合、両節点の識別子付き位置番号の集合間に部分集合関係が成立するので、マージ操作をここで止め、(M2)により(b)の節点 $[\{ 2, 3, 4 \}, 8]$ 以下をそのままマージの結果とする。富田法ではこうした深いマージは不可能であることに注意したい。

4. 1. 4 シフト操作

L R表から、先読み語(解析対象文のj+1番目の語 w_{j+1})の品詞がCでシフト操作"sh u"を(a)のtrssに施すべきことが分かったとしよう。このシフト操作を(a)に施すと(b)に示すtrssがえられる。それと共に(c)に示す逆ドット項を作る。シフト操作後のスタックトップの節点の位置番号はj+1になる。

(a) $\cdots - [M, s] - [\{ x \}, t] (\uparrow 7)$ "sh u"
 (b) $\cdots - [M, s] - [\{ x \}, t] - [\{ v(j+1) \}, u] (\uparrow 7)$
 (c) $l_j \ni [C \rightarrow \cdot w_{j+1}, j+1]$

4. 1. 5 レデュース操作

trssに対するレデュース操作(CFG規則 $A \rightarrow X_1 X_2 \cdots X_m$ を用いる)により、trssの一つのパス(a)は(b)に変わると共に、(c)に示す逆ドット項を作成する。

(a) $\cdots - [N_k, s_k] - [N_{k+1}, s_{k+1}] - \cdots - [N_{k+m}, s_{k+m}] (\uparrow 7)$
 (b) $\cdots - [N_k, s_k] - [N, t] (\uparrow 7)$

ここで状態tは、レデュース操作後にCFG規則の左辺の非終端記号Aと状態 s_k とからGOTO表を参照して決めた新しい状態である。また、 $N_k = \{ a, b, \dots \}$, $N_{k+1} = \{ c, d, \dots \}$, $\dots$, $N_{k+m-1} = \{ v, w, \dots, x, g \}$, $N_{k+m} = \{ j \}$, $N = \{ z \}$ であるとする。スタックトップの節点の識別子付き位置番号の集合 N_{k+m} には、最後にシフトした語の位置番号jに対する識別子付き位置番号 v_j のみが含まれ、レデュース後の新しいスタックトップの節点の識別子付き位置番号の集合Nには、新しいユニークな識別子付き位置番号 z_j のみが含まれることに注意したい。レデュース操作の前後で位置番号jは変わらない。

(c) 逆ドット項の作成:

$l_a \ni [A \rightarrow \cdot X_1 X_2 \cdots X_m, j]$	$\dots$
$l_b \ni [A \rightarrow \cdot X_1 X_2 \cdots X_m, j]$	$l_e \ni [A \rightarrow X_1 X_2 \cdots \cdot X_m, j]$
$\dots$	$l_f \ni [A \rightarrow X_1 X_2 \cdots \cdot X_m, j]$
$l_c \ni [A \rightarrow X_1 \cdot X_2 \cdots X_m, j]$	$\dots$
$l_d \ni [A \rightarrow X_1 X_2 \cdots \cdot X_m, j]$	$l_g \ni [A \rightarrow X_1 X_2 \cdots \cdot X_m, j]$

レデュース操作では、操作の指定するCFG規則の右辺の非終端記号の数だけ、スタックトップから節点をポップする。ポップする各節点には、CFG規則の右辺の非終端記号が対応するので、上記した(1)の場合、 $X_1, X_2, \dots, X_m$ のそれぞれには、(a)の節点、 $[N_{k+1}, s_{k+1}]$, $[N_{k+2}, s_{k+2}]$, $\dots$, $[N_{k+m}, s_{k+m}]$ が順に対応する。(a)に含まれる節点の位置番号は、どこまでがシフト済み(処理済み)かを示しているので、これを用いて逆ドット項のドット位置(逆ドット項の集合I

の添字)を知ることができる。逆ドット項内に書かれる位置番号は全て、スタックトップの節点の持つ位置番号]であり、位置番号に付加された識別子は、逆ドット項の作成には関与していないことに注意されたい。

4. 1. 6 節点のマージ操作 (M2) の妥当性

節点のマージ操作 (M1) が成り立つことは自明であろう。二つの節点[M, s]と[N, s]に関して (M2) が成立することは、次の (1) から (3) で証明できる。以下では、trssの集合をTRSSと呼ぶことにする。

- (1) YAGLR法では、全てのレデュース操作が終了してからシフト操作を行う。シフト操作を行わない限りTRSSの全てのスタックトップの節点の持つ位置番号は、最後にシフトした語の位置番号に等しく同一でなければならない。(それにより、レデュース操作で作成する全ての逆ドット項中の位置番号は、最後にシフトした語の位置番号であり変わらない)。
- (2) 逆ドット項の集合Iの添字は、逆ドット項中のドットの位置番号を表す。この添字は、レデュース操作の対象となる節点の持つ識別子付き位置番号の集合から1つずつ取り出されるもので、マージ前の二つの節点の識別子付き位置番号の集合から別々に取り出しても、マージ後の、識別子付き位置番号の和集合から取り出しても変わらない。
- (3) MがNの部分集合でない場合に、統合前の二つの節点の持つ全ての子節点を、統合節点の子節点として良いことは明らかである。一方、MがNの部分集合である場合に、節点[N, s]の子節点をそのまま統合節点の子節点として良い理由は次の通りである。MがNの部分集合であるということは、節点[M, s]を得るために行ったのと全く同じマージの履歴が、節点[N, s]に残されていることを意味している。従って、統合節点以降のマージの結果は、既に節点[N, s]の子節点以降に含まれているので、両節点の統合節点を[N, s]とし、[N, s]の子節点をそのまま統合節点の子節点としてよい。

4. 2 YAGLR法の手続き

YAGLR法による入力文解析の手続きを以下に示す。

- (1) trssの集合TRSSの初期状態を、($\#$ トM) [[(0), 0] (ト7'))とする。
- (2) TempStack:= []。
- (3) TRSSが空 ([]) なら (5) へ、
さもなければ、TRSSからtrssを一つ取り出し (TRSS:=TRSS-trss) 以下を行う。
L R表によりtrssに対する'操作'を決める。
もし'操作'が受理の場合: 成功として (3) へ戻る、
誤りの場合: 失敗として (3) へ戻る、
シフトの場合: trssをTempStackに入れて (3) へ戻る、
レデュースの場合: (4) へ、
シフト/レデュースの場合: シフト操作とレデュース操作を切り離し、シフト操作付きのtrssをTempStackに入れ、レデュース操作付きのtrssに対して (4) へ。
- (4) trssに対して指定のレデュース操作を行い、その結果新たに作られるtrss'をTRSSに入れる、
merge(TRSS, "top")を実行後 (3) へ戻る。
- (5) TempStackが空なら終了、
さもなければ、TempStack中の各trssにシフト操作を施し、結果をTRSSに入れ、
merge(TRSS, "top")を実行後 (2) へ戻る。

以上の手続きでは、シフト・レデュース・コンフリクトが生じたとき、富田法と同様、レデュース操作を優先させていることが分かる。

5. 富田法とYAGLR法による統語解析例

本章では図5. 1に示すCFGを用いて、富田法とYAGLR法のそれぞれに対する統語解析例を示し、両者の比較を行う。解析対象文は「文化 が きた から 伝わった から...」であるとし、解析中途まで双方のトレースを行う。図5. 1のCFGから図5. 2に示すLR表が得られる。なお、富田法のスタックの節点の構造は、[部分木番号, 状態]である。

- | | |
|--------------|--------------|
| (1) S → PP S | (5) v → きた |
| (2) S → v | (6) v → 伝わった |
| (3) PP → n p | (7) n → きた |
| (4) PP → S p | (8) n → 文化 |
| | (9) p → から |
| | (10) p → が |

図5. 1 簡単な日本語CFG

状態	アクション部				GOTO部	
	n	p	v	\$	PP	S
0	sh4		sh3		2	1
1		sh5		受理		
2	sh4		sh3		2	6
3		re2		re2		
4		sh7				
5	re4		re4			
6		sh5 rel		rel		
7	re3		re3			

図5. 2 図5. 1のCFGから得られたLR表

5. 1 富田法による解析

T1)	[_, 0]-
T2)	[_, 0]-[0, 4]-
T3)	[_, 0]-[0, 4]-[1, 7]-
T4)	[_, 0]-[2, 2]-
T5)	[_, 0]-[2, 2]-[3, 4]- [4, 3]-
T6)	[_, 0]-[2, 2]-[3, 4]- [5, 6]-
T7)	[_, 0]-[2, 2]-[3, 4]- [5, 6]- [6, 1]-
T8)	[_, 0]-[2, 2]-[3, 4]-[7, 7]- [5, 6]-[7, 5]- [6, 1]-
T9)	[_, 0]-[2, 2]-[8, 2]- [9, 2]- [10, 2]-
T10)	[_, 0]-[2, 2]-[11, 2]-[12, 3]- [10, 2]-
T11)	[_, 0]-[2, 2]-[11, 2]-[13, 6]- [10, 2]-
T12)	[_, 0]-[14, 1]- [15, 6]- [2, 2]-[11, 2]-[13, 6]- [10, 2]-
T13)	[_, 0]-[14, 1]- [16, 1]- [15, 6]- [2, 2]-[11, 2]-[13, 6]- [10, 2]-
T14)	[_, 0]-[17, 1]- [15, 6]- [2, 2]-[11, 2]-[13, 6]- [10, 2]-

(以下省略)

文化(n)	sh4	
が(p)	sh7	0 [n 文化]
きた(n;v)	re3	1 [p が]
きた(n)	sh4	2 [PP (0 1)]
きた(v)	sh3	
から(p)	sh7	3 [n きた]
	re2	4 [v きた]
から(p)	sh7	
	sh5, re1	5 [S (4)]
から(p)	sh7	
	sh5	
	sh5	6 [S (2 5)]
伝わった(v)	re3	7 [p から]
	re4	sh5 (at T8)によるマージ
伝わった(v)	sh3	
		8 [PP (3 7)]
		9 [PP (5 7)]
		10 [PP (6 7)]
から(p)	re2	マージ (圧縮統語森11)
		sh3 (at T9)によるマージ
		11 [PP (3 7) (5 7)]
		12 [v 伝わった]
から(p)	sh5, re1	13 [S (12)]
から(p)	sh5	
	sh5, re1	14 [S (10 13)]
	sh5	15 [S (11 13)]
から(p)	sh5	
		16 [S (2 15)]
	sh5	
から(p)	sh5	マージ (圧縮統語森17)
	sh5	17 [S (2 15) (10 13)]

5. 2 YAGLR法による解析

Y1)	[{ ⁰ 0}, 0]-
Y2)	[{ ⁰ 0}, 0]-[{ ¹ 1}, 4]-
Y3)	[{ ⁰ 0}, 0]-[{ ¹ 1}, 4]-[{ ² 2}, 7]-
Y4)	[{ ⁰ 0}, 0]-[{ ³ 2}, 2]-
Y5)	[{ ⁰ 0}, 0]-[{ ³ 2}, 2]-[{ ⁴ 3}, 4]- [{ ⁰ 0}, 0]-[{ ³ 2}, 2]-[{ ⁵ 3}, 3]-
Y6)	[{ ⁰ 0}, 0]-[{ ³ 2}, 2]-[{ ⁴ 3}, 4]- [{ ⁰ 0}, 0]-[{ ³ 2}, 2]-[{ ⁶ 3}, 6]-
Y7)	[{ ⁰ 0}, 0]-[{ ³ 2}, 2]-[{ ⁴ 3}, 4]- [{ ⁰ 0}, 0]-[{ ³ 2}, 2]-[{ ⁶ 3}, 6]- [{ ⁰ 0}, 0]-[⁷ 3}, 1]-
Y8)	[{ ⁰ 0}, 0]-[{ ³ 2}, 2]-[{ ⁴ 3}, 4]-[{ ⁸ 4}, 7]- [{ ⁰ 0}, 0]-[{ ³ 2}, 2]-[{ ⁶ 3}, 6]-[{ ⁹ 4}, 5]- [{ ⁰ 0}, 0]-[⁷ 3}, 1]-
Y9)	[{ ⁰ 0}, 0]-[{ ³ 2}, 2]-[{ ¹¹ 4}, 2]- [{ ⁰ 0}, 0]-

文化(n)	sh4	10 [n→・文化, 1]
が(p)	sh7	11 [p→・が, 2]
きた(n;v)	re3	10 [PP→・n p, 2]
		11 [PP→n・p, 2]
きた(n)	sh4	12 [n→・きた, 3]
きた(v)	sh3	12 [v→・きた, 3]
から(p)	sh7	
	re2	12 [S→・v, 3]
から(p)	sh7	
	sh5, re1	10 [S→・PP S, 3]
		12 [S→PP・S, 3]
から(p)	sh7	13 [p→・から, 4]
	sh5	
	sh5	マージ
伝わった(v)	re3	12 [PP→・n p, 4]
		13 [PP→n・p, 4]
	re4	12 [PP→・S P, 4]
		13 [PP→S・P, 4]
		10 [PP→・S P, 4]
		13 [PP→S・P, 4]*
伝わった(v)	sh3	15 [v→・伝わった, 5]

Y10) $[(\{^0\}, 0)] - [(\{^2\}, 2)] - [(\{^{14}\}, 2)] - [(\{^{15}\}, 3)] -$
 $[(\{^0\}, 0)]$ -----
Y11) $[(\{^0\}, 0)] - [(\{^2\}, 2)] - [(\{^{14}\}, 2)] - [(\{^{15}\}, 6)] -$
 $[(\{^0\}, 0)]$ -----
Y12) $[(\{^0\}, 0)] - [(\{^2\}, 2)] - [(\{^{14}\}, 2)] - [(\{^{15}\}, 6)] -$
 $[(\{^0\}, 0)]$ -----
 $[(\{^0\}, 0)] - [(\{^2\}, 2)]$ ----- $[(\{^{14}\}, 6)] -$
 $[(\{^0\}, 0)]$ ----- $[(\{^{15}\}, 1)] -$
Y13) $[(\{^0\}, 0)] - [(\{^2\}, 2)] - [(\{^2, ^{14}\}, 2)] - [(\{^{17}\}, 6)] -$
 $[(\{^0\}, 0)]$ -----
 $[(\{^0\}, 0)]$ ----- $[(\{^{16}\}, 1)] -$
(以下省略)

から(p) re2 14 $[S \rightarrow \cdot v, 5]$
から(p) sh5, rel 12 $[S \rightarrow \cdot PP S, 5]$
14 $[S \rightarrow PP \cdot S, 5]$
10 $[S \rightarrow \cdot PP S, 5]$
から(p) sh5
sh5, rel 10 $[S \rightarrow \cdot PP S, 5]*$
12 $[S \rightarrow PP \cdot S, 5]$
sh5
から(p) sh5 マージ
sh5

注) *付きの項は、重複項を表す。

ここで富田法とYAGLR法の比較をして見たい。T11)からT14)までとY11)からY13)までを比較してみよう。YAGLR法は、trssの構造が単純な分だけ、実装が容易である。また、YAGLR法は富田法以上にtrss同士のマージを進めることができるので、GSSを用いずとも計算効率と計算時に使用する記憶空間の縮小をはかることが可能になる。解析結果に含まれる曖昧性が増大するにつれて、YAGLR法と富田法の違いが際だってくる。6章では実験によりこのことを説明する。

6. 実験結果

本章では、木構造化スタックを用いた富田法（これはSGLRとよばれている[沼崎 90]）とYAGLRとの比較を、具体的な文法を用いて実行速度の比較を行う。実験にはSun3/2607-クォーターン、Quintus-Prologを用いる。

・解析対象文：aaa...a

・解析対象文：I saw a lady with telescope in the park
in the park.....

・使用文法： $S \rightarrow a$
 $S \rightarrow SS$
 $S \rightarrow SSSS$

・使用文法：123規則の英語文法

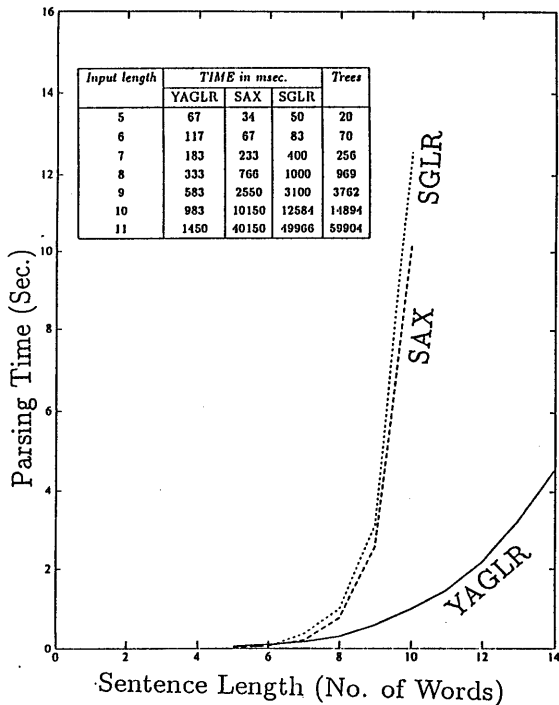


図6.1 実験結果1

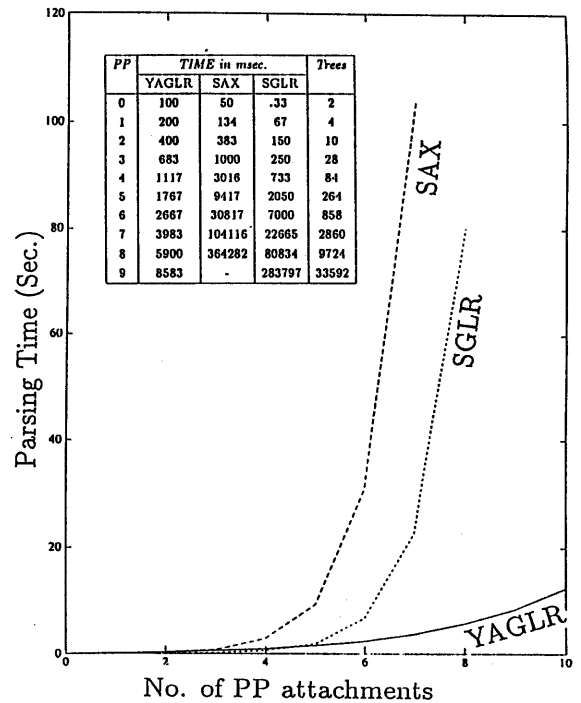


図6.2 実験結果2

図6. 1には、Johnsonらの与えた文法に対する実験結果を示す。この文法は、富田法ではアーリー法以上の解析時間のオーダーを要するものである。これに対して、付録Aに示すように、YAGLR法はアーリーと同等の解析時間のオーダー（文の長さ n に対して n^3 のオーダー）であることが証明されている。このことが図6. 1の結果からも読み取れる。文の長さが長くなるにつれて、YAGLR法の高速度性が際だって来る。たとえば n が10ではYAGLR法は約13倍、 n が11では約34倍高速である。図2はPP付加の数を増やすことにより、解析結果の曖昧性を増大させ、解析時間がどの様に変化するかを示したものである。この場合も図6. 1とはほぼ同様な結果が得られている。図6. 1と図6. 2には、参考のためにSAX [Matsumoto 89]の解析時間も示してあるが、SGLRとSAXとも、解析木を作らない場合の解析時間である。一方、YAGLR法の場合には、逆ドット項を作成する時間が含まれていることに注意。

7. おわりに

本論文ではYAGLR法と呼ぶ新しい統語解析アルゴリズムを提案した。YAGLR法は、既存の方法と比べていくつかの利点を持っていることを説明した。本論文の結論を以下に述べる。

- (A) Johnsonらの与えた特殊な文法に対しても、解析時間のオーダーは、解析対象文の長さ n に対してアーリー法と同等の n^3 のオーダーに抑えることができる。付録Aにその証明を与えてある。
- (B) レデュース操作時に、アーリー項と対称な逆ドット項を作成しながら統語解析を行う。逆ドット項を生成する利点について考察した。
 - ・先読み情報を用いているので、最終的に生成される逆ドット項の数はアーリー法より少ない。
 - ・逆ドット項を用いることにより、富田法以上にtrss同士のマージを行うことができるため、操作し易い木構造化スタックtrssを用いることができる。これは富田法で用いるグラフ構造化スタックより構造が単純であるとともに、グラフ構造化スタックと同様に、使用記憶空間を抑えることができる。
- (C) 実験により、YAGLR法が高速な統語解析アルゴリズムであることを実証した。YAGLR法の高速度性は、解析結果に含まれる曖昧性の数が増えるにつれて際だって来る。

今後の検討課題として、以下のものを挙げることができる。

- (1) YAGLR法の統語解析時間のオーダーが、一般のCFGに対して n^3 のオーダーであることの証明。
- (2) YAGLR法による統語解析に要する空間量の評価。
- (3) YAGLR法の並列解析アルゴリズムの研究。
- (4) 逆ドット項からの統語構造の並列抽出アルゴリズムの研究。

(1)は重要な研究課題であるが、(2)に述べたように、解析に要す空間の大きさの評価も行う必要がある。Kippsは富田法に比較的単純な修正を施すことで、一般のCFGに対する計算時間のオーダーを $\Omega(n^3)$ に抑えることができることを示した[Kipps 89]。しかし、このアルゴリズムは富田法以上の空間を要す。これに対してYAGLR法では、trssのマージを徹底的に行うアルゴリズムであるから、計算空間を抑えることができる可能性がある。今後、一般のCFGに対するYAGLR法の使用記憶空間の大きさの詳細な検討を行う必要がある。

(3)に述べたように、富田法と同様、YAGLR法の並列解析アルゴリズムを検討することも意味がある。本稿で述べたアルゴリズムは構型探索の逐次型アルゴリズムであり、できるだけ無駄な解析を省いたアルゴリズムである。この長所を生かしつつ、シフトレデュース・コンフリクトが発生したときの待ち合わせができるだけ少なくなる並列解析アルゴリズムを研究する必要がある。それには沼崎らの研究が参考になろう[Numazaki 90]。

YAGLR法では、統語解析後に得られた逆ドット項の集合から統語解析木を作り出さねばならない。これはアーリー法と対称なアルゴリズムを用いれば良い。一つの統語解析木を計算するための時間は n^2 のオーダーであることが知られている

[Aho 72]。この統語解析木の計算にも多くの並列性が観察されるので、統語解析木抽出用の並列アルゴリズムについて今後研究する必要がある。

参考文献

- [Aho 72] Aho, A.V. and Ullman, J.D.: The Theory of Parsing, Translation and Compiling, Prentice-Hall, New Jersey (1972).
- [Earley 70] Earley, J.: An Efficient Augmented-Context-Free Parsing Algorithm, Comm. of ACM, 13, 1-2, 95-102 (1970).
- [Johnson 89] Johnson, M.: The Computational Complexity of Tomita's Algorithm, International Parsing Workshop '89, Carnegie-Mellon University, 203-208 (1989).
- [Kipps 89] Kipps, J. N.: Analysis of Tomita's Algorithm for General Context-Free Parsing, International Parsing Workshop '89, Carnegie-Mellon University, 193-202 (1989).
- [Matsumoto 89] Matsumoto, Y.: Natural Language Parsing Systems based on Logic Programming, Ph.D Theses, Kyoto University (1989).
- [Numazaki 90] Numazaki, H. and Tanaka, H.: A New Parallel Algorithm for Generalized LR Parsing, COLING'90 305-310 (1990).
- [田中 89] 田中穂積: 自然言語解析の基礎, 産業図書 (1989).
- [Tomita 86] Tomita, M.: Efficient Parsing for Natural Language, Kluwer, Boston, Mass (1986).
- [Tomita 87] Tomita, M.: An Efficient Augmented-Context-Free Parsing Algorithm, Computational Linguistics, 13, 31-46 (1987).

付録A 解析対象文の長さに対する計算の複雑性

Johnsonらによれば、次の一連の文法 L_m に対して、解析対象文の長さを $n+2$ とした時、富田法の解析時間のオーダーは n^3 である [Johnson 89]

[Kipps 89] . $m \geq 3$ なら、解析時間のオーダーが n^3 を超えてしまう。

文法 L_m : (1) $S \rightarrow a$ (2) $S \rightarrow SS$ (3) $S \rightarrow S^{m+2}$

ここで S^{m+2} は記号 S が $m+2$ 個連結したものの省略形である。

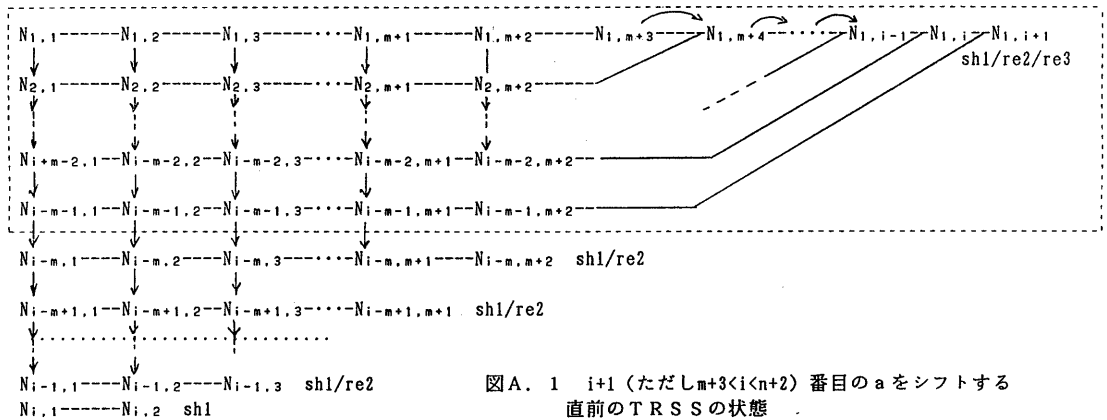
解析対象文を a^{n+2} とする(ただし $n \geq m$)。上記した規則に対して右のLR表が得られる。今、 i ($m+3 < i < n+2$) 番目の a をシフトし rel を施した直後のTRSSの状態を図A. 1に示す。ここで $N_{p,q}$ は節点を表し、各節点の持つ識別子付き位置番号の集合間には、矢印で示す包含関係がある。節点 X と Y の間に $X \rightarrow Y$ があれば、 X の識別子付き位置番号の集合が Y の部分集合であることを示す。

容易に示すことができるように、節点 $N_{p,q}$ の持つ状態 $T_{p,q}$ は、

$$T_{p,q} = \begin{cases} q & \text{if } 1 \leq q \leq m+3, \\ m+3 & \text{if } m+3 < q, \\ 0 & \text{if } q = 1. \end{cases}$$

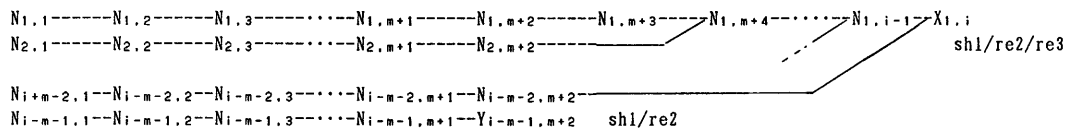
図A. 1のtrssに対して、シフト操作の前に、全てのレデュース操作を施す必要があるが、そのための計算コストを考えてみよう。まず、規則(2)を適用してレデュース操作を進める場合の方が、規則(3)を適用する場合に比べてコストがかかることは明らかである。したがって、計算コストの

状態	アクション部		GOTO部
	a	\$	
0	sh1		2
1	rel	rel	
2	sh1	acc	3
3	sh1/re2	re2	4
4	sh1/re2	re2	5
....			
$m+1$	sh1/re2	re2	$m+2$
$m+2$	sh1/re2	re2	$m+3$
$m+3$	sh1/re2/re3	re2/re3	$m+3$



図A. 1 $i+1$ (ただし $m+3 < i < n+2$) 番目の a をシフトする直前のTRSSの状態

オーダーを求めるためには、規則(2)を適用するレデュースを考えれば十分である。また、点線で囲まれたTRSSの部分のみが、入力文の長さに依存しているので、これに対するレデュース操作の計算コストを考えれば十分である。今、図A. 1の最上部のtrssに対して、規則(2)によるレデュース操作を一度施すと、下図の二つの新しいtrssが得られる。



上図のtrssは、図A. 1の統合化可能な他のtrssと最終的にマージされる。たとえば、シフト操作の直前では、sh1を待つtrssのスタックトップの状態が $m+3$ で等しいもの同士はマージされる。図A. 1の最長スタックと、上記した新たな二つのtrssの内、上のtrssは、スタックトップの節点の状態が $m+3$ であるからマージされる。ただし、図A. 1に記述した節点間の識別子付き位置番号の集合間に包含関係が成り立つため、マージ操作は先頭から二つの目までの節点同士 ($<N_{1,i+1} \text{ と } X_{1,i}>$, $<N_{1,i} \text{ と } N_{1,i-1}>$, $<N_{i-m-1,m+2} \text{ と } N_{i-m-2,m+2}>$) で行えばよい。先頭から3番目の節点同士 ($<N_{1,i-1} \text{ と } N_{1,i-2}>$ または $<N_{i-m,m+1} \text{ と } N_{i-m-2,m+1}>$) のマージは、識別子付き位置番号の集合間に部分集合関係が成立しているため、マージ操作はここで終わり、それ以上マージを葉の方向に向けて進める必要はない。

以上の考察から、re2操作後のマージには先頭から3番目の節点までのマージの手間を考慮すればよいことが分かる。しかも、それぞれのマージには、識別子付き位置番号の集合の要素が高々 $i+1$ 含まれるだけであるから、マージに要す計算コストは i のオーダーである。一方、re2は繰り返して行われるが、その回数は高々 i 回である。したがって、レデュース操作とマージ操作にかかる時間の総和のオーダーは i^2 である。以上より、 a^{n+2} の入力文を解析するのに要する総計算コストのオーダーは、 $\sum_{i=2}^{n+2} \Omega(i^2) = \Omega(n^3)$ となり証明された。

次に図A. 1の各節点中の識別子付き位置番号の数は、節点 $N_{1,j}$ に対して、 $2 \leq j \leq m+3$ の場合に1、 $m+4 \leq j \leq i-1$ の場合に2、

$i \leq j \leq i+1$ の場合に1, また節点 $N_{k,j}$ ($2 \leq k \leq i-m-1$, $2 \leq j \leq m+2$)に対して k である。したがって, 点線で囲まれた部分の節点中の識別子付き位置番号の総和は, $1*(m+4)+2*(i-m-4)+(m+2)(2+3+\dots+(i-m-1))$ であり, そのオーダは i^2 である。これは, 点線で囲まれた部分以外の節点に含まれる識別子付き位置番号の総和のオーダーと等しい。以上から, 入力文 a^{n+2} を解析するために必要なT R S Sの空間のオーダは n^2 であり, また, 作成される逆ドット項の総数のオーダーは n^3 であることが証明された。

付録B YAGLR法の位置づけ

