4F-3

高速な依存構造解析アルゴリズム

加藤　直人　　江原　暉将
NHK放送技術研究所

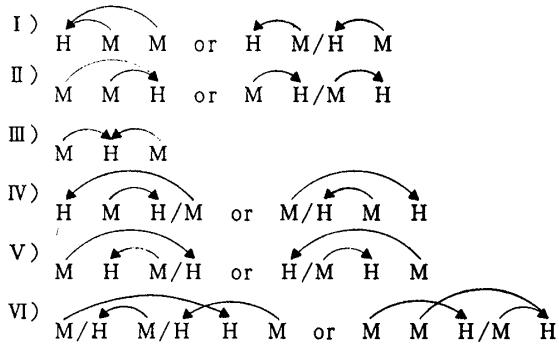
1．はじめに

構文解析を実現する文法理論の枠組みとしては構成文法、依存文法がある。現在、一般的には構成文法の中の文脈自由文法が構文解析に使われている。これは解析の効率が良く、計算機にインプリメントしやすい構文解析アルゴリズムがあるからである。[1] 一方、依存文法は文脈自由文法では不可欠な非終端接点が必要ない、語と語との直接的な関係がわかるなどの利点がある。

本文では、依存文法による効率の良い構文解析アルゴリズムについて報告する。

2．依存文法[2]

依存文法では語と語との関係を主要語・修飾語という依存関係でとらえる。依存関係を主要語と修飾語との位置関係により分類すると、次の6つの場合がある。ここで、Hは主要語、Mは修飾語を表す。また、M/H やH/M はある語が他の語の修飾語であり、同時にまた別の語の主要語であることを示す。



ここでは、便宜的に修飾語が2語または3語の場合だけしかあげていないが、修飾語はこれより多くてもよい。

Ⅰ）～Ⅳ）は多くの言語に見られる語順の特徴である。また、Ⅴ）における主要語は他の語の修飾語にもなるので、Ⅵ）の交差依存と同質のものといえる。本文ではⅠ）～Ⅳ）の依存関係のみを扱い、Ⅴ）、Ⅵ）は除外する。

日本語の係り受け構造はⅠ）の関係であり、尾関によって効率の良い解析アルゴリズムが提案されている。[3] 本文のアルゴリズムはⅠ）～Ⅳ）の関係が扱えるように、これを拡張したものである。

A Fast Algorithm for Dependency Structure Analysis
Naoto KATOH, Terumasa EHARA
NHK Science and Technical Research Laboratories

3．問題の定式化

文中のある単語列に対して依存関係を定めると依存構造ができる。依存構造を次のように再帰的に定義する。

定義3．1　依存構造

1)　単語xに対して、<x>は依存構造であり、xはこの依存構造の中心語である。

2)　中心語がそれぞれx、yである依存構造X、Yに対して、（X Y）、［Y X］はそれぞれ依存構造であり、yはこの依存構造の中心語である。ここで、xを修飾語、yを主要語という。

3)　上の1)、2)を満たす記号列のみが依存構造である。

（X Y）は依存構造Xの中心語xが依存構造Yの中心語yに前から係ることを表し、［Y X］はxがyに後ろから係ることを表している。2．で示した依存関係Ⅰ）～Ⅳ）はすべて、定義3．1の依存構造で表すことができる。また、定義3．1の中心語の定義から明らかに、1つの依存構造に対して、必ず1つの中心語が決まる。

定義3．2

単語列$x_1 x_2 \cdots x_n$に適切に括弧（<>，()，[]）を付け、依存構造になるようにしたものを$x_1 x_2 \cdots x_n$上の依存構造といい、$k(x_1 x_2 \cdots x_n)$と表す。この依存構造に対して決まる中心語を$cen(k(x_1 x_2 \cdots x_n))$と表す。また、$k(x_1 x_2 \cdots x_n)$の全体を$K(x_1 x_2 \cdots x_n)$と表す。

2つの単語x、yの間に依存関係があるとき、この整合性を非負の値をとる関数

1)　（x y）のとき、pen(x, y)

2)　［x y］のとき、pen'(x, y)

で表す。また、関数の値が0に近いほど整合性は高いとする。この関数は文法的関係、意味的関係などによって値が決まるものである。

この関数を用いて依存構造の適格度P(Z)を次のように定義する。

定義3．3

1)　Z＝<x>のとき
$P(Z)=0$

2)　X、Yは依存構造であり、Z＝（X Y）のとき
$P(Z)=P(X)+P(Y)+pen(x_{h1}, y_{h2})$

3)　X、Yは依存構造であり、Z＝［X Y］のとき
$P(Z)=P(X)+P(Y)+pen'(x_{h1}, y_{h2})$

ここで、$x_{h1}=cen(X)$、$y_{h2}=cen(Y)$

定義3．4

単語列$x = x_i x_{i+1} \cdots x_j$に対して

1) $OPTP(x_i x_{i+1} \cdots x_j)$

$= \min (x \in K(x_i x_{i+1} \cdots x_j))$ [P(x)]

= ($x \in K(x_i x_{i+1} \cdots x_j)$に対するP(x)の最小値)

2) $OPTK(x_i x_{i+1} \cdots x_j)$

$= \arg (x \in K(x_i x_{i+1} \cdots x_j))$ [P(x)]

= (1)の最小値を与える依存構造x)

4．アルゴリズムの構成

3．で導入した記号を用いると、文の依存構造解析をすることは次の問題を解くことに帰着される。

問題

文を構成する単語列$x_1 \cdots x_n$に対して

1) $OPTP(x_1 \cdots x_n)$

$= \min (x \in K(x_1 \cdots x_n))$ [P(x)]

2) $OPTK(x_1 \cdots x_n)$

$= \arg (x \in K(x_1 \cdots x_n))$ [P(x)]

を求めよ。

この問題を解くためには次の再帰方程式を使えばよい。

定理4．1

単語列$x_i x_{i+1} \cdots x_j$に対して

1) $OPTP(x_i) = 0$

2) $OPTP(x_i x_{i+1} \cdots x_j)$

$= \min (i \leqq k \leqq j-1)$ [$OPTP(x_i \cdots x_k)$

$+ PEN(x_{h1}, x_{h2}) + OPTP(x_{k+1} \cdots x_j)$]

ここで、

$PEN(x_{h1}, x_{h2})$

$= \min [pen(x_{h1}, x_{h2}), pen'(x_{h1}, x_{h2})]$ (4.1)

$x_{h1} = cen(OPTK(x_i \cdots x_k))$

$x_{h2} = cen(OPTK(x_{k+1} \cdots x_j))$

3) 2)で求められたkに対して

$OPTK(x_i x_{i+1} \cdots x_j)$

$= \begin{cases} (OPTK(x_i \cdots x_k)\ OPTK(x_{k+1} \cdots x_j)) \\ \quad \text{if} \quad PEN(x_{h1}, x_{h2}) = pen(x_{h1}, x_{h2}) \\ [OPTK(x_i \cdots x_k)\ OPTK(x_{k+1} \cdots x_j)] \\ \quad \text{if} \quad PEN(x_{h1}, x_{h2}) = pen'(x_{h1}, x_{h2}) \end{cases}$

［証明略］

定理4．1でOPTP($x_1 \cdots x_n$)、OPTK($x_1 \cdots x_n$)を求めれば依存構造が得られる。このアルゴリズムをパスカル風に記述するとFig.4.1 のようになる。

table2(i,j) はi番目の単語からj番目の単語までの最適な依存構造を記憶するためのテーブルである。したがって、最適な依存構造は、table2(1,n) に求められる。

```
for i:= 1 to n do
 begin
  table1(i,1):= 0;
  table2(i,1):= '<xi>';
  table3(i,1):= 'xi'
 end
for j:= 2 to n do
 for i:= 1 to n-j+1 do
  begin
   for k:= 1 to j-1 do
    f1(k):= table1(i,k)
       + PEN(table3(i,k), table3(i+k,j-k))
       + table1(i+k,j-k);
   table1(i,j):= min( 1≦k≦j-1 )f1(k);
   table2(i,j):= arg( 1≦k≦j-1 )f1(k);
   table3(i,j):= cen(table2(i,j))
  end
```

Fig.4.1 依存構造解析アルゴリズム

これまで説明の便宜上、OPTKは一意に定まるとしたが、一般には複数個になる。その場合への拡張は簡単である。

5．例題

次の文に対する依存構造解析を考える。

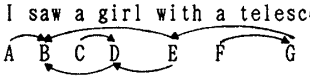


Fig.4.1のアルゴリズムにしたがって計算すると、table2(1,6)より次の７つの依存構造が得られる。

table2(1,6)＝{[[(<A><B>)(<C><D>)][<E>(<F><G>)]], －①

[(<A>[<B>(<C><D>)])[<E>(<F><G>)]], －①

[(<A><B>)[(<C><D>)[<E>(<F><G>)]]], －②

[(<A><B>)(<C>[<D>[<E>(<F><G>)]])], －②

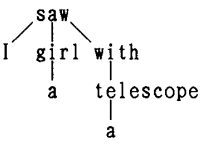
(<A>[[<B>(<C><D>)][<E>(<F><G>)]]), －①

(<A>[<B>[(<C><D>)[<E>(<F><G>)]]]), －②

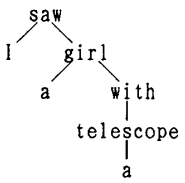
(<A>[<B>(<C>[<D>[<E>(<F><G>)]])])} －②

これらのうち同値なものをまとめて依存文法のtree構造に書き直すと、次の２つのようになる。

①のとき



②のとき



6．おわりに

以上、高速な依存構造解析アルゴリズムについて述べた。本文では述べなかったが、このアルゴリズムの効率は単語数の３乗に比例する。

【参考文献】

[1] 野村：「自然言語処理の基礎技術」電子情報通信学会（1988）

[2] 児玉：「語順の普遍性」山口書店（1987）

[3] 尾関：「最適文節列を選択するための多段決定アルゴリズム」信学技報 SP86－32（1986）