확률문맥자유문법의 규칙확률 추정을 위한 새로운 기법

변성찬 나동열
연세대학교 전산학과
강원도 원주시 홍업면 매지리 234, 우:220-710
{schbyun, dyra}@dragon.yonsei.ac.kr

A New Method for Estimating Rule Probabilities of Stochastic Context-Free Grammars

Seong-Chan Byun Dong-Yul Ra
Dept. of Computer Science, Yonsei University

요 약

본 논문에서는 확률문맥자유문법의 규칙확률을 추정하여 주는 새로운 알고리즘을 제시한다. 이 알고리즘은 이미 잘 알려진 인사이드-아웃사이드 알고리즘에 비하여 개념적으로 이해하기 쉽다는 장점을 가지고 있으며 속도면에서 훨씬 빠르다는 것이 실험으로 입증되었다.

1 서론

확률문맥자유문법 (stochastic context-free grammar; SCFG)은 통계기반 자연어 구문 분석에서 많이 사용된다 [3,5]. SCFG에서 필수적인 것은 각 규칙마다 확률값을 추정하여야 한다는 것이다. 규칙의 확률에 올바른 값을 추정하여 주는 것이 매우 중요하고도 어려운 문제이다. 규칙확률의 추정 문제에 관한 지금까지 유일한 것으로 인사이드-아웃사이드 (inside-outside) 알고리즘이 있다 [2,6].

본 논문에서는 규칙확률을 추정하는 새로운 알고리즘을 제안한다. 본 논문의 알고리즘은 규칙확률의 기본적인 개념을 그대로 적용한 것이다 [4]. 따라서 알고리즘을 이해하는 데 있어서 인사이드-아웃사이드의 경우보다 용이하다. 우리의 알고리즘의 보다 중요한 장점은 인사이드-아웃사이드 알고리즘에 비교하여 속도가 훨씬 빠르다는 점이다.

2 배경

2.1 확률문맥자유문법

확률 문맥 자유문법 (stochastic context free grammars; SCFG)은 문맥자유문법(CFG) 에 확률 개념을 추가하여 확장한 것이다 [3]. 문맥자유문법 $G=(N,\Sigma,R,S)$ 가 주어졌다 하자. 여기에서 N은 비단말기호(nonterminal symbol), Σ 는 단말기호(terminal symbol) 의 집합을 나타낸다. R 은 규칙 집합을 나타내며 S는 시초비단말기호 (starting nonterminal symbol)을 나타낸다. G에 기반한 한 SCFG는 각 규칙마다 확률값을 가진다. 즉 규칙 r 은 규칙확률 $P(r)$, $0 \le P(r) \le 1$, 을 가진다. 이때 좌측편(left hand side)심볼이 같은 규칙들의 확률의 합은 1 이 되어야 한다. 즉

$$\sum_{\alpha} P(A \rightarrow \alpha) = 1 \text{ for each nonterminal } A$$

본 논문에서 그리스 문자 α, β 는 비단말기호나 단말기호로 된 임의의 스트링을 나타내기로 한다. $P(A \rightarrow \alpha)$ 의 의미를 살펴 보자. 다음과 같은 문장 유도(derivation) 과정에서

$$S \Rightarrow^{*} x A \beta$$

까지 유도가 된 상황이라 하자 (본 논문에서 유도는 모두 좌측우선유도(leftmost derivation)을 가정하므로 x 는 단말기호로만 된 스트링을 나타낸다) . 이때 다음 유도를 위해서는 비단말기호 A 를 A-규칙(심볼 A를 좌측편심볼로 가지고 있는 규칙) 중 하나를 이용하여 그 규칙의 오른편(right hand side)스트링으로 대체하여야 한다. 만약 $A \rightarrow \alpha$ 를 이용한다면 이 유도는

$$S \Rightarrow^{*} x A \beta \Rightarrow^{*} x \alpha \beta$$

이 된다. 이때 A-규칙이 여러 개라면 그 중 어느 것을 사용하여도 무방하다. $P(A\rightarrow\alpha)$ 는 여러개의 A-규칙 중에서 $A\rightarrow\alpha$ 를 사용할 확률을 나타낸다.

입력 문장 $\omega = a_1 \cdots a_n$ 이 주어 졌다 하자. 이 문장을 문법 G 가 유도 과정 $S \Rightarrow^* \omega$ 을 사용하여 생성하였다 하자. 이 유도과정에서 차례로 규칙 $r_1, \cdots, r_l$ 이 사용되었다면 문장 ω 의 생성확률은 다음과 같다.

$$P(\omega) = \prod_{i=1}^{i=l} P(r_i)$$

이때 규칙 열 $< r_1, \cdots, r_l >$를 유도(derivation) 또는 파스(parse) 라 부른다.[1] 하나의 유도(과정)즉 파스는 바로 하나의 파스트리와 일대일로 대응된다. 파스 $D = < r_1, \cdots, r_l >$ 의 확률은 $P(D) = \prod_{i=1}^{i=l} P(r_i)$ 로 정의된다.

문장 ω 의 가능한 파스(유도)가 한 개이면 파스의 확률과 문장의 생성 확률은 같다. 그러나 자연어에서는 애매성으로 인하여 한 문장에 대하여 두 개 이상의 파스가 있을 수 있다. 문장 ω 에 대하여 가능한 모든 파스의 집합이 $\{ D_1, \cdots, D_m \}$ 이라 하자 ($m > 1$). 이 경우 문장의 확률은 다음과 같다.

$$P(\omega) = \sum_{i=1}^{m} P(D_i)$$

문장을 구문분석한 결과 여러 개의 파스가 나온 경우 그 중 하나를 선택하여야 한다. 이것은 애매성을 해소하는 것이다. 애매성 해소는 자연어 처리 분야에서 해결하기 힘든 문제로 남아 있다. 통계기반 확률문법을 사용하는 주요한 장점의 하나는 애매성의 해소를 위한 중요한 정보를 제공한다는 것이다. 즉 문장 ω 에 대하여 파싱을 수행한 결과 여러 개의 파스 $\{ D_1, \cdots, D_m \}$ 가 나왔다 하자. 이 경우 애매성을 해소하는 기법은 다음과 같이 최대확률을 가진 파스를 선택하는 것이다.

$$D_k = ARGMAX_{D_i} P(D_i)$$

2.2 인사이드-아웃사이드(Inside-Outside) 알고리즘

이 알고리즘은 Baker 에 의하여 맨 처음 소개된 것으로서[2] 규칙의 확률을 추정(estimation)하는 *지금까지 알려진 유일한 알고리즘이다.* 이 알고리즘은 문맥자유문법 중 촘스키 정규형 (Chomsky normal form)을 가진 문법에만 적용이 가능하다. 즉 문법의 각 규칙은 다음의 형태만 가능하다.

$$X\rightarrow YZ \text{ (단, } X, Y \in \Sigma, Y\neq S, Z\neq S \text{)}$$

또는

$$X\rightarrow a \text{ (단, } a \in \Sigma \text{)}$$

먼저 인사이드(inside) 확률을 정의하자. 입력문장이 $w = a_1 \cdots a_n$ 이라 하자. 인사이드 확률 $I(i,j,X)$ 는 i 번째 단어에서 j 번째 단어까지의 부스트링(substring)을 생성할 확률이다.

$$I(i,j,X) = Prob\ (\ X \Rightarrow^* a_i \cdots a_j\ )$$

$I(i,j,X)$ 의 계산은 다음과 같이 순환적(recursion)인 방식으로 가능하다.

$$I(i,i,X) = P(X\rightarrow a_i) \text{ if } X\rightarrow a_i \in R$$
$$0 \qquad \text{otherwise}$$

$$I(i,j,X) = \sum_{X\rightarrow YZ}\sum_{i\le k\le j-1} I(i,k,Y)I(k+1,j,Z)P(X\rightarrow YZ)$$
$$\text{if } i < j$$

위의 식에 의하여 모든 i, j, X 에 대하여 인사이드확률을 구한다.

아웃사이드 확률 $O(i,j,X)$ 는 초기심볼 S 에서 출발하여 부스트링 $a_i \cdots a_j$ 이외의 바깥쪽의 스트링을 생성하고 그 사이에는 비단말기호 X를 생성할 확률을 말한다. 즉,

$$O(i,j,X) = Prob(\ S \Rightarrow^* a_1 \cdots a_{i-1} X\ a_{j+1} \cdots a_n\ )$$

따라서 아웃사이드확률은 다음과 같이 계산될 수 있다.

$$O(1,n,X) = \begin{matrix} 1 & \text{if } X = S \\ 0 & \text{otherwise} \end{matrix}$$

$$O(i,j,X) = \sum_{Y\rightarrow ZX}\sum_{1\le k\le i-1} O(k,j,Y)I(k,i-1,Z)P(Y\rightarrow ZX)$$
$$+ \sum_{Y\rightarrow XZ}\sum_{j+1\le k\le T} O(i,k,Y)I(j+1,k,Z)P(Y\rightarrow XZ)$$

아웃사이드 확률의 계산에는 인사이드확률이 사용되므로 *아웃사이드확률을 계산하기 전에 먼저* 모든 인사이드확률을 계산하여 놓아야 한다.

모든 인사이드, 아웃사이드확률이 구해진 상황에서 규칙 $X\rightarrow YZ$ 의 확률은 다음과 같이 재추정(reestimation)된다.

1) 여기서 유도는 엄밀히 말하면 좌우선유도(leftmost derivation)를 의미한다. 그렇지 않으면 하나의 파스트리에 대하여 여러개의 유도가 가능하기 때문이다.

$$P(X\rightarrow YZ) = \frac{\sum_{1\le i(j)\le n}[O(i,j,X)P(X\rightarrow YZ)\sum_{i\le k\le j-1}I(i,k,Y)I(k+1,j,Z)]}{\sum_{1\le i\le j\le n}O(i,j,X)I(i,j,X)}$$

$$P(X\rightarrow a) = \frac{\sum_{k\ s.t.\ a_k=a}I(k,k,X)O(k,k,X)}{\sum_{1\le i\le j\le n}O(i,j,X)I(i,j,X)}$$

위의 두 식은 한 개의 문장만을 이용하여 규칙확률을 재추정한 것이다. 여러 문장을 가진 말뭉치를 이용하여 재추정할 경우에는 각 문장마다에 대하여 위식의 분모, 분자를 구한 다음 분자는 분자끼리 합한 것을 분모는 분모끼리 합한 것으로 나누어 재추정된 규칙확률을 구한다. 재추정된 규칙확률을 이용하여 다시 재추정 작업을 수행한다. 이와 같이 재추정의 반복 작업을 훈련말뭉치를 생성할 확률값의 변화가 정해진 임계값(threshold) 이하가 될 때까지 수행한다.

3 새로운 규칙확률 재추정 알고리즘

3.1 기본 개념에 의한 확률 재추정 기법

앞에서 설명한 바와 같이 규칙 $A\rightarrow a$ 의 확률 $P(A\rightarrow a)$ 는 문법에 의하여 문장을 생성하는 데 있어서 비단말기호 A 를 확장할 때 이 규칙 $A\rightarrow a$ 을 사용할 확률을 나타낸다 [4]. 따라서

$$P(A\rightarrow a) = \frac{\text{number of times } A\rightarrow a \text{ is used in } A\text{'s expansion}}{\text{number of times } A \text{ is expanded}}$$

어떤 문장 w 에 대한 파스가 한 개라고 하자. 이 경우 특정 규칙 $A\rightarrow a$ 를 사용한 횟수는 이 파스에서 이 규칙이 나타난 횟수(count)이다. 만약 문장에 대한 파스가 여러 개인 경우라면 문장 w 를 생성하는 데 있어서 $A\rightarrow a$ 가 사용된 사용횟수기대치 (expected usage count)를 구하여야 한다. 문장 w 가 여러 개의 파스 $D_1,\cdots, D_m$ 를 가지고 있다 하자. 각 파스 D_i 에서 규칙 $A\rightarrow a$ 가 사용된 횟수를 $C(D_i, A\rightarrow a)$ 라 하자. 규칙 $A\rightarrow a$ 의 사용횟수기대치는 각 파스의 확률을 고려하여야 한다. 파스 D_i 에 대해 다음과 같은 가중치를 생각한다.

$$W(D_i) = \frac{P(D_i)}{P(w)}$$

(단, $P(w) = \sum_{i=1}^{m} P(D_i)$)

$C(D_i, A\rightarrow a)$ 와 가중치 $W(D_i)$를 곱한 값을 파스 D_i 에 대한 상대사용횟수(relative usage count) $RC(D_i, A\rightarrow a)$ 라 하자.

$$RC(D_i, A\rightarrow a) = W(D_i)\,C(D_i, A\rightarrow a) = \frac{P(D_i)}{P(w)}\,C(D_i, A\rightarrow a)$$

여기에서 정규화되지(normalized) 않은 가중사용횟수 $WC(D_i, A\rightarrow a)$ 를 다음과 같이 정의한다. 이 개념은 다음 3.2 절에서 이용된다.

$$WC(D_i, A\rightarrow a) = P(D_i)C(D_i, A\rightarrow a)$$

문장 w의 생성에서 $A\rightarrow a$ 가 사용된 사용횟수 기대치 $E(w, A\rightarrow a)$ 는 모든 파스의 상대사용횟수를 합한 값이 된다. 즉

$$E(w, A\rightarrow a) = \sum_{i=1}^{m} RC(D_i, A\rightarrow a)$$

문장 w 에서 비단말기호 A 가 확장된 횟수는 모든 A-규칙이 사용된 횟수의 합과 같다. 따라서 문장 w 에서 규칙의 사용된 횟수에 기반한 규칙확률의 재추정값은 다음과 같다.

$$P(A\rightarrow a) = \frac{E(w, A\rightarrow a)}{\sum_{\gamma\ s.t.\ A\rightarrow\gamma\in R} E(w, A\rightarrow\gamma)}$$

이 식은 한 개의 문장만을 이용한 경우이므로 훈련말뭉치 $\Omega = \{ w^1, \ldots, w^Q\}$를 이용할 경우에는 각 문장 w^q 를 이용하여 기대사용횟수 $E_q(w^q, A\rightarrow a)$ 를 구한다. 결국 전체 말뭉치를 이용한 규칙확률 재추정값은 다음과 같이 된다 :

$$P(A\rightarrow a) = \frac{\sum_{q=1}^{Q} E_q(w^q, A\rightarrow a)}{\sum_{q=1}^{Q}\sum_{\gamma\ s.t.\ A\rightarrow\gamma\in R} E_q(w^q, A\rightarrow\gamma)} \quad (1)$$

지금까지의 방법에 의한 확률 재추정 기법을 정리하면 다음과 같은 알고리즘을 얻는다 [4] :

1. Assign initial probability values for all rules ;
2. Repeat
3. For each sentence w^q in training corpus Ω do
   - 3.1 Obtain parses $D_1,\cdots, D_m$ for w^q ;
   - 3.2 Compute $P(w^q)$;
   - 3.3 For i = 1 to m do
     - 2.3.1 Obtain $C(D_i, A\rightarrow a)$ for all rules $A\rightarrow a$;
     - 2.3.2 Compute $RC(D_i, A\rightarrow a)$ for all rules $A\rightarrow a$;
   - 3.4 Compute $E_q(w^q, A\rightarrow a)$ for all rules $A\rightarrow a$;

4. Update $P(A \rightarrow \alpha)$
   for all rules $A \rightarrow \alpha$ using eq. (1) ;
5. Until change in P(Ω)= $\Pi_{q=1}^{Q} P(w^q)$
   $\leq$ given threshold ;

3.2 규칙확률 재추정 알고리즘

위의 알고리즘은 개념적으로 이해하기가 쉽다. 그러나 위 알고리즘을 그대로 이용하는 데는 문제가 있다. 그것은 속도 면에서 지수함수적인 시간복잡도(time complexity)를 가질 수 있기 때문이다. 그 원인은 문장에 대한 가능한 파스의 수는 문장의 길이에 대하여 지수함수 값을 가질 수 있기 때문이다. 즉 위 알고리즘의 단계 3에서 m 의 값이 문장 w 의 길이 n 의 지수함수 값이 될 수 있다. 즉 위의 알고리즘의 시간 복잡도는 $O(2^n)$ 이 된다.

여기서 우리는 위의 알고리즘을 보다 효율적으로 만드는 기법을 소개한다. 주어진 문법이 촘스키정규형이라 가정하자. 우리는 문장을 파싱하는 과정에서 확률의 계산에 필요한 정보인 규칙의 사용횟수를 점진적으로 준비하여 가는 방법을 택한다. 이 방법은 CYK 파싱알고리즘에 기반을 두고 있다 [1]. 이 알고리즘은 테이블 t 를 이용한다. 셀 $t_{i,j}$ 에 들어 있는 비단말기호 A 는 루트레이블이 A 인 여러 개의 트리를 나타낸다. 즉 그 트리들의 루트가 A 가 된다. $t_{i,j}$ 에 A 가 들어갈 조건은 다음과 같다.

$$t_{i,j} \ni A \text{ if and only if } A \Rightarrow^* a_i \cdots a_{i+j-1}$$

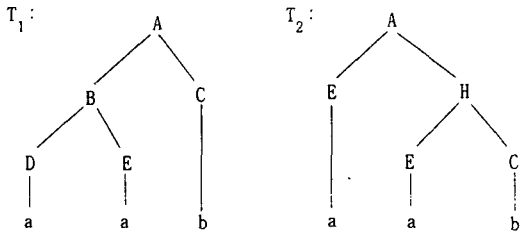
즉 A 가 a_i 에서 시작하여 길이가 j 인 부스트링을 생성한다면 A 를 $t_{i,j}$ 에 넣는다. 이것은 다음과 같은 조건이 성립하면 가능하다:

$$t_{i,k} \ni B \text{ 이고 } t_{i+k,j-k} \ni C \text{ 이고 } A \rightarrow BC \in R \text{ 라면 } A \text{ 를 } t_{i,j} \text{ 에 넣는다.}$$

결국 t 의 하나의 셀 안에 있는 비단말 기호는 문장의 일부를 생성하는 부파스트리(sub-parse tree)의 루트노드의 레이블이 되며 이 부파스트리를 나타낸다고 볼 수 있다.

그러나 A 를 $t_{i,j}$ 에 넣으려 할 때 $t_{i,j}$ 에 이미 A 가 존재할 수도 있다. 이것은 $a_i \cdots a_{i+j-1}$ 를 생성하는 A를 루트로 하는 다른 부파스트리가 이전에 발견되어 입력되었기 것이다. 이때 A 를 $t_{i,j}$ 에 중복하여 넣지 않고 이미 존재하는 A가 새로이 발견된 부파스트리도 함께 나타내도록 한다. (이 점이 바로 CYK 알고리즘에게 선형적인 시간복잡도 $O(n^3)$ 를 가지도록 보장하여 준다.) 이는 결국 $t_{i,j}$ 에 존재하는 A 는 여러 개의 부파스트리를 대변할 수도 있음을 나타낸다.

예를 들어 다음 그림이 입력스트링의 일부인 $a_3\ a_4\ a_5$ = aab 에 대한 두 개의 부파스트리를 나타낸다고 하자.



먼저 부파스트리 T_1 을 발견하면 $t_{3,3}$ 에는 A 를 넣는다. 나중에 부파스트리 T_2 를 발견하여 A를 넣고자 $t_{3,3}$ 을 조사하면 이미 A 가 존재하므로 새로이 중복하여 넣지는 않는다. 대신에 이미 존재하는 A 가 위의 두 개의 부파스트리를 함께 나타내도록 한다.

우리 알고리즘의 핵심은 CYK 테이블 t 의 각 셀의 각 비단말기호 A 에 다음과 같은 두가지의 값을 붙여 놓는것이다.

(1) 심볼 A 가 나타내는 부파스트리들의 확률의 합, $\lambda(A)$. 즉,

$$\lambda(A) = \sum_{i=1}^{m} P(T_i)$$

단, $T_1, \cdots, T_m$ 는 A 가 나타내는 부파스트리들임

(2) 심볼 A 에 대한 각 부파스트리들마다 사용된 모든 규칙의 비정규화된 가중사용횟수 (weighted usage count) 를 구한다. 그리고 이를 A 의 모든 부파스트리들에 대하여 합한 값을 구하여 붙여 놓는다. 이를 이 비정규화 가중 사용횟수의 합 (sum of weighted usage count; SWC)이라 부르자. 즉 $SWC(A, r)$ 는 규칙 r 이 심볼 A 가 나타내는 모든 부파스트리에서 사용된 비정규화된 가중 사용횟수의 합을 나타낸다. 즉,

$$SWC(A, r) = \sum_{i=1}^{m} WC(T_i, r)$$

여기서 $T_1, \cdots, T_m$ 는 심볼 A 가 나타내는 부파스트리들이며 $WC(T_i, r)$ 는 규칙 r 이 부파스트리 T_i 에서 사용된 비정규화된 가중사용횟수이다.

예로서 위의 그림에 대하여 λ 와 SWC 값을 구

하여 보자. 각 규칙의 확률이 다음과 같이 주어졌다 하자.

$P(A\rightarrow BC) = 0.4$ $P(A\rightarrow EH) = 0.6$
$P(B\rightarrow DE) = 0.3$ $P(H\rightarrow EC) = 0.2$
$P(D\rightarrow a) = 0.1$ $P(E\rightarrow a) = 0.3$
$P(C\rightarrow b) = 0.15$

$t_{3,3}$ 내의 심볼 A 에게 $\lambda(A)$, $SWC(A, A\rightarrow BC)$, $SWC(A, A\rightarrow EH)$, $SWC(A, H\rightarrow EC)$, ..., 등을 붙이게 된다. 이들의 계산은 다음과 같다:

$$\lambda(A) = P(T_1) + P(T_2) =$$
$$P(A\rightarrow BC)P(B\rightarrow DE)P(D\rightarrow a)P(E\rightarrow a)P(C\rightarrow b)$$
$$+ P(A\rightarrow EH)P(H\rightarrow EC)\,P(E\rightarrow a)^2 P(C\rightarrow b)$$
$$= 0.00054 + 0.00162 = 0.00216$$
$$SWC(A, E\rightarrow a) =$$
$$P(T_1)C(T_1, E\rightarrow a) + P(T_2)C(T_2, E\rightarrow a)$$
$$= 0.00054 \times 1 + 0.00162 \times 2 = 0.00378$$

CYK 파싱 테이블의 모든 셀의 모든 비단말기호에 대하여 λ 값과 SWC 값이 계산되었다고 하자. 그러면 규칙확률은 다음과 같은 식에 의하여 계산된다 (단, 여기에서 S 는 $t_{1,n}$ 안에 있는 심볼임).

$$P(w) = \lambda(S)$$
$$E(w, A\rightarrow\alpha) = \frac{SWC(S, A\rightarrow\alpha)}{\lambda(S)} \quad (2)$$

위의 두 식으로 부터 우리는 알 수 있는 사실은 만약 CYK 테이블의 셀안의 각 비단말기호에 대하여 λ 값과 SWC 값을 구하는 효율적인 방법을 구할 수만 있다면 규칙의 사용횟수기대치를 구할 수 있다는 점이다.

본 논문의 핵심은 아래와 같이 하면 λ 값과 SWC 값을 효율적으로 구할 수 있다는 것이다. CYK 파싱 테이블의 셀의 비단말기호에 대한 λ 값과 SWC 값은 CYK 파싱 알고리즘이 진행되면서 셀에 비단말 기호를 넣거나 넣으려고 시도할 때마다 다음 방식에 따라 계산하여 붙여 주거나 갱신한다.

● 만약 $B\rightarrow a \in R$ 이고 $a = a_i$ 이면 B 를 $t_{i,1}$ 에 삽입한다. 이때 이 B 에 다음 λ, SWC 값을 붙인다.

$\lambda(B) = P(B\rightarrow a)$,
$SWC(B, B\rightarrow a) = P(B\rightarrow a)$

● 만약 $t_{i,k} \ni B$ 이고 $t_{i+k,j-k} \ni C$ 이며 $A\rightarrow BC \in R$ 라면 A 를 $t_{i,j}$ 에 넣는다. 이때,

(i) $\lambda(A) = \lambda(A) + \lambda(B)\,\lambda(C)\,P(A\rightarrow BC)$

(ii) 규칙 r 이 B나 C에서 0이 아닌 SWC값을 가지고 있거나 또는 규칙 $A\rightarrow BC$라면, A에 r의 SWC값을 다음 방법으로 계산하여 붙인다.

$$SWC(A, r) = SWC(A, r)$$
$$+ \delta(r, A\rightarrow BC)\,\lambda(B)\,\lambda(C)\,P(A\rightarrow BC)$$
$$+ SWC(B, r)\,\lambda(C)\,P(A\rightarrow BC)$$
$$+ SWC(C, r)\,\lambda(B)\,P(A\rightarrow BC) \quad \text{where}$$
$$\delta(r, A\rightarrow BC) = \begin{cases} 1 & \text{if } r = A\rightarrow BC \\ 0 & \text{otherwise.} \end{cases}$$

(단, $\lambda(A)$ 와 $SWC(A, r)$ 의 초기값은 0 이다. $SWC(A, r)$ 이 0 이면 A노드에 이값을 붙이지 않는다. 즉 0 이 아닌 SWC 값만 붙여 놓는다.)

우리는 CYK 파싱 알고리즘을 이용하여 다음과 같이 셀내의 모든 비단말기호의 λ 값과 SWC 값을 효율적으로 계산할 수 있다.

```
(1) for i = 1 to n do
      if B→a ∈ R and a = a_i then
        begin add B to t_{i,1} ;
         attach λ(B) and SWC(B, B→a) to B ;
        end;
(2) for j = 2 to n do
      for i = 1 to n do
        if i+j-1 ≤ n then
          for k = 1 to j-1 do
            if t_{i,k} ∋ B
              and t_{i+k,j-k} ∋ C
              and A→BC ∈ R then
              begin add A to t_{i,j}
                    if A ∉ t_{i,j}
                  attach λ(A) and
                  SWC(A, r) to A;
    end;
(3) Compute E_q(w^q, A→α) by eq. (2)
      for each rule A→α in R ;
```

3.1 절에 소개한 기본 개념에 입각한 알고리즘의 3.1-3.4 부분을 각 문장 w^q 에 대하여 $E_q(w^q, A\rightarrow\alpha)$ 를 구하는 위의 방법으로 구현하므로서 우리의 규칙 확률 재추정 알고리즘을 얻는다.

4 실험 결과

인사이드-아웃사이드 알고리즘과 우리의 알고리즘과의 속도를 비교하기 위하여 두 알고리즘을 구현하였다. 그리고 동일 데이터를 사용하여 그 성능을 비교하였다. 우리의 알고리즘을 구현하는데 있어 주의할 점은 트리에 나타난 규칙에 대해서만 SWC 값을 매달아 놓아야 한다 (셀의 비단말기호

가 나타내는 파스트리에 한번이라도 나타난 규칙에 대해서만 SWC 값을 이 비단말기호가 가지는 연결리스트(linked list) 에 저장한다.) 동일한 문법과 동일한 훈련용 입력 문장 집합에 대하여 두 알고리즘은 동일한 규칙확률을 추정하였다. 이것은 우리의 알고리즘이 올바른 알고리즘임을 나타낸다.

다음 표는 실험 결과를 나타낸다. 수행 시간 (run time)을 비교한 결과 본 논문에서 제안된 알고리즘이 인사이드-아웃사이드 알고리즘보다 2-4 배 정도 빠르다. 실험에 따르면 문법의 종류에 따라 속도의 향상이 크게 차이가 난다.

	평균수행시간 (Clk/문장)		
	인사이드-아웃사이드알고리즘 (A)	제안된 알고리즘 (B)	속도비 (A/B)
문법1	179.52	72.33	2.5
문법2	111.25	46.27	2.4
문법3	363.33	100.23	3.6

5 결론

본 논문에서는 확률문맥자유문법의 규칙의 확률을 추정하는 새로운 방법을 제안하였다. 제안된 새로운 기법은 지금까지 알려진 유일한 기법인 인사이드-아웃사이드 알고리즘에 비하여 장점을 가지고 있다. 제안된 알고리즘은 규칙확률의 밑바탕이 되는 개념을 그대로 적용하여 얻은 것으로 개념적으로 이해하기 쉽다. 따라서 알고리즘을 구현한 프로그램도 훨씬 간단하여진다. 실험 결과 제안된 알고리즘이 인사이드-아웃사이드 알고리즘보다 2에서 4 배 정도 더 빠른 것으로 나타났다.

참 고 문 헌

[1] A.V. Aho and J.D. Ullman, The Theory of Parsing, Translation, and Compiling, Vol. 1: Parsing, (Prentice Hall, Englood Cliffs, NJ, 1972).

[2] J. K. Baker, Trainable grammars for speech recognition, Speech Communication Papers for the 97th meeting of the Acoustical Society of America, (J. J. Wolf and D. H. Klatt. eds.) (1979) 547-550.

[3] T. L. Booth and R. A. Thompson, Applying probability measures to abstract languages, IEEE Trans. Comput. C-22(5) (1973) 442-450.

[4] T. Fusisaki, F. Jelinek, J. Cocke, E. Black and T. Nishino, A probabilistic parsing method for sentence disambiguation, Current Issues in Parsing Technology, (M. Tomita. eds.), Kluwer Academic Publishers (1991), 139-152.

[5] F. Jelinek, J. D. Lafferty and R. L. Mercer, Basic methods of probabilistic context-free grammars, Speech Recognition and Understanding:Recent Advances, Trends and Applications, F75, NATO ASI Series, (P. Laface and R.De Mori. eds.) Springer Verlag, (1992) 345-360.

[6] K. Lari and S. J. Young, The estimation of stochastic context-free grammars using inside-outside algorithm, Computer Speech and Language 5 (1990) 35-56.