Stochastic Multiple Context-Free Grammar for RNA Pseudoknot Modeling

Yuki Kato and Hiroyuki Seki

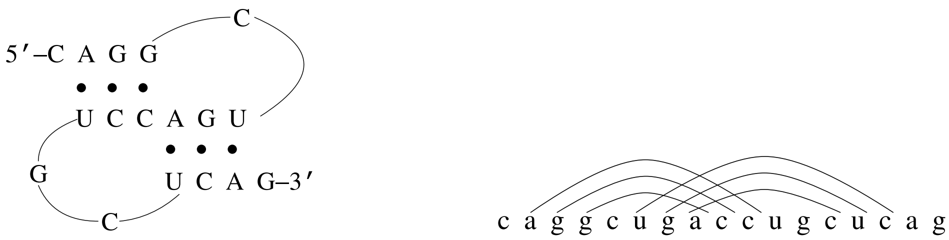
Abstract

Several formal grammars have been proposed for modeling RNA secondary structure including substructure called pseudoknot. In this paper, we focus on multiple context-free grammars (MCFGs), which are natural extension of context-free grammars and can represent pseudoknots, and extend a specific subclass of MCFGs to a probabilistic model called a stochastic MCFG (SMCFG). We design a polynomial time parsing algorithm for finding the most probable derivation tree and a probability parameter estimation method based on the EM (expectation maximization) algorithm. Also, we show some experimental results of RNA secondary structure prediction with pseudoknots using SMCFG algorithm.

1 Introduction

Non-coding RNAs fold into characteristic structures determined by interactions between mostly Watson-Crick complementary base pairs. Such a base paired structure is called the *secondary structure*. *Pseudoknot* (Figure 1 (a)) is one of the typical substructures found in the secondary structures of several RNAs, including rRNAs, tmRNAs and viral RNAs. An alternative graphic representation of a pseudoknot is arc depiction where arcs connect base pairs (Figure 1 (b)). It has been recognized that pseudoknots play an important role in RNA functions such as ribosomal frameshifting and regulation of translation.

Many attempts have so far been made at modeling RNA secondary structure by formal grammars in order to analyze RNA secondary structure. In a grammatical approach, an application to secondary structure prediction can be made by designing and implementing a parsing algorithm for the grammar. Eddy and Durbin [5], and Sakakibara et al. [14] modeled RNA secondary structure without pseudoknots by using stochastic context-free grammars (stochastic CFGs, SCFGs). For pseudoknotted structure, however, another approach has to be taken since a single CFG cannot represent crossing dependencies of base pairs in pseudoknots (Figure 1 (b)) for lack of generative power. Brown and Wilson [2] proposed a model



(a) Pseudoknot

(b) Arc depiction of (a)

Figure 1: Example of RNA secondary structure

based on intersections of SCFGs to describe RNA pseudoknots. Cai et al. [3] introduced a model based on parallel communication grammar systems using a single CFG synchronized with a number of regular grammars to represent pseudoknots. Akutsu [1] provided dynamic programming algorithms for RNA pseudoknot prediction without using grammars. On the other hand, several grammars have been proposed where the grammar itself can fully describe pseudoknotted structure. Rivas and Eddy [12, 13] provided a dynamic programming algorithm for predicting RNA secondary structure including pseudoknots, and introduced a new class of grammars called RNA pseudoknot grammars (RPGs) for deriving sequences with gap. Uemura et al. [16] defined specific subclasses of tree adjoining grammars (TAGs) named SL-TAGs and extended SL-TAGs (ESL-TAGs) respectively, and predicted RNA pseudoknots by using parsing algorithm of ESL-TAG. Matsui et al. [11] proposed pair stochastic tree adjoining grammars (PSTAGs) based on ESL-TAGs and tree automata for aligning and predicting pseudoknots, which showed good prediction accuracy. These grammars have generative power stronger than CFGs and polynomial time algorithms for parsing problem.

In our previous work [9], we identified RPGs, SL-TAGs and ESL-TAGs as subclasses of *multiple context-free grammars* (MCFGs) [7, 8, 15], which can model RNA pseudoknots, and showed a candidate subclass of the minimum grammars for representing pseudoknots. The generative power of MCFGs is stronger than that of CFGs and MCFGs have a polynomial time parsing algorithm like the CYK (Cocke-Younger-Kasami) algorithm for CFGs. In this paper, we extend the above candidate subclass of MCFGs to a probabilistic model called a stochastic MCFG (SMCFG). We present a polynomial time parsing algorithm for finding the most probable derivation tree, which is applicable to RNA secondary structure prediction including pseudoknots. In addition, we mention a probability parameter estimation method based on the EM (expectation maximization) algorithm. Finally, we show some experimental results of pseudoknot prediction for two RNA families using SMCFG algorithm, which show very good prediction accuracy.

2 Multiple Context-Free Grammar

For an alphabet Σ, let Σ^* denote the set of all finite sequences over Σ. The empty sequence is denoted by ε. For a sequence $w \in \Sigma^*$, let $|w|$ denote the length of w, that is, the number of symbols occurring in w.

A *multiple context-free grammar* (MCFG) [7, 8, 15] is a 5-tuple $G = (N, T, F, P, S)$ where N is a finite set of nonterminals, T is a finite set of terminals, F is a finite set of functions, P is a finite set of (production) rules and $S \in N$ is the start symbol. For each $A \in N$, a positive integer denoted by $\dim(A)$ is given and A derives $\dim(A)$-tuples of terminal sequences. For the start symbol S, $\dim(S) = 1$. For each $f \in F$, positive integers d_i $(0 \le i \le k)$ are given and f is a total function from $(T^*)^{d_1} \times \cdots \times (T^*)^{d_k}$ to $(T^*)^{d_0}$ satisfying the following condition (F):

(F) Let $\overline{x_i} = (x_{i1}, \ldots, x_{id_i})$ denote the ith argument of f for $1 \le i \le k$. The hth component of the function value for $1 \le h \le d_0$, denoted by $f^{[h]}$, is defined as

$$f^{[h]}[\overline{x_1}, \ldots, \overline{x_k}] = \beta_{h0} z_{h1} \beta_{h1} z_{h2} \cdots z_{hv_h} \beta_{hv_h} \qquad (*)$$

where $\beta_{hl} \in T^*$ $(0 \le l \le v_h)$ and $z_{hl} \in \{x_{ij} \mid 1 \le i \le k,\ 1 \le j \le d_i\}$ $(1 \le l \le v_h)$. The total number of occurrences of x_{ij} in the right hand sides of $(*)$ from $h = 1$ through d_0 is at most one.

Each rule in P has the form of $A_0 \to f[A_1, \ldots, A_k]$ where $A_i \in N$ $(0 \le i \le k)$ and $f : (T^*)^{\dim(A_1)} \times \cdots \times (T^*)^{\dim(A_k)} \to (T^*)^{\dim(A_0)} \in F$. If $k \ge 1$, then the rule is called a *nonterminating rule*, and if $k = 0$, then it is called a *terminating rule*. A terminating rule $A_0 \to f[\ ]$ with $f^{[h]}[\ ] = \beta_h$ $(1 \le h \le \dim(A_0))$ is simply written as $A_0 \to (\beta_1, \ldots, \beta_{\dim(A_0)})$.

Example 1. (1) Let $G_1 = (N_1, T_1, F_1, P_1, S)$ be an MCFG where $N_1 = \{S, A\}$, $T_1 = \{a, b\}$ and $P_1 = \{S \to J[A],\ A \to f_a[A] \mid f_b[A] \mid (\varepsilon, \varepsilon)\}$ where $\dim(S) = 1,\ \dim(A) = 2,\ J[(x_1, x_2)] = x_1 x_2$ and $f_\alpha[(x_1, x_2)] = (\alpha x_1, \alpha x_2)$ with $\alpha = a, b$.

(2) Let $G_2 = (N_2, T_2, F_2, P_2, S)$ be an MCFG where $N_2 = \{S, A\}$, $T_2 = \{a_i \mid 1 \le i \le 2m\}$ and $P_2 = \{S \to J_m[A],\ A \to g[A] \mid (\varepsilon, \ldots, \varepsilon)\}$ where $\dim(S) = 1,\ \dim(A) = m,\ J_m[(x_1, \ldots, x_m)] = x_1 \cdots x_m$ and $g[(x_1, \ldots, x_m)] = (a_1 x_1 a_2, \ldots, a_{2m-1} x_m a_{2m})$. $\square$

For a function f defined by $(*)$ in condition (F) and tuples of terminal sequences $\overline{\alpha_i} = (\alpha_{i1}, \ldots, \alpha_{id_i}) \in (T^*)^{d_i}$ $(1 \le i \le k)$, let $f[\overline{\alpha_1}, \ldots, \overline{\alpha_k}]$ denote the tuple of terminal sequences obtained from the right hand sides of $(*)$ by substituting α_{ij} $(1 \le i \le k,\ 1 \le j \le d_i)$ into x_{ij}. For instance, $f_a[(bba, ab)] = (abba, aab)$ in Example 1 (1). We recursively define the relation $\stackrel{*}{\Rightarrow}$ by the following (L1) and (L2):

(L1) If $A \to \overline{\alpha} \in P$ ($\overline{\alpha} \in (T^*)^{\dim(A)}$), then we write $A \overset{*}{\Rightarrow} \overline{\alpha}$.

(L2) If $A \to f[A_1, \ldots, A_k] \in P$ and $A_i \overset{*}{\Rightarrow} \overline{\alpha_i}$ ($1 \leq i \leq k$), then we write $A \overset{*}{\Rightarrow} f[\overline{\alpha_1}, \ldots, \overline{\alpha_k}]$.

The language generated by an MCFG G is defined as $L(G) = \{w \in T^* \mid S \overset{*}{\Rightarrow} w\}$. In parallel with the relation $\overset{*}{\Rightarrow}$, we recursively define derivation trees as follows:

(D1) If $A \to \overline{\alpha} \in P$ ($\overline{\alpha} \in (T^*)^{\dim(A)}$), then a derivation tree for $\overline{\alpha}$ is the tree with the root labeled A which has $\overline{\alpha}$ as the only one child.

(D2) If $A \to f[A_1, \ldots, A_k] \in P$, $A_i \overset{*}{\Rightarrow} \overline{\alpha_i}$ ($1 \leq i \leq k$) and $t_1, \ldots, t_k$ are derivation trees for $\overline{\alpha_1}, \ldots, \overline{\alpha_k}$, then a derivation tree for $f[\overline{\alpha_1}, \ldots, \overline{\alpha_k}]$ is the tree with the root labeled A (or $A : f$ if necessary) which has $t_1, \ldots, t_k$ as (immediate) subtrees from left to right.

Example 1 (continued). (1) By (L1), $A \overset{*}{\Rightarrow} (\varepsilon, \varepsilon)$ since $A \to (\varepsilon, \varepsilon) \in P$. Since $f_a[(\varepsilon, \varepsilon)] = (a, a)$ and $f_b[(a, a)] = (ba, ba)$, we have $A \overset{*}{\Rightarrow} (a, a)$ and $A \overset{*}{\Rightarrow} (ba, ba)$ by (L2). Also by $S \to J[A]$, $S \overset{*}{\Rightarrow} J[(ba, ba)] = baba$. In fact, $L(G_1) = \{ww \mid w \in \{a, b\}^*\}$.

(2) Likewise, $A \overset{*}{\Rightarrow} (\varepsilon, \ldots, \varepsilon)$ by (L1), $A \overset{*}{\Rightarrow} f[(\varepsilon, \ldots, \varepsilon)] = (a_1 a_2, \ldots, a_{2m-1} a_{2m})$ by (L2), etc. This tells us that $L(G_2) = \{a_1^n \cdots a_{2m}^n \mid n \geq 0\}$. $\square$

To introduce subclasses of MCFGs, we define a few terminologies. Let $G = (N, T, F, P, S)$ be an arbitrary MCFG. For a function $f : (T^*)^{d_1} \times \cdots \times (T^*)^{d_k} \to (T^*)^{d_0}$, let $\dim(f) = d_0$, $\text{rank}(f) = k$ and $\deg(f) = \Sigma_{j=0}^{k} d_j$, which are called the *dimension* of f, the *rank* of f and the *degree* of f respectively. Next, let us define the dimension, rank and degree of G as $\dim(G) = \max\{\dim(f) \mid f \in F\}$, $\text{rank}(G) = \max\{\text{rank}(f) \mid f \in F\}$ and $\deg(G) = \max\{\deg(f) \mid f \in F\}$ respectively. By definition, $\deg(G) \leq \dim(G)(\text{rank}(G) + 1)$. With these parameters, we define subclasses of MCFGs. An MCFG G with $\dim(G) \leq m$ and $\text{rank}(G) \leq r$ is called an (m, r)-MCFG. Likewise, an MCFG G with $\dim(G) \leq m$ is called an m-MCFG.

In our previous work [9], we have shown that a candidate subclass of the minimum grammars which can represent repeating pseudoknots is the class of $(2, 2)$-MCFGs with degree five or less. In the following, we will focus on a $(2, 2)$-MCFG with degree five or less and extend it to a probabilistic model in the next section.

Example 2. Consider a $(2, 1)$-MCFG $G_3 = (\{S, A, X_1, X_2\}, \{a, c, g, u\}, F_3, P_3, S)$ for generating RNA sequences where P_3 and F_3 are as follows:

$$
\begin{array}{ll}
S \to J[A], & J[(x_1, x_2)] = x_1 x_2, \\
A \to UP^{\alpha}_{1L}[A],\ X_1 \to UP^{\alpha}_{1L}[A], & UP^{\alpha}_{1L}[(x_1, x_2)] = (\alpha x_1, x_2), \\
A \to UP^{\alpha}_{1R}[A] \mid UP^{\alpha}_{1R}[X_1], & UP^{\alpha}_{1R}[(x_1, x_2)] = (x_1 \alpha, x_2), \\
A \to UP^{\alpha}_{2L}[A],\ X_2 \to UP^{\alpha}_{2L}[A], & UP^{\alpha}_{2L}[(x_1, x_2)] = (x_1, \alpha x_2), \\
A \to UP^{\alpha}_{2R}[A] \mid UP^{\alpha}_{2R}[X_2], & UP^{\alpha}_{2R}[(x_1, x_2)] = (x_1, x_2 \alpha), \\
A \to BP^{\alpha\beta}[A], & BP^{\alpha\beta}[(x_1, x_2)] = (\alpha x_1, x_2 \beta), \\
A \to (\varepsilon, \varepsilon). &
\end{array}
$$

Note that $\alpha \in \{a, c, g, u\}$ and $(\alpha, \beta) \in \{(a, u), (u, a), (c, g), (g, c)\}$. Functions have mnemonic names where UP and BP stand for unpair and base pair respectively. The RNA sequence agacuu in Figure 2 can be generated by the above rules as follows: $A \stackrel{*}{\Rightarrow} BP^{gc}[(\varepsilon, \varepsilon)] = (g, c)$, $A \stackrel{*}{\Rightarrow} BP^{au}[(g, c)] = (ag, cu)$, $X_2 \stackrel{*}{\Rightarrow} UP^{a}_{2L}[(ag, cu)] = (ag, acu)$, $A \stackrel{*}{\Rightarrow} UP^{u}_{2R}[(ag, acu)] = (ag, acuu)$ and $S \stackrel{*}{\Rightarrow} J[(ag, acuu)] = agacuu$. G_3 has a derivation tree (Figure 3) for agacuu which represents the pseudoknot shown in Figure 2. $\square$

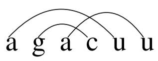


Figure 2: Example of a pseudoknot

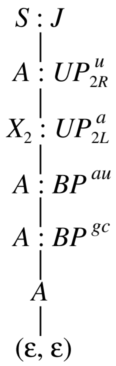


Figure 3: A derivation tree in G_3

3 Stochastic Multiple Context-Free Grammar

First, we briefly review stochastic (probabilistic) grammars. In a stochastic grammar, each rule has an associated probability such that the sum of the probabilities of rules with the same left hand side is one. Sample rules of an SCFG are $S \stackrel{0.4}{\to} SS$ and $S \stackrel{0.6}{\to} ab$ where S is a nonterminal, a and b are terminals and the number above an arrow is a probability that the rule is applied. A stochastic grammar with probability parameters θ generates different sequences w with probabilities $P(w \mid \theta)$ [1] and defines the probability distribution over sequences, that is, $\sum_w P(w \mid \theta) = 1$. Note that a non-stochastic grammar either generates a sequence w or

[1] $P(w \mid \theta)$ represents conditional probability, that is, the probability of w when θ is given.

does not. In the following, we will regard nonterminals as states and applying rules as state transitions for convenience. If the right hand side of a rule contains one or more terminals, the nonterminal in the left hand side can be interpreted to emit symbols.

Next, we extend a subclass of MCFG to a probabilistic model which we call a *stochastic multiple context-free grammar* (SMCFG). Let G_s be a $(2, 2)$-MCFG with $\deg(G_s) \leq 5$. G_s has m different nonterminals denoted by $W_1, \ldots, W_m$, and the rules and functions are shown in Table 1. Each nonterminal W_v $(1 \leq v \leq m)$ has one of the 12 types of the forms of rules which are E, S, D, B_1, B_2, B_3, B_4, U_{1L}, U_{1R}, U_{2L}, U_{2R} and P for indicating end, start, delete, bifurcation, unpair and pair respectively. Note that delete nonterminals are used to deal with gaps in sequence alignment. The type of W_v is denoted by type(v) and we predefine type$(1) = S$, that is, W_1 is the start symbol. We assign a symbol emission probability denoted by $e_v(a_i)$ etc. and state transition probability denoted by $t_v(y)$ to each rule as shown in Table 1. In the table, a_i, a_j, a_k and a_l are the ith, jth, kth and lth terminals in a sequence respectively. All the transition probabilities of bifurcation nonterminals are defined as one since most of the nonterminals for modeling RNA secondary structure have the type of either unpair or pair, and bifurcation nonterminals are sometimes used to deal with concatenating and wrapping operation. This single choice of transition for bifurcation nonterminal reduces time complexities of SMCFG algorithms. For each nonterminal W_v, Δ_v^{1L}, Δ_v^{1R}, Δ_v^{2L} and Δ_v^{2R} are defined as the number of symbols emitted by W_v (see Table 2). This notation simplifies the description of the algorithms presented in the next section. Let $\mathcal{C}_v$ be the set of indices y such that W_v can make a transition to W_y. Let $\mathcal{P}_v$ be the set of indices y such that W_y can make a transition to W_v. To avoid non-emitting cycles, we assume that the nonterminals are numbered such that $v < y$ for all $y \in \mathcal{C}_v$. It is easy to see that G_s can generate RNA sequences corresponding to pseudoknots (see Example 2).

4 Algorithms for SMCFG

In RNA structure analysis using stochastic grammars, we have to deal with the following three problems [4]:

(1) Calculate the optimal alignment of a sequence to a parameterized stochastic grammar. (alignment problem)

(2) Calculate the probability of a sequence given a parameterized stochastic grammar. (scoring problem)

(3) Estimate optimal probability parameters for a stochastic grammar given a set of example sequences. (training problem)

In this section, we give solutions to each problem for the specific SMCFG G_s.

Table 1: SMCFG G_s

Type	Rule	Function	Emission	Transition
E	$W_v \to (\varepsilon, \varepsilon)$		1	1
S	$W_v \to J[W_y]$	$J[(x_1, x_2)] = x_1 x_2$	1	$t_v(y)$
D	$W_v \to SK[W_y]$	$SK[(x_1, x_2)] = (x_1, x_2)$	1	$t_v(y)$
B_1	$W_v \to C_1[W_y, W_z]$	$C_1[x_1, (x_{21}, x_{22})] = (x_1 x_{21}, x_{22})$	1	1
B_2	$W_v \to C_2[W_y, W_z]$	$C_2[x_1, (x_{21}, x_{22})] = (x_{21} x_1, x_{22})$	1	1
B_3	$W_v \to C_3[W_y, W_z]$	$C_3[x_1, (x_{21}, x_{22})] = (x_{21}, x_1 x_{22})$	1	1
B_4	$W_v \to C_4[W_y, W_z]$	$C_4[x_1, (x_{21}, x_{22})] = (x_{21}, x_{22} x_1)$	1	1
U_{1L}	$W_v \to UP_{1L}[W_y]$	$UP_{1L}[(x_1, x_2)] = (a_i x_1, x_2)$	$e_v(a_i)$	$t_v(y)$
U_{1R}	$W_v \to UP_{1R}[W_y]$	$UP_{1R}[(x_1, x_2)] = (x_1 a_j, x_2)$	$e_v(a_j)$	$t_v(y)$
U_{2L}	$W_v \to UP_{2L}[W_y]$	$UP_{2L}[(x_1, x_2)] = (x_1, a_k x_2)$	$e_v(a_k)$	$t_v(y)$
U_{2R}	$W_v \to UP_{2R}[W_y]$	$UP_{2R}[(x_1, x_2)] = (x_1, x_2 a_l)$	$e_v(a_l)$	$t_v(y)$
P	$W_v \to BP[W_y]$	$BP[(x_1, x_2)] = (a_i x_1, x_2 a_l)$	$e_v(a_i, a_l)$	$t_v(y)$

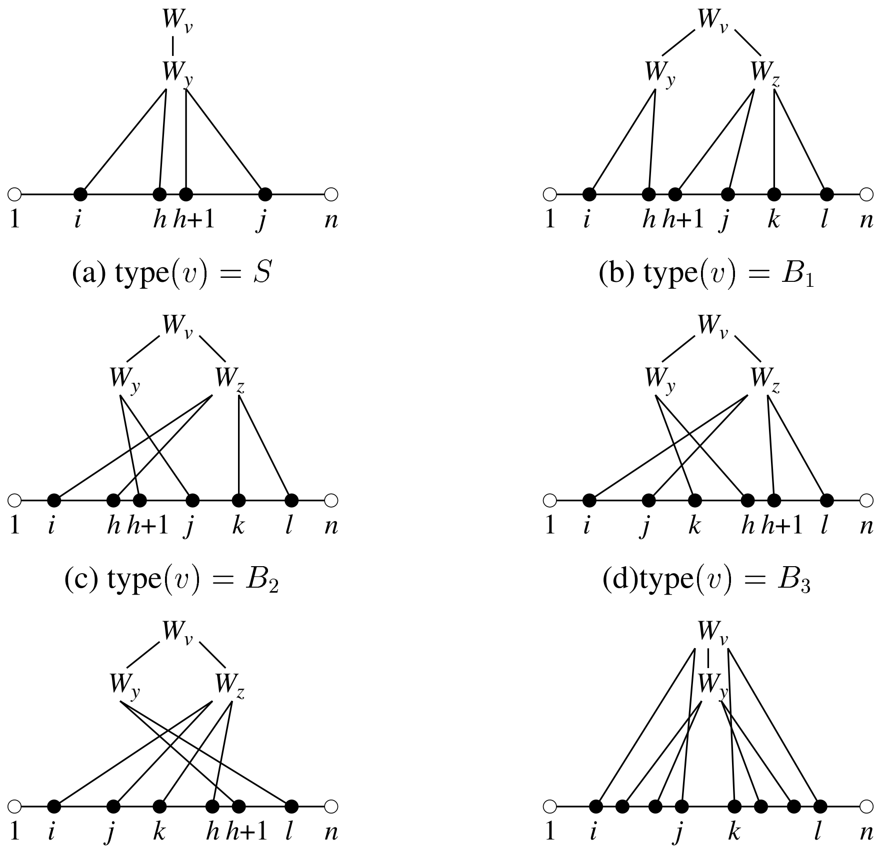
4.1 Alignment Problem

The alignment problem for G_s is to find the most probable derivation tree for a given input sequence. This problem can be solved by a dynamic programming algorithm similar to the CYK algorithm for SCFGs [4], and in this paper, we also call the parsing algorithm for G_s the CYK algorithm. We fix an input sequence $w = a_1 \cdots a_n$ ($|w| = n$). In fact, w is an RNA sequence composed of four symbols a, c, g and u. Let $\gamma_v(i, j)$ and $\gamma_y(i, j, k, l)$ be the logarithm of maximum probabilities of a derivation subtree rooted at a nonterminal W_v for a terminal subsequence $a_i \cdots a_j$ and of a derivation subtree rooted at a nonterminal W_y for a tuple of terminal subsequences $(a_i \cdots a_j, a_k \cdots a_l)$ respectively. The variables $\gamma_v(i, i-1)$ and $\gamma_y(i, i-1, j, j-1)$ are the logarithm of maximum probabilities for an empty sequence ε and a pair of ε. Let $\tau_v(i, j)$ and $\tau_y(i, j, k, l)$ be traceback variables for constructing a derivation tree, which are calculated together with $\gamma_v(i, j)$ and $\gamma_y(i, j, k, l)$. The CYK algorithm uses five dimensional dynamic programming matrix to calculate γ, which leads to $\log P(w, \hat{\pi} \mid \theta)$ where $\hat{\pi}$ is the most probable derivation tree. For notational convenience, we will use $e_v(a_i, a_j, a_k, a_l)$ indicating that for type$(v) = U_{1L}$, $e_v(a_i, a_j, a_k, a_l) = e_v(a_i)$, for type$(v) = U_{1R}$, $e_v(a_i, a_j, a_k, a_l) = e_v(a_j)$, for type$(v) = U_{2L}$, $e_v(a_i, a_j, a_k, a_l) = e_v(a_k)$, for type$(v) = U_{2R}$, $e_v(a_i, a_j, a_k, a_l) = e_v(a_l)$, for type$(v) = P$, $e_v(a_i, a_j, a_k, a_l) = e_v(a_i, a_l)$ and for the other types, $e_v(a_i, a_j, a_k, a_l) = 1$. The illustration of the iteration step in the CYK algorithm is shown in Figure 4. The detailed description of the algorithm is as follows:

Table 2: The number of symbols emitted by nonterminals

Type	Δ_v^{1L}	Δ_v^{1R}	Δ_v^{2L}	Δ_v^{2R}
E	0	0	0	0
S	0	0	0	0
D	0	0	0	0
B_1	0	0	0	0
B_2	0	0	0	0
B_3	0	0	0	0
B_4	0	0	0	0
U_{1L}	1	0	0	0
U_{1R}	0	1	0	0
U_{2L}	0	0	1	0
U_{2R}	0	0	0	1
P	1	0	0	1

Algorithm 1 (CYK).
Initialization:
for $i \leftarrow 1$ **to** $n+1$, $j \leftarrow i$ **to** $n+1$, $v \leftarrow 1$ **to** m
  do if type$(v) = E$
    then $\gamma_v(i, i-1, j, j-1) \leftarrow 0$
  else $\gamma_v(i, i-1, j, j-1) \leftarrow -\infty$
Iteration:
for $i \leftarrow n$ **downto** 1, $j \leftarrow i-1$ **to** n, $k \leftarrow n+1$ **downto** $j+1$, $l \leftarrow k-1$ **to** n, $v \leftarrow 1$ **to** m
  do if type$(v) = E$
    then if $j = i-1$ **and** $l = k-1$
      then skip
    else $\gamma_v(i, j, k, l) \leftarrow -\infty$
  if type$(v) = S$
    then $\gamma_v(i, j) \leftarrow \max_{y \in \mathcal{C}_v} \max_{h=i-1,\ldots,j} [\log t_v(y) + \gamma_y(i, h, h+1, j)]$
      $\tau_v(i, j) \leftarrow \arg\max_{(y,h)} [\log t_v(y) + \gamma_y(i, h, h+1, j)]$
  if type$(v) = B_1$
    then $\gamma_v(i, j, k, l) \leftarrow \max_{h=i-1,\ldots,j} [\gamma_y(i, h) + \gamma_z(h+1, j, k, l)]$
      $\tau_v(i, j, k, l) \leftarrow \arg\max_{(y,z,h)} [\gamma_y(i, h) + \gamma_z(h+1, j, k, l)]$
  if type$(v) = B_2$



(a) type$(v) = S$ (b) type$(v) = B_1$

(c) type$(v) = B_2$ (d)type$(v) = B_3$

(e) type$(v) = B_4$ (f) otherwise

Figure 4: Illustration of the iteration step for calculating γ

then $\gamma_v(i,j,k,l) \leftarrow \max_{h=i-1,\dots,j}[\gamma_y(h+1,j) + \gamma_z(i,h,k,l)]$

$\tau_v(i,j,k,l) \leftarrow \arg\max_{(y,z,h)}[\gamma_y(h+1,j) + \gamma_z(i,h,k,l)]$

if type$(v) = B_3$

then $\gamma_v(i,j,k,l) \leftarrow \max_{h=k-1,\dots,l}[\gamma_z(i,j,h+1,l) + \gamma_y(k,h)]$

$\tau_v(i,j,k,l) \leftarrow \arg\max_{(y,z,h)}[\gamma_z(i,j,h+1,l) + \gamma_y(k,h)]$

if type$(v) = B_4$

then $\gamma_v(i,j,k,l) \leftarrow \max_{h=k-1,\dots,l}[\gamma_z(i,j,k,h) + \gamma_y(h+1,l)]$

$\tau_v(i,j,k,l) \leftarrow \arg\max_{(y,z,h)}[\gamma_z(i,j,k,h) + \gamma_y(h+1,l)]$

if type$(v) = P$

then if $j = i-1$ **or** $l = k-1$

then $\gamma_v(i,j,k,l) \leftarrow -\infty$

else $\gamma_v(i,j,k,l) \leftarrow \max_{y \in \mathcal{C}_v}[\log e_v(a_i,a_l) + \log t_v(y) + \gamma_y(i+1,j,k,l-1)]$

$\tau_v(i,j,k,l) \leftarrow \arg\max_{y}[\log e_v(a_i,a_l) + \log t_v(y) + \gamma_y(i+1,j,k,l-1)]$

$$
\begin{aligned}
\textbf{else } \gamma_v(i,j,k,l) &\leftarrow \max_{y \in \mathcal{C}_v}[\log e_v(a_i,a_j,a_k,a_l) + \log t_v(y) \\
&\qquad\qquad + \gamma_y(i+\Delta_v^{1L}, j-\Delta_v^{1R}, k+\Delta_v^{2L}, l-\Delta_v^{2R})] \\
\tau_v(i,j,k,l) &\leftarrow \arg\max_{y}[\log e_v(a_i,a_j,a_k,a_l) + \log t_v(y) \\
&\qquad\qquad + \gamma_y(i+\Delta_v^{1L}, j-\Delta_v^{1R}, k+\Delta_v^{2L}, l-\Delta_v^{2R})] \qquad \square
\end{aligned}
$$

When the calculation terminates, we obtain $\log P(w, \hat{\pi} \mid \theta) = \gamma_1(1, n)$. If there are b bifurcation nonterminals and a other nonterminals, the time complexity of the CYK algorithm is $O(amn^4 + bn^5)$. It is apparent that the space complexity is $O(mn^4)$.

To recover the optimal derivation tree, we use the traceback variables τ and the pushdown stack holding tuples of integers of the forms (v, i, j) and (y, i, j, k, l). The full description of the traceback algorithm is as follows:

Algorithm 2 (CYK traceback)**.**
Initialization:
$(v, h) \leftarrow \tau_1(1, n)$
attach W_v as the child of W_1
push $(v, 1, h, h+1, n)$ on the stack
Iteration:
while the stack is not empty
    do pop
    if type$(v) = E$
        then attach $(\varepsilon, \varepsilon)$ as the child of W_v
    if type$(v) = S$
        then $(y, h) \leftarrow \tau_v(i, j)$
            attach W_y as the child of W_v
            push $(y, i, h, h+1, j)$
    if type$(v) = B_1$
        then $(y, z, h) \leftarrow \tau_v(i, j, k, l)$
            attach W_y, W_z as the children of W_v
            push $(z, h+1, j, k, l)$
            push (y, i, h)
    if type$(v) = B_2$
        then $(y, z, h) \leftarrow \tau_v(i, j, k, l)$
            attach W_y, W_z as the children of W_v
            push (z, i, h, k, l)
            push $(y, h+1, j)$
    if type$(v) = B_3$
        then $(y, z, h) \leftarrow \tau_v(i, j, k, l)$

attach W_y, W_z as the children of W_v
push $(z, i, j, h + 1, l)$
push (y, k, h)
if type$(v) = B_4$
then $(y, z, h) \leftarrow \tau_v(i, j, k, l)$
attach W_y, W_z as the children of W_v
push (z, i, j, k, h)
push $(y, h + 1, l)$
else $y \leftarrow \tau_v(i, j, k, l)$
attach W_y as the child of W_v
push $(y, i + \Delta_v^{1L}, j - \Delta_v^{1R}, k + \Delta_v^{2L}, l - \Delta_v^{2R})$ $\square$

4.2 Scoring Problem

As in SCFGs [10], the scoring problem for G_s can be solved by the inside algorithm. The inside algorithm calculates the summed probabilities $\alpha_v(i, j)$ and $\alpha_y(i, j, k, l)$ of all derivation subtrees rooted at a nonterminal W_v for a subsequence $a_i \cdots a_j$ and of all derivation subtrees rooted at a nonterminal W_y for a tuple of subsequences $(a_i \cdots a_j, a_k \cdots a_l)$ respectively. The variables $\alpha_v(i, i-1)$ and $\alpha_y(i, i-1, j, j-1)$ are defined for empty sequences in a similar way to the CYK algorithm. Therefore, we can easily obtain the inside algorithm by replacing max operations with summations in the CYK algorithm. When the calculation terminates, we obtain the probability $P(w \mid \theta) = \alpha_1(1, n)$. The time and space complexities of the algorithm are identical with those of the CYK algorithm.

Algorithm 3 (Inside)**.**
Initialization:
for $i \leftarrow 1$ **to** $n + 1$, $j \leftarrow i$ **to** $n + 1$, $v \leftarrow 1$ **to** m
do if type$(v) = E$
then $\alpha_v(i, i-1, j, j-1) \leftarrow 1$
else $\alpha_v(i, i-1, j, j-1) \leftarrow 0$
Iteration:
for $i \leftarrow n$ **downto** 1, $j \leftarrow i - 1$ **to** n, $k \leftarrow n + 1$ **downto** $j + 1$, $l \leftarrow k - 1$ **to** n, $v \leftarrow 1$ **to** m
do if type$(v) = E$
then if $j = i - 1$ **and** $l = k - 1$
then skip
else $\alpha_v(i, j, k, l) \leftarrow 0$
if type$(v) = S$

then $\alpha_v(i,j) \leftarrow \sum_{y \in \mathcal{C}_v} \sum_{h=i-1}^{j} t_v(y)\alpha_y(i,h,h+1,j)$

if type$(v) = B_1$

then $\alpha_v(i,j,k,l) \leftarrow \sum_{h=i-1}^{j} \alpha_y(i,h)\alpha_z(h+1,j,k,l)$

if type$(v) = B_2$

then $\alpha_v(i,j,k,l) \leftarrow \sum_{h=i-1}^{j} \alpha_y(h+1,j)\alpha_z(i,h,k,l)$

if type$(v) = B_3$

then $\alpha_v(i,j,k,l) \leftarrow \sum_{h=k-1}^{l} \alpha_z(i,j,h+1,l)\alpha_y(k,h)$

if type$(v) = B_4$

then $\alpha_v(i,j,k,l) \leftarrow \sum_{h=k-1}^{l} \alpha_z(i,j,k,h)\alpha_y(h+1,l)$

if type$(v) = P$

then if $j = i-1$ **or** $l = k-1$

then $\alpha_v(i,j,k,l) \leftarrow 0$

else $\alpha_v(i,j,k,l) \leftarrow \sum_{y \in \mathcal{C}_v} e_v(a_i,a_l)t_v(y)\alpha_y(i+1,j,k,l-1)]$

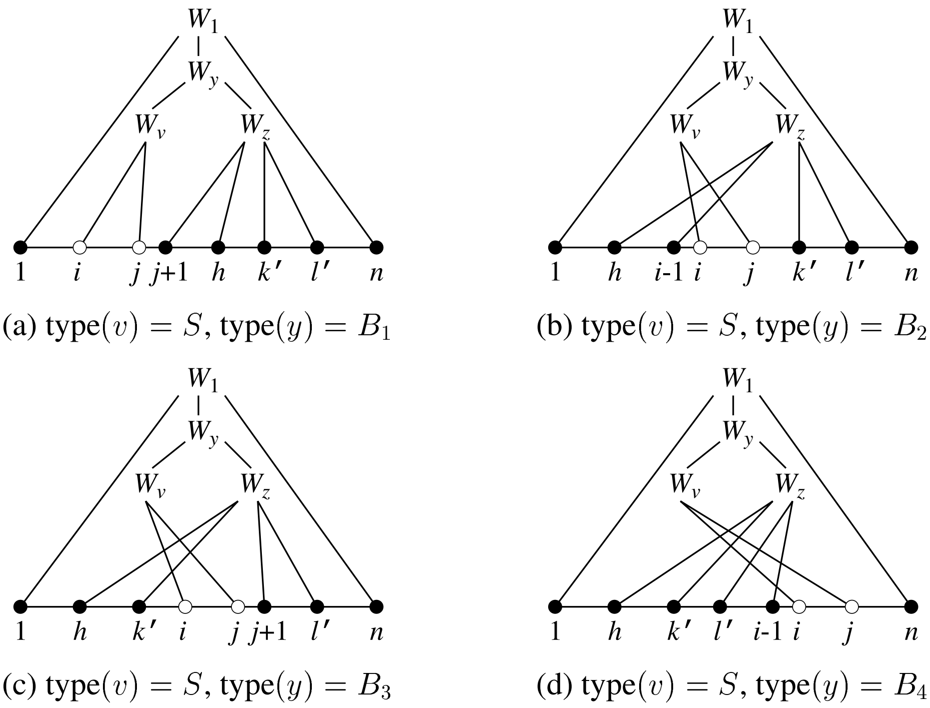
else $\alpha_v(i,j,k,l) \leftarrow \sum_{y \in \mathcal{C}_v} e_v(a_i,a_j,a_k,a_l)t_v(y)$

$\alpha_y(i+\Delta_v^{1L}, j-\Delta_v^{1R}, k+\Delta_v^{2L}, l-\Delta_v^{2R})$ □

In order to re-estimate the probability parameters of G_s, we need the outside algorithm. The outside algorithm calculates the summed probability $\beta_v(i,j)$ of all derivation trees excluding subtrees rooted at a nonterminal W_v generating a subsequence $a_i \cdots a_j$. Also, it calculates $\beta_y(i,j,k,l)$, the summed probability of all derivation trees excluding subtrees rooted at a nonterminal W_y generating a tuple of subsequences $(a_i \cdots a_j, a_k \cdots a_l)$. What has to be noticed is calculating the outside variables β requires the inside variables α. Unlike CYK and inside algorithms, the outside algorithm recursively works its way inward. Figure 5 shows the iteration step in the algorithm. Formally, the outside algorithm is as follows:

Algorithm 4 (Outside)**.**
Initialization:
$\beta_1(1,n) \leftarrow 1$
Iteration:
for $i \leftarrow 1$ **to** $n+1, j \leftarrow n$ **downto** $i-1, k \leftarrow j+1$ **to** $n+1, l \leftarrow n$ **downto** $k-1, v \leftarrow 1$ **to** m

do if type$(v) = S$, $\mathcal{P}_v = \{y\}$, type$(y) = B_1$, $\mathcal{C}_y = \{v, z\}$



(a) type$(v) = S$, type$(y) = B_1$ (b) type$(v) = S$, type$(y) = B_2$

(c) type$(v) = S$, type$(y) = B_3$ (d) type$(v) = S$, type$(y) = B_4$

Figure 5: Illustration of the iteration step for calculating β

then $\beta_v(i,j) \leftarrow \sum_{h=j}^{n} \sum_{k'=h+1}^{n+1} \sum_{l'=k'-1}^{n} \beta_y(i,h,k',l')\alpha_z(j+1,h,k',l')$

if type$(v) = S$, $\mathcal{P}_v = \{y\}$, type$(y) = B_2$, $\mathcal{C}_y = \{v,z\}$

then $\beta_v(i,j) \leftarrow \sum_{h=1}^{i} \sum_{k'=j+1}^{n+1} \sum_{l'=k'-1}^{n} \beta_y(h,j,k',l')\alpha_z(h,i-1,k',l')$

if type$(v) = S$, $\mathcal{P}_v = \{y\}$, type$(y) = B_3$, $\mathcal{C}_y = \{v,z\}$

then $\beta_v(i,j) \leftarrow \sum_{h=1}^{i} \sum_{k'=h-1}^{i-1} \sum_{l'=j}^{n} \beta_y(h,k',i,l')\alpha_z(h,k',j+1,l')$

if type$(v) = S$, $\mathcal{P}_v = \{y\}$, type$(y) = B_4$, $\mathcal{C}_y = \{v,z\}$

then $\beta_v(i,j) \leftarrow \sum_{h=1}^{i} \sum_{k'=h-1}^{i-1} \sum_{l'=k'+1}^{i} \beta_y(h,k',l',j)\alpha_z(h,k',l',i-1)$

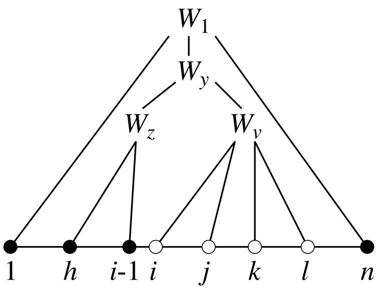
if type$(v) \neq S$, $\mathcal{P}_v = \{y\}$, type$(y) = B_1$, $\mathcal{C}_y = \{z,v\}$

then $\beta_v(i,j,k,l) \leftarrow \sum_{h=1}^{i} \beta_y(h,j,k,l)\alpha_z(h,i-1)$

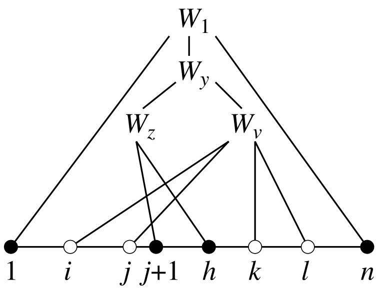
if type$(v) \neq S$, $\mathcal{P}_v = \{y\}$, type$(y) = B_2$, $\mathcal{C}_y = \{z,v\}$

then $\beta_v(i,j,k,l) \leftarrow \sum_{h=j}^{k-1} \beta_y(i,h,k,l)\alpha_z(j+1,h)$

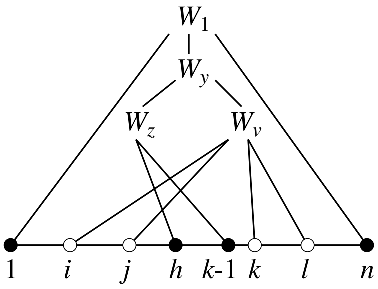
if type$(v) \neq S$, $\mathcal{P}_v = \{y\}$, type$(y) = B_3$, $\mathcal{C}_y = \{z,v\}$



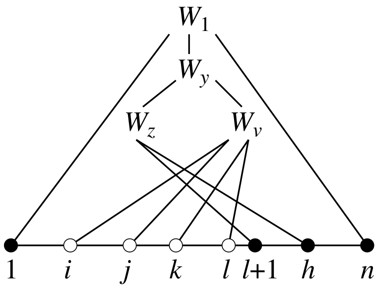
(e) type$(v) \neq S$, type$(y) = B_1$



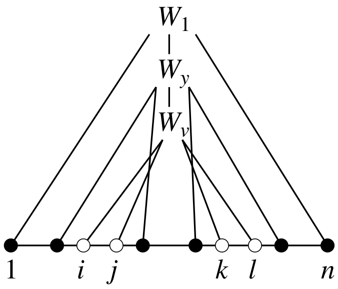
(f) type$(v) \neq S$, type$(y) = B_2$



(g) type$(v) \neq S$, type$(y) = B_3$



(h) type$(v) \neq S$, type$(y) = B_4$



(i) otherwise

Figure 5: Illustration of the iteration step for calculating β (continued)

$$\textbf{then } \beta_v(i,j,k,l) \leftarrow \sum_{h=j+1}^{k} \beta_y(i,j,h,l)\alpha_z(h,k-1)$$

if type$(v) \neq S$, $\mathcal{P}_v = \{y\}$, type$(y) = B_4$, $\mathcal{C}_y = \{z,v\}$

$$\textbf{then } \beta_v(i,j,k,l) \leftarrow \sum_{h=l}^{n} \beta_y(i,j,k,h)\alpha_z(l+1,h)$$

$$\textbf{else } \beta_v(i,j,k,l) \leftarrow \sum_{y\in\mathcal{P}_v} \beta_y(i-\Delta_y^{1L}, j+\Delta_y^{1R}, k-\Delta_y^{2L}, l+\Delta_y^{2R})$$
$$e_y(a_{i-\Delta_y^{1L}}, a_{j+\Delta_y^{1R}}, a_{k-\Delta_y^{2L}}, a_{l+\Delta_y^{2R}})t_y(v) \qquad \square$$

The time and space complexities of the outside algorithm are the same as those of CYK and inside algorithms.

4.3 Training Problem

The training problem for G_s can be solved by the EM (expectation maximization) algorithm called the inside-outside algorithm where the inside variables α and outside variables β are used to re-estimate probability parameters. First, we consider the probability that a nonterminal W_v is used at positions i, j, k and l in a derivation of a single sequence w. If type$(v) = S$, the probability is $\frac{1}{P(w|\theta)}\alpha_v(i,j)\beta_v(i,j)$, otherwise $\frac{1}{P(w|\theta)}\alpha_v(i,j,k,l)\beta_v(i,j,k,l)$. By summing these over all positions in the sequence, we can obtain the expected number of times that W_v is used for w as follows: for type$(v) = S$, the expected count is

$$\frac{1}{P(w \mid \theta)} \sum_{i=1}^{n+1} \sum_{j=i-1}^{n} \alpha_v(i,j)\beta_v(i,j),$$

otherwise

$$\frac{1}{P(w \mid \theta)} \sum_{i=1}^{n+1} \sum_{j=i-1}^{n} \sum_{k=j+1}^{n+1} \sum_{l=k-1}^{n} \alpha_v(i,j,k,l)\beta_v(i,j,k,l).$$

Next, we extend these expected values from a single sequence w to multiple independent sequences $w^{(r)}$ $(1 \le r \le N)$. Let $\alpha^{(r)}$ and $\beta^{(r)}$ be the inside and outside variables calculated for each input sequence $w^{(r)}$. Then we can obtain the expected number of times that a nonterminal W_v is used for training sequences $w^{(r)}$ $(1 \le r \le N)$ by summing the above terms over all sequences:

$$E(v) = \begin{cases} \displaystyle\sum_{r=1}^{N}\sum_{i=1}^{n+1}\sum_{j=i-1}^{n} \frac{1}{P(w^{(r)} \mid \theta)} \alpha_v^{(r)}(i,j)\beta_v^{(r)}(i,j), & (\text{type}(v) = S) \\ \displaystyle\sum_{r=1}^{N}\sum_{i=1}^{n+1}\sum_{j=i-1}^{n}\sum_{k=j+1}^{n+1}\sum_{l=k-1}^{n} \frac{1}{P(w^{(r)} \mid \theta)} \alpha_v^{(r)}(i,j,k,l)\beta_v^{(r)}(i,j,k,l). & (\text{type}(v) \ne S) \end{cases}$$

Similarly, the expected number of times that W_v makes a transition to W_y for training sequences is

$$E(v \to y) = \begin{cases} \displaystyle\sum_{r=1}^{N}\sum_{i=1}^{n+1}\sum_{j=i-1}^{n}\sum_{h=i-1}^{j} \frac{1}{P(w^{(r)} \mid \theta)} \beta_v^{(r)}(i,j)t_v(y) & (\text{type}(v) = S) \\ \quad \alpha_y^{(r)}(i,h,h+1,j), & \\ \displaystyle\sum_{r=1}^{N}\sum_{i=1}^{n+1}\sum_{j=i-1}^{n}\sum_{k=j+1}^{n+1}\sum_{l=k-1}^{n} \frac{1}{P(w^{(r)} \mid \theta)} \beta_v^{(r)}(i,j,k,l) & (\text{type}(v) \ne S) \\ \quad e_v(a_i,a_j,a_k,a_l)t_v(y) & \\ \quad \alpha_y^{(r)}(i+\Delta_v^{1L}, j-\Delta_v^{1R}, k+\Delta_v^{2L}, l-\Delta_v^{2R}). & \end{cases}$$

The expected number of times that W_v emits symbols for training sequences is

$$E(v \to a) = \begin{cases} \displaystyle\sum_{r=1}^{N}\sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=j+1}^{n+1}\sum_{l=k-1}^{n}\frac{1}{P(w^{(r)} \mid \theta)}\delta(a_i^{(r)} = a) \quad (\text{type}(v) = U_{1L}) \\ \qquad \beta_v^{(r)}(i,j,k,l)\alpha_v^{(r)}(i,j,k,l), \\ \displaystyle\sum_{r=1}^{N}\sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=j+1}^{n+1}\sum_{l=k-1}^{n}\frac{1}{P(w^{(r)} \mid \theta)}\delta(a_j^{(r)} = a) \quad (\text{type}(v) = U_{1R}) \\ \qquad \beta_v^{(r)}(i,j,k,l)\alpha_v^{(r)}(i,j,k,l), \\ \displaystyle\sum_{r=1}^{N}\sum_{i=1}^{n-1}\sum_{j=i-1}^{n-1}\sum_{k=j+1}^{n}\sum_{l=k}^{n}\frac{1}{P(w^{(r)} \mid \theta)}\delta(a_k^{(r)} = a) \quad (\text{type}(v) = U_{2L}) \\ \qquad \beta_v^{(r)}(i,j,k,l)\alpha_v^{(r)}(i,j,k,l), \\ \displaystyle\sum_{r=1}^{N}\sum_{i=1}^{n-1}\sum_{j=i-1}^{n-1}\sum_{k=j+1}^{n}\sum_{l=k}^{n}\frac{1}{P(w^{(r)} \mid \theta)}\delta(a_l^{(r)} = a) \quad (\text{type}(v) = U_{2R}) \\ \qquad \beta_v^{(r)}(i,j,k,l)\alpha_v^{(r)}(i,j,k,l), \end{cases}$$

and for type$(v) = P$,

$$E(v \to ab) = \sum_{r=1}^{N}\sum_{i=1}^{n-1}\sum_{j=i}^{n-1}\sum_{k=j+1}^{n}\sum_{l=k}^{n}\frac{1}{P(w^{(r)} \mid \theta)}\delta(a_i^{(r)} = a, a_l^{(r)} = b)\beta_v^{(r)}(i,j,k,l)$$
$$\alpha_v^{(r)}(i,j,k,l)$$

where $\delta()$ is 1 if the expression in the parenthesis is ture, and 0 if the expression is false.

Now, we re-estimate probability parameters by using the above expected counts. Let $\hat{t}_v(y)$ be re-estimated transition probabilities from W_v to W_y. Also, let $\hat{e}_v(a)$ and $\hat{e}_v(a,b)$ be re-estimated emission probabilities that W_v emits a symbol a and two symbols a and b respectively. We can obtain each re-estimated probability by the following equations:

$$\hat{t}_v(y) = \frac{E(v \to y)}{E(v)}, \quad \hat{e}_v(a) = \frac{E(v \to a)}{E(v)}, \quad \hat{e}_v(a,b) = \frac{E(v \to ab)}{E(v)}.$$

Note that the expected count correctly corresponding to its nonterminal type must be substituted for the above equations. For example, if type$(v) = S$,

$$\hat{t}_v(y) = \frac{\sum_{r=1}^{N}\sum_{i=1}^{n+1}\sum_{j=i-1}^{n}\sum_{h=i-1}^{j}\frac{1}{P(w^{(r)}|\theta)}\beta_v^{(r)}(i,j)t_v(y)\alpha_y^{(r)}(i,h,h+1,j)}{\sum_{r=1}^{N}\sum_{i=1}^{n+1}\sum_{j=i-1}^{n}\frac{1}{P(w^{(r)}|\theta)}\alpha_v^{(r)}(i,j)\beta_v^{(r)}(i,j)}.$$

The detailed description of the other re-estimated probability parameters can be easily obtained in a similar way.

5 Experimental Results

5.1 Data for Experiments

The dataset for experiments was taken from "Rfam" (version 7.0) [6] which is a database of multiple sequence alignment and covariance models [5] representing non-coding RNA families. There are 13 RNA families with pseudoknot annotation and we selected two viral RNA families named Corona_pk_3 and Tombus_3_IV (see Table 3). Corona_pk_3 has a simple pseudoknotted structure, whereas Tombus_3_IV has a multi-branched structure with a pseudoknot.

5.2 Implementation

We specified SMCFG G_s by utilizing secondary structure annotations of Corona_pk_3 and Tombus_3_IV in Rfam respectively. Rules were determined by considering consensus secondary structure. Probability parameters were estimated by the simplest pseudocounting method known as the Laplace's rule [4]: to add one extra count to the true counts for each base configuration observed in a few selected sequences. The other sequences in the alignment were used as the test sequences for prediction (see Table 3). We implemented the CYK algorithm with traceback in ANSI C on a machine with Intel Pentium D CPU 2.80 GHz and 2.00 GB RAM. Straightforward implementation gives rise to a serious problem of lack of memory space due to the higher order dynamic programming matrix (remember that the space complexity of the CYK algorithm is $O(mn^4)$). The dynamic programming matrix in our specified model is sparse, and therefore, we successfully implemented the matrix as a hash table storing only nonzero probability values (equivalently, finite values of the logarithm of probabilities).

5.3 Tests

We tested the accuracy of structure prediction using the CYK algorithm. We calculated the sensitivity which is the ratio of the number of correct base pairs predicted by the algorithm (true positives) to the total number of base pairs specified by the trusted annotation. Specificity was not calculated since the number of base pairs specified by the trusted annotation was the same as that of base pairs predicted by the algorithm. The results are shown in Table 4. A nearly correct prediction (94.4% sensitivity) for Corona_pk_3 is shown in Figure 6 where underlined base pairs agree with trusted ones. The secondary structures predicted by our algorithm agree very well with the trusted structures. The running time of prediction in Corona_pk_3 is much shorter than that of prediction in Tombus_3_IV since every sequence in Corona_pk_3 can be generated by rules without bifurcation nonterminals. In this case, the time complexity of the CYK algorithm is $O(m^2 n^4)$.

Table 3: Two RNA families from Rfam ver. 7.0

Family	Range of length	# of annotated sequences	# of test sequences
Corona_pk_3	62–64	14	10
Tombus_3_IV	89–92	18	12

Table 4: Prediction results

Family	Sensitivity [%]			CPU time [sec]		
	Average	Min	Max	Average	Min	Max
Corona_pk_3	99.4	94.4	100.0	27.8	26.0	30.4
Tombus_3_IV	100.0	100.0	100.0	244.8	215.2	257.5

Corona_pk3 (EMBL accession #: X51325.1)

[Trusted structure in Rfam]

```
CUAGUCUUAUACACAAUGGUAAGCCAGUGGUAGUAAAGGUAUAAGAAAUUUGCUACUAUGUUA
    [[[[[[[[              ((( ((((((( ]]]]]]]]  ))))))) )))
```

[Prediction by SMCFG]

```
CUAGUCUUAUACACAAUGGUAAGCCAGUGGUAGUAAAGGUAUAAGAAAUUUGCUACUAUGUUA
    [[[[[[[[              ((((((((((  ]]]]]]]]   ))))))))))
```

Figure 6: Comparison of a prediction result with a trusted structure in Rfam

5.4 Comparison with PSTAG

We compared the prediction accuracy of our SMCFG algorithm with that of PSTAG algorithm [11]. PSTAGs, as we have mentioned before, are proposed for modeling pairwise alignment of RNA sequences with pseudoknots and assign a probability to each alignment of TAG derivation trees. PSTAG algorithm, based on dynamic programming, calculates the most likely alignment for the pair of TAG derivation trees where one of them is in the form of an unfolded sequence and the other is a TAG derivation tree for known structure. The result of comparison in Corona_pk_3 and Tombus_3_IV is shown in Table 5. SMCFG method shows better performance in accuracy than PSTAG method in these two RNA families.

Table 5: Comparison of SMCFG with PSTAG

Model	Average sensitivity [%]	
	Corona_pk_3	Tombus_3_IV
SMCFG	99.4	100.0
PSTAG	94.6	97.4

6 Discussion

In the computational experiments using SMCFG, we obtained good prediction results in accuracy and we did not trained probability parameters using the inside-outside algorithm any more. Part of the reason for success of prediction without training is that we were able to obtain good structural alignment from the database. The word "good" means that every trusted structure is little different from consensus structure and the number of gaps in alignment is relatively few. In fact, the first experimental results, omitted in the previous section, showed 76.6 % average sensitivities in Corona_pk_3 and 95.7 % in Tombus_3_IV. We should notice that there are more gaps in the alignment of Corona_pk_3 than that of Tombus_3_IV. Changing rules such that delete rules are not successively used after the terminating rule $W_v \to (\varepsilon, \varepsilon)$, we can obtain the present results shown in Table 4. Hence, prediction accuracy will depend on the way to construct rules. We think that the most sensitive factor for prediction accuracy will be the number of consecutive gaps in alignment.

PSTAG method aligns an unfolded sequence with a derivation tree representing trusted structure. In SMCFG, rules are constructed according to a consensus structure and then the most probable derivation tree is calculated. In this sense, SMCFG and PSTAG have a common property that both of them take structural alignment into consideration implicitly or explicitly. Time and space complexities of SMCFG algorithm have the same order as those of PSTAG algorithm, whereas SMCFG algorithm consumes less memory than PSTAG algorithm since the dynamic programming matrix of SMCFG algorithm is sparse. This greatly contributes to practicability in computational structure prediction.

7 Conclusion

In this paper, we have proposed a probabilistic model named SMCFG, and designed a polynomial time parsing algorithm and a parameter estimation method for SMCFG. Moreover, we have demonstrated computational experiments of RNA secondary structure prediction with pseudoknots using SMCFG parsing algorithm, which shows very good performance in accuracy.

Comparing with other prediction methods such as a thermodynamical approach, stochastic grammars have an advantage in easily modeling RNA secondary structure we would like to analyze and training probability parameters. We should notice that there are a trade-off between prediction accuracy and cost for constructing an initial grammar.

The major difference between MCFGs and other grammars fully describing RNA pseudoknots is that the generative power of MCFGs can be set arbitrarily by choosing grammar parameters; dimension, rank and degree (see section 2). Thus, SMCFG method can be applied to more complicated biological structure analysis such as protein secondary structure prediction, which is left as future work.

Acknowledgments

This work is supported in part by Grant-in-Aid for Scientific Research from Japan Society for the Promotion of Science (JSPS). We also wish to thank JSPS Research Fellowships for Young Scientists for their generous financial assistance. The authors thank Professor Tadao Kasami for his invaluable discussions on grammar-based analysis and Dr. Yoshiaki Takata for his useful comments on implementation of high dimensional dynamic programming.

References

[1] T. Akutsu, "Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots," Discrete Applied Mathematics, vol. 104, pp. 45–62, 2000.

[2] M. Brown and C. Wilson, "RNA pseudoknot modeling using intersections of stochastic context free grammars with applications to database search," Pacific Symposium on Biocomputing, pp. 109–125, 1996.

[3] L. Cai, R. L. Malmberg and Y. Wu, "Stochastic modeling of RNA pseudoknotted structures: a grammatical approach," Bioinformatics, vol. 19, suppl. 1, pp. i66–i73, 2003.

[4] R. Durbin, S. R. Eddy, A. Krogh and G. Mitchison, Biological Sequence Analysis, Cambridge University Press, 1998.

[5] S. R. Eddy and R. Durbin, "RNA sequence analysis using covariance models," Nucleic Acids Research, vol. 22, no. 11, pp. 2079–2088, 1994.

[6] S. Griffiths-Jones, A. Bateman, M. Marshall, A. Khanna and S. R. Eddy, "Rfam: an RNA family database," Nucleic Acids Research, vol. 31, no. 1, pp. 439–441, 2003.

[7] T. Kasami, H. Seki and M. Fujii, "Generalized context-free grammar and multiple context-free grammar," IEICE Transactions on Information & Systems, vol. J71-D, no. 5, pp. 758–765, 1988 (in Japanese).

[8] T. Kasami, H. Seki and M. Fujii, "On the membership problem for head languages and multiple context-free languages," IEICE Transactions on Information & Systems, vol. J71-D, no. 6, pp. 935–941, 1988 (in Japanese).

[9] Y. Kato, H. Seki and T. Kasami, "On the generative power of grammars for RNA secondary structure," IEICE Transactions on Information & Systems, vol. E88-D, no. 1, pp. 53–64, 2005.

[10] K. Lari and S. J. Young, "The estimation of stochastic context-free grammars using the inside-outside algorithm," Computer Speech and Language, vol. 4, pp. 35–56, 1990.

[11] H. Matsui, K. Sato and Y. Sakakibara, "Pair stochastic tree adjoining grammars for aligning and predicting pseudoknot RNA structures," Bioinformatics, vol. 21, no. 11, pp. 2611–2617, 2005.

[12] E. Rivas and S. R. Eddy, "A dynamic programming algorithm for RNA structure prediction including pseudoknots," Journal of Molecular Biology, vol. 285, pp. 2053–2068, 1999.

[13] E. Rivas and S. R. Eddy, "The language of RNA: A formal grammar that includes pseudoknots," Bioinformatics, vol. 16, no. 4, pp. 334–340, 2000.

[14] Y. Sakakibara, M. Brown, R. Hughey, I. S. Mian, K. Sjölander, R. C. Underwood and D. Haussler, "Stochastic context-free grammars for tRNA modeling," Nucleic Acids Research, vol. 22, pp. 5112–5120, 1994.

[15] H. Seki, T. Matsumura, M. Fujii and T. Kasami, "On multiple context-free grammars," Theoretical Computer Science, vol. 88, pp. 191–229, 1991.

[16] Y. Uemura, A. Hasegawa, S. Kobayashi and T. Yokomori, "Tree adjoining grammars for RNA structure prediction," Theoretical Computer Science, vol. 210, pp. 277–303, 1999.