Computational Genomics **Fall Semester, 2009**

Lecture 10: December 22, 2009

Lecturer: Roded Sharan *Scribe: Adam Weinstock and Yaara Ben-Amram Segre*

10.1 Context Free Grammars

10.1.1 Introduction

In lecture 6, we saw Nussinov's algorithm[6], which solves the problem of RNA folding by maximizing base-pairing. However, that solution uses an arbitrary scoring function and provides no tools for probabilistic modeling of different RNA sequences. We now wish to create profiles to describe the folding patterns of RNA families and give a likelihood score to the pertinence of an RNA sequence to a specific family. As seen in lectures 7-8, HMMs can be used to create probabilistic profiles for sequence families. Such models are limited, however, as they are unable to describe relations between characters in distant locations along the sequence. In this lecture, we shall review stochastic context free grammars and show how they may be used to overcome the latter limitation.[1]

10.1.2 Reminder: Formal Languages

A formal language, or grammar, G is defined by a quartet $G = (V, \Sigma, S, R)$ Where:

- V is a non-terminal alphabet (e.g., $\{A, B, C, D, E, \ldots\}$).
- Σ is a terminal alphabet (e.g., $\{a, c, g, t\}$).
- $S \in V$ is a special start symbol.
- R is a set of rewriting rules called productions.

Formal languages are divided into four categories:

- **Regular grammars** — The simplest grammars, can be described using rules of the forms $W \to aW, W \to a$. They can be decided using deterministic finite automata (DFA). Note that an HMM is equivalent to a DFA with probabilities assigned to each transition.

[1]This lecture is based on Durbin et al., "Biological Sequence Analysis" (cambridge, '98) and slides by B. Majoros, Duke university.

- **Context free grammars (CFGs)** — constructed using rules of the form $W \to \beta$, where β is any combination of terminals and non-terminals. They are equivalent to push down automata.

- **Context sensitive grammars** — constructed using rules of the form $\alpha W \beta \to \alpha \gamma \beta$, where α, β, γ are combinations of terminals and non-terminals. They are equivalent to linear bounded automata.

- **Unrestricted grammars** — constructed using rules of the form $\alpha W \beta \to \gamma$, where α, β, γ are combinations of terminals and non-terminals. They are equivalent to Turing machines.

10.1.3 Context Free Grammars

Like all formal languages, a context-free grammar is defined by a quartet $G = (V, \Sigma, S, R)$. Productions in R are rules of the form: $X \to \lambda$, where $X \in V, \lambda \in (V \cup \Sigma)^*$.
The"context-freeness" is imposed by the requirement that the left hand side of each production rule may contain only a single symbol, and that symbol must be a non-terminal. Thus, a CFG cannot specify context-sensitive rules ($wXz \to w\delta z$) where the derivation of X is dependent on X's neighbors w, z.

10.1.4 Derivations

A derivation (or parse) consists of a series of applications of production rules from R, beginning with the start non-terminal S and ending with the terminal string x:
$S \Rightarrow s_1 \Rightarrow s_2 \Rightarrow s_3 \Rightarrow \ldots \Rightarrow x$
Where $s_i \in (V \cup \Sigma)^*, x \in \Sigma^*$. Each step, we replace non-terminal X with the right hand side of a rule from R, whose left hand side is X. If such a derivation exists, we say that G generates x, and denote it by $S \Rightarrow^* x$.

CFGs can be ambiguous. There may be many ways to derive x. Two derivations generating x may vary only in the order in which production rules are applied, which seems somewhat redundant. Consider the following example:

$G = (\{S, A, B\}, \{a, b\}, S, R)$

$R = \{S \to AB, A \to a, B \to b\}$

Two derivations can create the string 'ab':

1. $S \to AB \to aB \to ab$

2. $S \to AB \to Ab \to ab$

In order to avoid this redundancy, we will concentrate on leftmost derivations, where the leftmost non-terminal is always replaced first. All permutations of a certain set of productions will be represented by a single leftmost derivation. Note that we may still encounter multiple derivations of x.

10.1.5 Context-Free Versus Regular

The advantage of CFGs over HMMs lies in their ability to model arbitrary runs of matching pairs of elements, such as matching pairs of parentheses:

$$\ldots(((((((\ldots)))))))\ldots$$

When the number of matching pairs is unbounded, a finite-state model such as an HMM is inadequate to enforce the constraint that all left elements must have a matching right element. In contrast, in a CFG we can use rules such as $X \to (X)$.
A sample derivation using such a rule is:

$$X \Rightarrow (X) \Rightarrow ((X)) \Rightarrow (((X))) \Rightarrow ((((X)))) \Rightarrow (((((X)))))$$

An additional rule such as $X \to \epsilon$ is necessary to terminate the recursion.

10.1.6 Parse Trees

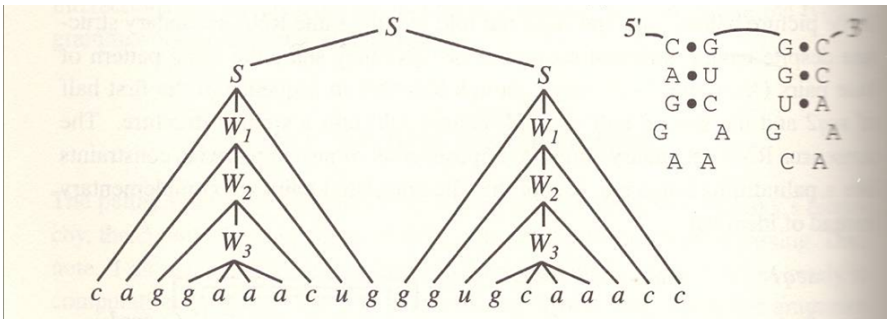

Figure 10.1: A sample parse tree deriving the sequence on the right[4].

A parse tree is a representation of a parse of a string by a CFG. The root of the tree is the start non-terminal S. The leaves – terminal symbols of the string. Internal nodes – non-terminals. The children of an internal node are the productions of that non-terminal (left-to-right order). Note that:

1. A subtree spans a contiguous sequence segment.
2. Each leftmost derivation corresponds with a single parse tree. This relation is one to one and onto.

10.1.7 Chomsky Normal Form

Any CFG which does not derive the empty string (i.e., $\epsilon \notin L(G)$) can be converted into an equivalent grammar in *Chomsky Normal Form (CNF)*. A CNF grammar is one in which all productions are of the form:

$$X \to YZ \text{ or } X \to a$$

for non-terminals X, Y, Z, and terminal a.
Transforming a CFG into CNF can be accomplished by appropriately-ordered application of the following operations:

- Eliminating useless symbols (non-terminals that only derive ϵ)
- Eliminating null productions ($X \to \epsilon$)
- Eliminating unit productions ($X \to Y$)
- Factoring long right hand side expressions
  ($A \to abc$ factored into $A \to aB, B \to bC, C \to c$)
- Factoring terminals ($A \to cB$ is factored into $A \to CB, C \to c$)

Example:

NON-CNF: $S \to aSt|tSa|cSg|gSc|L$
$L \to NNNN$
$N \to a|c|g|t$

CNF: $S \to AS_T|TS_A|CS_G|GS_C|NL_1$
$S_A \to SA$
$S_T \to ST$
$S_C \to SC$
$S_G \to SG$
$L_1 \to NL_2$
$L_2 \to NN$
$N \to a|c|g|t$
$A \to a$

$C \to c$
$G \to g$
$T \to t$

CNF allows us to easily implement parsers, at the expanse of adding many non-terminals and productions. Though CNF is a less intuitive way to view the problem domain, it may be beneficial in terms of computation.

10.2 Stochastic Context Free Grammers

A stochastic context-free grammar (SCFG) is a CFG plus a probability distribution on productions:

$$G = (V, \Sigma, S, R, Pp)$$

Where $Pp : R \to [0, 1]$. Pp provides a probability distribution for each non-terminal X. That is, the probabilities of all its productions sum to one:

$$\forall_{X \in V} \sum_{X \to \lambda} Pp(X \to \lambda) = 1$$

The probability of a derivation $S \Rightarrow^* x$ is the product of the probabilities of all its productions:

$$\prod_i P_P(X_i \to \lambda_i)$$

We can sum over all possible (leftmost) derivations of a given string x to get the probability that G will generate x at random:

$$P(x|G) = P(S \Rightarrow^* x|G) = \sum_j P(S \Rightarrow_j^* x|G)$$

Note: We do not state 'given G' ($|G$) on many occasions in order to avoid long equations.

Example:
Consider the grammar:

$G = (V_G, \Sigma, S, R_G, P_G)$
$V_G = \{S, L, N\}$,
$\Sigma = \{a, c, g, t\}$

R_G is the set consisting of:

$S \to aSt|tSa|cSg|gSc|L$
$L \to NNNN$
$N \to a|c|g|t$

The probability of the sequence '*acgtacgtacgt*' is given by:

$P(\text{acgtacgtacgt}) =$
$P(S \Rightarrow aSt \Rightarrow acSgt \Rightarrow acgScgt \Rightarrow acgtSacgt \Rightarrow acgtLacgt \Rightarrow$
$acgtNNNNacgt \Rightarrow acgtaNNNacgt \Rightarrow acgtacNNacgt \Rightarrow acgtacgNacgt \Rightarrow$
$acgtacgtacgt) =$
$0.2 \times 0.2 \times 0.2 \times 0.2 \times 0.2 \times 1 \times 0.25 \times 0.25 \times 0.25 \times 0.25 = 1.25 \times 10^{-6}$

Note that this sequence has only one possible leftmost derivation under grammar G, and therefore the sum consists of a single term.

10.3 The Parsing Problem

Given a grammar G and a string x, we would like to know:

- Can G derive x?
- If so, what series of productions would be used during the derivation? (There may be multiple answers.)

When G is an SCFG, additional questions arise:

- What is the probability that G derives x? $[P(x|G)]$
- What is the most probable derivation of x via G? $[argmax_j(P(S \Rightarrow_j^* x|G))]$

10.4 Solving the Parsing Problem: The CYK Algorithm

(Cocke and Schwartz[2], 1970; Younger[7], 1967; Kasami, 1965[5])
The CYK algorithm is a dynamic programming algorithm which utilizes the fact that all subtrees of a given parse tree span a contiguous subsequence segment. Given a grammar $G = (V, \Sigma, S, R)$ in CNF, we build a dynamic programming matrix D, such that $D_{i,j}$ holds the set of non-terminals that could derive the subsequence $x[i, j]$. We start with subsequences of length one, and increase subsequence length by one each iteration. We initialize all cells $D_{i,i}$ to contain all non-terminals which produce x_i. The remainder of the DP matrix is then

computed along diagonals parallel to the main diagonal. Computing cell $D_{i,j}$, we attempt to find all partitions: $x[i,k], x[k+1,j]$ and productions such that:

$$(A \rightarrow BC) \in R \tag{10.1}$$

$$B \Rightarrow^* x[i,k] \tag{10.2}$$

$$C \Rightarrow^* x[k+1,j] \tag{10.3}$$

Note that

$$B \Rightarrow^* x[i,k] \text{ iff } B \in D_{i,k} \tag{10.4}$$

$$C \Rightarrow^* x[k+1,j] \text{ iff } C \in D_{k+1,j} \tag{10.5}$$

It follows that G generates x iff $S \in D_{1,n}$, which signifies $S \Rightarrow^* x$.

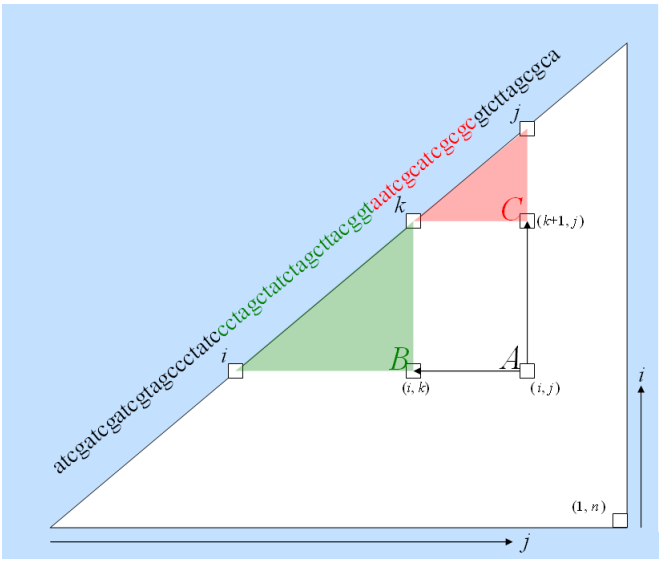

Figure 10.2: **The CYK algorithm:** $D_{i,j}$ contains all the non-terminals X which can derive the entire subsequence: 'actagctatctagcttacggtaatcgcatcgcgc'. $D_{k+1,j}$ contains only those non-terminals which can derive the red subsequence. $D_{i,k}$ contains only those non-terminals which can derive the green subsequence. *Image by B. Majoros.*

10.4.1 Pseudocode for the CYK Algorithm

1. Initialization:

$$\forall 1 \leq i \leq n, D_{i,i} = \{A | A \rightarrow x_i \in R\}$$

2. For each i = n-1 down to 1 do

   For each j = i+1 up to n do

   $$D_{i,j} = \{A | A \to BC \in R, \text{ s.t. } B \in D_{i,k} \wedge C \in D_{k+1,j}, i \leq k < j\}$$

3. Termination:

   $S \Rightarrow^* x$ iff $S \in D_{1,n}$

Complexity: We denote the number of non-terminals by $m = |V|$ and the length of the sequence x by $n = |x|$. Note that in CNF the number of production rules is at most $|R| = m^3 + m|\Sigma| = O(m^3)$ since each rule is either of the form $A \to BC$ or $A \to a$.
For each of the $O(n^2)$ cells of the DP matrix we check all combinations of $O(n)$ partitions and $O(m^3)$ rules, hence total time complexity is $O(n^3 m^3)$. For each cell we hold the set of possible non-terminals deriving the subsequence $x[i, j]$. As there are potentially $O(m)$ such non-terminals, total space complexity is $O(n^2 m)$.

10.5 Extending HMMs for SCFG

SCFGs extend CFGs in much the same way HMMs extended regular grammars. We will now present the inside and outside algorithms[1], which are analogous to the forward and backward algorithms used in HMMs. The SCFG model can be viewed as inferring the hidden data (the non-terminals) from the observed data (the sequence of terminals), gradually reconstructing the parse tree from the bottom up, ending at the root.

10.6 The Inside Algorithm

The inside algorithm is an analog of the forward algorithm. It evaluates the probability of deriving the subsequence $x[i, j]$ given that the root of the corresponding subtree is X. We denote this as:

$$\alpha(i, j, X) = P(X \Rightarrow^* x[i, j]|G) = P(x[i, j]|X_{ij}, G) \tag{10.6}$$

where X_{ij} is the event of the non-terminal X being the root of the subtree spanning $x[i, j]$. Computing $\alpha(i, j, X)$ is similar to the computation of $D_{i,j}$ in the CYK algorithm (from the leaves of the parse tree towards its root). The difference is that if previously we accumulated all partitions and productions that allow for the derivation of $x[i, j]$ from X, we now accumulate the *probabilities* of all such combinations.

$$\alpha(i, j, X) = \sum_{Y,Z} \sum_{k=i}^{j-1} P(X \to YZ) P(x[i, k]|Y_{ik}, G) P(x[k + 1, j]|Z_{(k+1)j}, G) \tag{10.7}$$

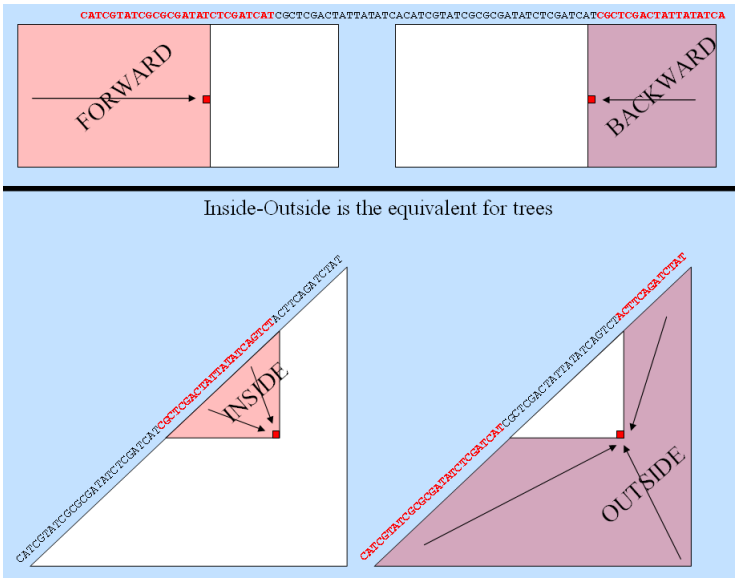

Figure 10.3: The inside and outside algorithms are analogous to the forward and backward algorithms used in HMMs. *Image by B. Majoros.*

With probability $P(X \rightarrow YZ)$ we use the rule $X \rightarrow YZ$, in which case we know that both Y_{ik} and $Z_{(k+1)j}$ occur. But from (10.7) we know that the probability $P(x[i,k]|Y_{ik}, G)$ of Y being the root of the subtree spanning $x[i,k]$ is exactly $\alpha(i,k,Y)$, which was calculated in a previous iteration. Similarly, $P(x[k+1,j]|Z_{(k+1)j}, G) = \alpha(k+1,j,Z)$.

10.6.1 Pseudocode for the Inside Algorithm

1. Initialization:
   for $i = 1$ up to n do

   foreach non-terminal X do

   $\alpha(i,i,X) = P(X \rightarrow x_i)$

2. for $i = n-1$ down to 1 do

   for $j = i+1$ up to n do

   foreach non-terminal X do

   $$\alpha(i,j,X) = \sum_{Y,Z} \sum_{k=i}^{j-1} P(X \rightarrow YZ)\alpha(i,k,Y)\alpha(k+1,j,Z)$$

The probability $P(x|G)$ of the full input sequence x can then be found in the final cell of the matrix: $\alpha(1, n, S)$.

Complexity: Complexity analysis for the inside algorithm is essentially the same as that of CYK, aside of the fact that we now have to save the probability corresponding with each possible non-terminal. Therefore the complexity is again $O(n^3m^3)$ time and $O(m^2n)$ space.

10.7 The Outside Algorithm

The outside algorithm is an analog of the backward algorithm. It evaluates the probability $\beta(i, j, X)$ of a complete parse tree rooted at S for the complete sequence x, excluding the parse subtree for subsequence $x[i, j]$ rooted at X.

$$\beta(i, j, X) = P(S \Rightarrow^* x_1 \dots x_{i-1} X x_{j+1} \dots x_n) = P(x[1, i-1], X_{ij}, x[j+1, n]|G) \qquad (10.8)$$

During computation we shall make use of $\alpha(i, j, X)$ values, applying the inside algorithm is therefore a precondition for outside computation.

We initialize $\beta(1, n, S)$ to one, since the root of all parse trees is the start non-terminal S. The rest of the DP matrix is filled in diagonals from the root towards the leaves.
A parse in which X_{ij} occurs has one of two forms:

- X_{ij} is the left child of a rule $Y \to XZ$. In this case

  $$S \Rightarrow^* x_1 \dots x_{i-1} Y x_{k+1} \dots x_n \Rightarrow x_1 \dots x_{i-1} X Z x_{k+1} \dots x_n \Rightarrow x_1 \dots x_{i-1} X x_{j+1} \dots x_k x_{k+1} \dots x_n$$

  Note that $Z \Rightarrow^* x_{j+1}..x_k$ (this means Z_{ik} occurs).
  The probability of this event is therefore

  $$P(x_1..x_{i-1} Y x_{k+1}..x_n) P(Y \to XZ) P(Z \Rightarrow^* x_{j+1}..x_k) = \beta(i, k, Y) P(Y \to XZ) \alpha(j+1, k, Z) \qquad (10.9)$$

- X_{ij} is the right child of a rule $Y \to ZX$. In a similar fashion, the probability of this event is

  $$P(x_1..x_{k-1} Y x_{j+1}..x_n) P(Y \to ZX) P(Z \Rightarrow^* x_k..x_{i-1}) = \beta(k, j, Y) P(Y \to ZX) \alpha(k, i-1, Z) \qquad (10.10)$$

Each iteration, the length of the excluded sequence diminishes by one ($j - i$ decreases). At the end of the computation, the value $\beta(i, i, X)$ represents $P(x_1..x_{i-1}Xx_{i+1}..x_n)$, thus we get

$$P(x|G) = \sum_{X \in V} P(X \to x_i)\beta(i, i, X) \tag{10.11}$$

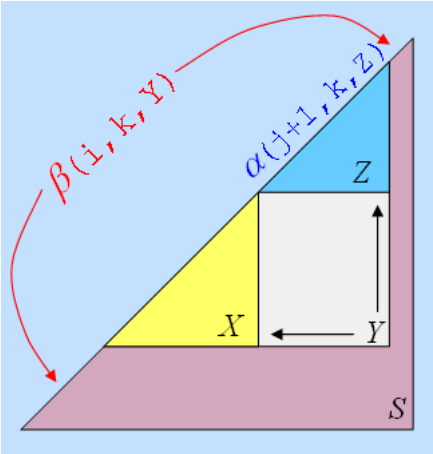

Figure 10.4: Example of a parse where X_{ij} is the left child of a rule $Y \to XZ$. The probability of this event is $\beta(i, k, Y)P(Y \to XZ)\alpha(j + 1, k, Z)$. *Image by B. Majoros*

10.7.1 Pseudocode for the Outside Algorithm

1. Initialization:

   $\beta(1, n, S) = 1$
   foreach $X \neq S$ set $\beta(1, n, X) = 0$

2. for $i = 1$ up to n do

   for $j = n$ down to i do

   foreach non-terminal X do

$$\beta(i, j, X) = \sum_{Y,Z \in V} \sum_{k=j+1}^{n} P(Y \to XZ)\alpha(j + 1, k, Z)\beta(i, k, Y) +$$

$$\sum_{Y,Z \in V} \sum_{k=1}^{i-1} P(Y \to ZX)\alpha(k, i - 1, Z)\beta(k, j, Y)$$

3. Termination (for any i): $P(x|G) = \sum_{X \in V} P(X \to x_i)\beta(i,i,X)$

Complexity: Time and space complexity for the outside algorithm are identical to those of the inside algorithm.

10.7.2 Inferring the Probabilities X_{ij}

We now wish to infer the probabilities of all events X_{ij}. These are informative, since they allow us to evaluate the probability of specific events leading to the final structure of the RNA molecule (in our case). For example, we could give the likelihood that a certain subsequence forms a stem-loop structure rather than a bifurcation of two smaller secondary structures. We wish to calculate $P(X_{ij}|x,G)$. By Bayes rule:

$$P(X_{ij}|x,G) = \frac{1}{P(x|G)} P(X_{ij},x|G) \tag{10.12}$$

applying the chain rule we get

$$P(X_{ij}|x,G) = \frac{1}{P(x|G)} P(x_1..x_{i-1}Xx_{j+1}..x_n|G)P(x_i..x_j|x_1..x_{i-1}Xx_{j+1}..x_n) \tag{10.13}$$

Since we are dealing with a context free grammar, the probability of applying a certain rule to a non-terminal X is independent of the flanking sequences w, z.

$$P(X \to YZ) = P(wXz \to wYZz) \tag{10.14}$$

It follows that

$$P(x_i..x_j|x_1..x_{i-1}Xx_{j+1}..x_n) = P(x_i..x_j|X_{ij}) \tag{10.15}$$

Going back to the full expression we get

$$P(X_{ij}|x,G) = \frac{1}{P(x|G)} P(x_1..x_{i-1}Xx_{j+1}..x_n|G)P(x_i..x_j|X_{ij}) \tag{10.16}$$

In summation:

$$P(X_{ij}|x,G) = \frac{1}{P(x|G)} \beta(i,j,X)\alpha(i,j,X) \tag{10.17}$$

10.8 Parameter Estimation Using EM

We seldom know the exact probability of each production rule. The parameters of our model (probabilities of the different production rules) are usually estimated given a training set of sequences. As was the case with HMMs, the difficulty lies in the fact that even for our

training sequences, the hidden states are unknown. We do not know which non-terminal is responsible for the emission of which character sequence.

We use a version of the EM algorithm in order to estimate the model's parameters:

1. Give an initial estimate of the parameters. Using this initial estimate, apply the inside and outside algorithms to the sequences of the training set.
2. Summing over all parse trees deriving the training set, calculate the following expectations:
   - Expectation of the number of times each rule is applied: $E(X \to YZ)$.
   - Expectation of the number of times a non-terminal X appears: $E(X)$.
3. Having calculated these values, give a new estimate of the parameters and update the model:
$$P_{new}(X \to YZ) = E(X \to YZ)/E(X) \tag{10.18}$$
4. Repeat the process iteratively, until convergence is reached (the likelihood of the training set given the new estimate increases only marginally).

10.8.1 Calculating $E(X \to YZ)$

Applying the rule $X \to YZ$ at position (i, j) is described by this derivation:

$$S \Rightarrow^* x_1..x_{i-1}Xx_{j+1}..x_n \Rightarrow x_1..x_{i-1}YZx_{j+1}..x_n \Rightarrow^*$$

$$x_1..x_{i-1}x_i..x_kZx_{j+1}..x_n \Rightarrow^* x_1..x_{i-1}x_i..x_kx_{k+1}..x_jx_{j+1}..x_n = x$$

for some $i \leq k < j$. The probability of this event is therefore:

$$P(X_{ij}, X \to YZ|x, G) = \frac{1}{P(x|G)}P(X_{ij}, X \to YZ, x|G) \tag{10.19}$$

$$= \frac{1}{P(x|G)}\sum_{k=i}^{j-1}\beta(i,j,X)P(X \to YZ)\alpha(i,k,Y)\alpha(k+1,j,Z) \tag{10.20}$$

Summing over all possible positions (i, j) we get the expectation of $X \to YZ$:

$$E(X \to YZ) = \frac{1}{P(x|G)}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=i}^{j-1}\beta(i,j,X)P(X \to YZ)\alpha(i,k,Y)\alpha(k+1,j,Z) \tag{10.21}$$

10.8.2 Calculating $E(X \to a)$

Applying the rule $X \to a$ at position (i,i) (the non-terminal X emits $a = x_i$) is described by this derivation:

$$S \Rightarrow^* x_1..x_{i-1}Xx_{i+1}..x_n \Rightarrow x_1..x_{i-1}a\ x_{i+1}..x_n$$

The probability of this event is therefore:

$$P(X_{ii}, X \to a|x, G) = \frac{1}{P(x|G)}P(X_{ii}, X \to a, x|G) \tag{10.22}$$

$$= \frac{1}{P(x|G)}\delta(x_i, a)\beta(i,i,X)P(X \to a) \tag{10.23}$$

Where $\delta(x_i, a) = \begin{cases} 1 & \text{if } x_i = a \\ 0 & \text{otherwise} \end{cases}$

Summing over all possible positions (i,i) we get the expectation of $X \to a$:

$$E(X \to a) = \frac{1}{P(x|G)}\sum_{i=1}^{n}\delta(x_i, a)\beta(i,i,X)P(X \to a) \tag{10.24}$$

10.8.3 Calculating $E(X)$

The expectation of the non-terminal X is the sum of $P(X_{ij}|x, G)$ over all positions (i,j). That is:

$$E(X) = \sum_{i=1}^{n}\sum_{j=1}^{n}P(X_{ij}|x, G) \tag{10.25}$$

From (10.17) we get:

$$E(X) = \frac{1}{P(x|G)}\sum_{i=1}^{n}\sum_{j=1}^{n}\beta(i,j,X)\alpha(i,j,X) \tag{10.26}$$

10.8.4 EM Update Equations

We deduce the following EM update equations:

$$P_{new}(X \to YZ|X) = \frac{E(X \to YZ)}{E(X)} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=i}^{j-1}\beta(i,j,X)P(X \to YZ)\alpha(i,k,Y)\alpha(k+1,j,Z)}{\sum_{i=1}^{n}\sum_{j=1}^{n}\beta(i,j,X)\alpha(i,j,X)} \tag{10.27}$$

$$P_{new}(X \to a|X) = \frac{E(X \to a)}{E(X)} = \frac{\sum_{i=1}^{n}\delta(x_i,a)\beta(i,i,X)P(X \to a)}{\sum_{i=1}^{n}\sum_{j=1}^{n}\beta(i,j,X)\alpha(i,j,X)} \tag{10.28}$$

10.9 Covariance Models

Covariance models[3] provide a general modeling scheme for RNA families, using SCFGs. The problem we shall address is creating an SCFG model for the consensus structure of an RNA family. CMs will also provide us with the means to decide whether a given RNA sequence conforms to the consensus structure, and assign that decision with a likelihood score.

10.9.1 A Basic SCFG for RNA Structure

Consider an SCFG with 7 state types, as shown in figure (10.5):

State type	Description	Production	Emission	Transition
P	(pair emitting)	$P \rightarrow aYb$	$e_v(a,b)$	$t_v(Y)$
L	(left emitting)	$L \rightarrow aY$	$e_v(a)$	$t_v(Y)$
R	(right emitting)	$R \rightarrow Ya$	$e_v(a)$	$t_v(Y)$
B	(bifurcation)	$B \rightarrow SS$	1	1
D	(delete)	$D \rightarrow Y$	1	$t_v(Y)$
S	(start)	$S \rightarrow Y$	1	$t_v(Y)$
E	(end)	$E \rightarrow \varepsilon$	1	1

Figure 10.5: A basic SCFG that describes RNA folding behavior[4].

It can efficiently describe the typical behavior of RNA folding:

- P states emit base paired nucleotides (stems).
- L, R states emit unpaired nucleotides (single strands, loop segments, bulge, dangling end).
- B states allow for bifurcations (junctions).
- S states represent the start of a new structure, at the root of the tree or immediately following a bifurcation.
- D states allow for deletions with respect to the consensus.
- E states allow for the termination of a structure (note that emitting states only allow us to extend a structure)

Note, however, that these state types are generic and not position specific. While this grammar represents the probabilities of transitions from one state to the other, it does not allow for different behaviors at different positions along the consensus, e.g. "position 5 is more likely to be part of a stem", "position 4 is more likely to contain a purine", etc.

10.9.2 A Simple Example

Here is a simple SCFG which demonstrates how, by assigning uniform probabilities, we can favor a parse tree which maximizes base pairing. In effect, this simple grammar imitates Nussinov's algorithm.

$$G = (V_G, \Sigma, S, R_G, P_G)$$
$$V_G = \{S, P, L, R, B, E\}$$
$$\Sigma = \{a, c, g, u\}$$
$$R_G = \left\{ \begin{array}{l} S \rightarrow P|L|R|B|E \\ P \rightarrow aSu|uSa|cSg|gSc \\ L \rightarrow aS|cS|gS|uS \\ R \rightarrow Sa|Sc|Sg|Su \\ B \rightarrow SS \\ E \rightarrow \epsilon \end{array} \right\}$$

For each non-terminal, all productions have equal probabilities. Specifically, all emitting productions have probability $\frac{1}{4}$. Lets examine two possible parses of the string 'cacgug':

1. $\boldsymbol{S \Rightarrow_1^* cacgug}$: $S \Rightarrow P \Rightarrow cSg \Rightarrow cPg \Rightarrow caSug \Rightarrow caPug \Rightarrow cacSgug \Rightarrow cacEgug \Rightarrow cacgug$
   This derivation has the probability:
   $\frac{1}{5} \times \frac{1}{4} \times \frac{1}{5} \times \frac{1}{4} \times \frac{1}{5} \times \frac{1}{4} \times \frac{1}{5} \times 1 = \frac{1}{40000} = 2.5 \times 10^{-5}$
2. $\boldsymbol{S \Rightarrow_2^* cacgug}$: $S \Rightarrow L \Rightarrow cS \Rightarrow cR \Rightarrow cSg \Rightarrow cLg \Rightarrow caSg \Rightarrow caRg \Rightarrow caSug \Rightarrow caLug \Rightarrow cacSug \Rightarrow cacRug \Rightarrow cacSgug \Rightarrow cacEgug \Rightarrow cacgug$
   This derivation has the probability:
   $\frac{1}{5} \times \frac{1}{4} \times \frac{1}{5} \times \frac{1}{4} \times \frac{1}{5} \times \frac{1}{4} \times \frac{1}{5} \times \frac{1}{4} \times \frac{1}{5} \times \frac{1}{4} \times \frac{1}{5} \times \frac{1}{4} \times \frac{1}{5} \times 1 = \frac{1}{3.2 \times 10^8} = 3.125 \times 10^{-9}$

The first parse is significantly more likely. All emitting productions have the general form $S \Rightarrow X \Rightarrow wSz$ (where $X \in V_G$ and $w, z \in \Sigma \cup \{\epsilon\}$) and therefore a probability of $\frac{1}{20}$. For this reason, the most probable parse contains a minimal number of productions. The pair emitting state is thus preferred over the left/right emitting states, whenever possible. Hence the most probable parse yields maximal base pairing.

Note: This is not entirely accurate, since this model puts a "probability penalty" on branching. The branching itself decreases the probability of the parse by a factor of 5 ($S \Rightarrow B \Rightarrow SS$). Terminating the new branch further decreases the probability by a factor of 5 ($S \Rightarrow E \Rightarrow \epsilon$). Nussinov's algorithm has no such penalty.

10.9.3 Extending the Basic Grammar

To allow for position specific behavior, each production along the parse tree generating the consensus is assigned a unique non-terminal. These non-terminals are called nodes. Eight

node types are defined, in accordance with the basic state types:

Node	Description	Main state type
MATP	(pair)	P
MATL	(single strand, left)	L
MATR	(single strand, right)	R
BIF	(bifurcation)	B
ROOT	(root)	S
BEGL	(begin, left)	S
BEGR	(begin, right)	S
END	(end)	E

Figure 10.6: Eight different node types allow for position specific behavior[4].

Each node is given a (unique) name, comprised of a node type and a serial number, e.g. MATL 2, signifying that the 2nd production along the parse tree should be a left emission. When constructing the parse tree, we prefer MATL nodes over MATR nodes. This is rather an arbitrary choice, meant to reduce ambiguity, similar to our choice of leftmost derivations. Figure 10.7 illustrates a parse tree corresponding with a given consensus structure. Note that this kind of parse tree cannot derive sequences deviating from the consensus – it contains no insertion/deletion nodes.

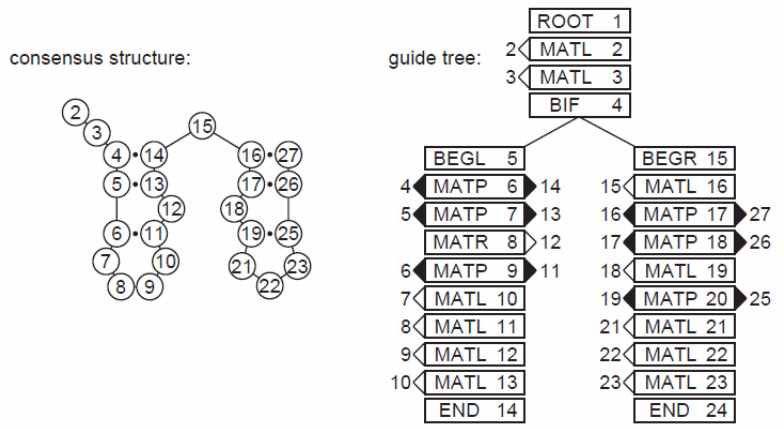

Figure 10.7: Creating a guide (parse) tree for a consensus structure[4].

10.9.4 The Final CM

In order to account for deviations from the consensus, each node is expanded into several states, divided into 2 sets:

- A split set — Represents the main consensus state. A parse tree must visit exactly one state from each split set. The split set allows for deletions: a split state may emit less symbols (terminals) than indicated by the node type.
- An insert set — Allows for insertions with respect to the consensus. A parse tree may pass through the insert states zero or more times.

Figure 10.8 details the translation of node types to split and insert sets.

Node	States
MATP	[MP ML MR D] IL IR
MATL	[ML D] IL
MATR	[MR D] IR
BIF	[B]
ROOT	[S] IL IR
BEGL	[S]
BEGR	[S] IL
END	[E]

Figure 10.8: Split and insert sets corresponding with different node types. The split set is written in square brackets[4].

Different split states signify the emission/deletion of consensus symbols. For example, a pair emitting node of type MATP may emit both symbols (MP), only the left symbol (ML), only the right symbol (MR), or none of the two (D). Only emitting nodes (MATP, MATL, MATR) may have deletions. Non-emitting nodes cannot have deletions and so their split sets are singletons.

Transitions:

- A split state can transit to an insert state in the same node or a split state in the next node.
- An IL state can transit to itself, an IR in the same node or a split state in the next node.
- An IR can transit to itself or to a split state in the next node.

Figure 10.9 depicts the resulting CM.

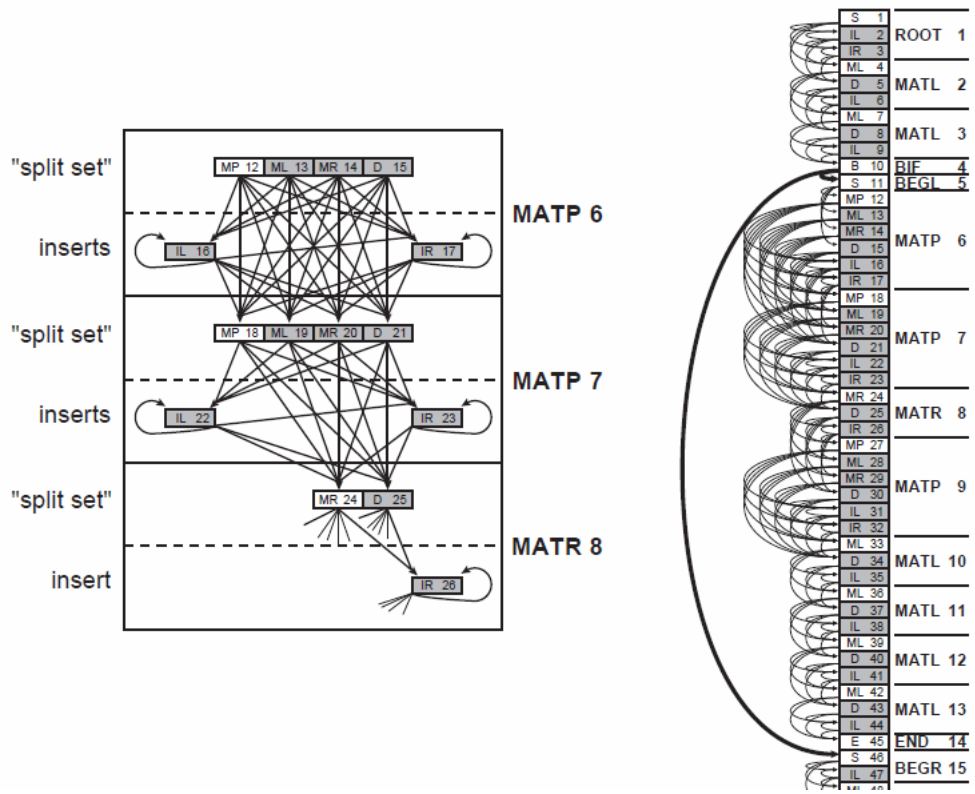

Figure 10.9: Expanding the nodes to create the final CM. On the right: CM created by expanding the guide tree in figure 10.7. On the left: An enlarged segment of the CM, depicting the states and transitions of nodes MATP 6, MATP 7, MATR 8[4].

10.10 Specialized Inside/Outside Algorithms for CMs

Unlike CNF grammars, CMs follow a strict form. Specialized versions of the inside and outside algorithms can be made to utilize that strictness of form to compute posterior probabilities $P(X_{ij}|x, G)$ more efficiently.

Notations for this section:

- A CM has n states (non-terminals) $W_1, W_2, \ldots, W_m$. Each state pertains to one of the 7 state types defined earlier. W_1 is the (root) start state for the whole CM. There may be several end states as a CM usually represents a multi-branched structure, each branch (e.g. stem-loop) having its own end state.
- $s_v \in \{P, L, R, D, S, B, E\}$ denotes the type of state v.

- End states can generate the null sequence ϵ. In fact, all parses end with the production $W_v \rightarrow \epsilon$ for some $v, s_v = E$. It is therefore necessary to assign a probability to the generation of the null sequence. Let $x[j+1, j] = \epsilon$ denote the null subsequence, preceding the terminal x_j.
- $\Delta_v^L, \Delta_v^R \in \{0, 1\}$ denote the number of terminals emitted to the left and right by state W_v.
- $t_v(y)$ denotes the probability of transition $P(W_v \rightarrow W_y)$.
- $e_v(x_i, x_j)$ denotes the probability of emitting the terminal x_i to the left and x_j to the right, by state v. This notation will be used for all emission probabilities (regardless of whether x_i, x_j are emitted or not):
  - for P states $e(x_i, x_j) = e(x_i)e(x_j)$
  - for L states $e(x_i, x_j) = e(x_i)$
  - for R states $e(x_i, x_j) = e(x_j)$
  - for non-emitting states $e(x_i, x_j) = 1$
- $\mathcal{C}_v$ denotes the children of state W_v. $\mathcal{C}_v$ is an ordered list of indices y for the states W_y that W_v can make a transition to.
- $\mathcal{P}_v$ denotes the children of state W_v. $\mathcal{P}_v$ is an ordered list of indices y for the states W_y that can make a transition to W_v.
- θ denotes the model's parameters – the probabilities corresponding with each production and emission.

State (s_v)	Production	Δ_v^L	Δ_v^R	Emission	Transition
P	$W_v \rightarrow x_i W_y x_j$	1	1	$e_v(x_i, x_j)$	$t_v(y)$
L	$W_v \rightarrow x_i W_y$	1	0	$e_v(x_i)$	$t_v(y)$
R	$W_v \rightarrow W_y x_j$	0	1	$e_v(x_j)$	$t_v(y)$
D	$W_v \rightarrow W_y$	0	0	1	$t_v(y)$
S	$W_v \rightarrow W_y$	0	0	1	$t_v(y)$
B	$W_v \rightarrow W_y W_z$	0	0	1	1
E	$W_v \rightarrow \epsilon$	0	0	1	1

Figure 10.10: Properties of all state types according to our notation[4].

Figure 10.10 summarizes the properties of all 7 state types.
Lets examine some of the qualities of CMs:

1. **A state has 6 productions at the most** — A split state can transition to insert states in the same node (at most 2), or to split states in the next node (at most 4). A general SCFG state may have as many as $O(m^2)$ productions. We find that in CMs, although the number of states is $m = O(n)$, each state has only a constant number of productions.

2. B **states have only one production** — A bifurcation (BIF) node always transits to 2 start nodes (BEGL, BEGR). All three node types are non-emitting, have singleton split sets, and no insert sets. So we get a single production (with probability 1) $W_v \to W_y W_z$, where $s_v = B$, $s_y = S$, $s_z = S$.

3. **A single branching production** — The B production is the only one to produce two non-terminals. This is important, since such productions are computationally difficult. Generating the subsequence $x[i, j]$ from W_v using the branching production $W_v \to W_y W_z$ is described by this derivation:

$$W_v \Rightarrow W_y W_z \Rightarrow^* x_i..x_k W_z \Rightarrow^* x_i..x_k x_{k+1}..x_j$$

   We have to consider all partitions $x[i, k]$,$x[k+1, j]$ such that $W_y \Rightarrow^* x[i, k]$ and $W_z \Rightarrow^* x[k+1, j]$. This may take $O(n)$ operations.

4. **Ordered states** — Transitions are only possible from a node to itself or the next. This dictates that $y > v$ for all indices $y \in \mathcal{C}_v$, or $y \geq v$ for insert states. This prevents futile cycles composed of non-emitting states.

10.10.1 Inside Algorithm for CMs

The specialized inside algorithm computes $\alpha(i, j, W_v)$ differently for each state type s_v.
Non-emitting states may generate the null sequence. It is therefore necessary to calculate the probability of deriving sequences of length 0, $\alpha(j+1, j, W_v)$.
Generating the null sequence ϵ:

- $\boldsymbol{s_v = E}$: An end state can only generate a null sequence, so

$$P(W_v \Rightarrow \epsilon | s_v = E) = 1$$

- $\boldsymbol{s_v \in S, D}$: Start and deletion states cannot generate a null sequence directly, but may, through non-emitting productions, derive an end state. Assume $y \in \mathcal{C}_v$ and

$s_z = E$ (possibly $y = z$), the derivation is then:

$$W_v \Rightarrow W_y \Rightarrow^* W_z \Rightarrow \epsilon$$

The probability of this event is:

$$P(W_v \Rightarrow W_y, W_y \Rightarrow^* \epsilon|\theta) = P(W_v \Rightarrow W_y)P(W_y \Rightarrow^* \epsilon) = t_v(y)\alpha(j+1, j, W_y) \quad (10.29)$$

Summing over all possible $y \in \mathcal{C}_v$ we get:

$$\alpha(j+1, j, W_v) = \sum_{y \in \mathcal{C}_v} t_v(y)\alpha(j+1, j, W_y) \quad (10.30)$$

- $\boldsymbol{s_v = B}$: A bifurcation state may derive two empty structures through this derivation:
  $W_v \Rightarrow W_y W_z \Rightarrow^* \epsilon W_z \Rightarrow^* \epsilon\epsilon$
  Since $P(W_v \Rightarrow W_y W_z) = 1$, the probability of this event is:

$$P(W_v \Rightarrow^* \epsilon) = \alpha(j+1, j, W_y)\alpha(j+1, j, W_z) \quad (10.31)$$

- $\boldsymbol{s_v \in P, L, R}$: Emitting states cannot generate a null sequence, so

$$P(W_v \Rightarrow^* \epsilon | s_v \in \{P, L, R\}) = 0$$

10.10.2 Pseudocode for the Specialized Inside Algorithm

1. Initialization:

   for $j = 0$ up to n do

   for $v = m$ down to 1 do

$$\alpha(j+1, j, W_v) = \begin{cases} s_v = E: & 1 \\ s_v = S, D: & \sum_{y \in \mathcal{C}_v} t_v(y)\alpha(j+1, j, W_y) \\ s_v = B: & \alpha(j+1, j, W_y)\alpha(j+1, j, W_z) \\ s_v = P, L, R: & 0 \end{cases}$$

2. for $i = n$ down to 1 do

   for $j = i$ up to n do

for $v = m$ down to 1 do

$$\alpha(i,j,W_v) = \begin{cases} s_v = E: & 0 \\ s_v = P, j = i: & 0 \\ s_v = B: & \sum_{k=i-1}^{j} \alpha(i,k,W_y)\alpha(k+1,j,W_z) \\ \text{otherwise}: & e_v(x_i,x_j)\sum_{y\in\mathcal{C}_v} t_v(y)\alpha(i+\Delta_v^L, j-\Delta_v^R, W_y) \end{cases}$$

Complexity: For each of the $O(n^2)$ cells of the DP matrix we have to compute $\alpha(i,j,W_v)$ for all $1 \le v \le m$. This takes $O(n)$ time if $s_v = B$ and $O(1)$ otherwise (since $|\mathcal{C}_v| \le 6$). Suppose the CM has b bifurcation states and a other states ($a+b=m$), then total time complexity is $O(an^2 + bn^3)$. Space complexity is $O(mn^2)$.

10.10.3 Outside algorithm for CMs

The specialized outside algorithm seems to be simpler than the general one, since most productions have a single child (non-terminal). Actually, this is true for all productions save one – the branching production. Branching always yields two S type states, so these require special attention. Outside computation requires values claculated by the inside algorithm, unlike the general version, the specialized outside only requires $\alpha(i,j,W_z)$ for $s_z = S$.

Computing $\boldsymbol{\beta(i,j,W_v)}$:

- $\boldsymbol{s_v \in P,L,R,B,D,E}$: Assume W_v is generated by the pair emitting production $W_y \to x_{i-1}W_v x_{j+1}$ for some $y \in \mathcal{P}_v$, $s_y = P$. The event $W_1 \Rightarrow^* x_1..x_{i-1}W_v x_{j+1}$ corresponds with the derivation:

$$W_1 \Rightarrow^* x_1..x_{i-2}W_y x_{j+2}..x_n \Rightarrow x_1..x_{i-2}x_{i-1}W_v x_{j+1}x_{j+2}..x_n$$

The probability of this event is:

$$\begin{aligned} P(x_1..x_{i-1}W_v x_{j+1}..x_n|\theta) &= P(x_1..x_{i-2}W_y x_{j+2}..x_n)P(W_y \Rightarrow x_{i-1}W_v x_{j+1}) && (10.32)\\ &= \beta(i-1,j+1,W_y)t_y(v)e_y(x_{i-1},x_{j+1}) && (10.33)\end{aligned}$$

In the general case (without asserting $s_v = P$) x_{i-1}, x_{j+1} may or may not have been emited by W_y, depending on Δ_y^L, Δ_y^R. we then get:

$$P(x_1..x_{i-1}W_v x_{j+1}..x_n|\theta) = \beta(i-\Delta_y^L, j+\Delta_y^R, W_y)t_y(v)e_y(x_{i-\Delta_y^L}, x_{j+\Delta_y^R}) \quad (10.34)$$

Summing over all possible $y \in \mathcal{P}_v$ we get:

$$\beta(i,j,W_v) = \sum_{y\in\mathcal{P}_v} \beta(i-\Delta_y^L, j+\Delta_y^R, W_y)t_y(v)e_y(x_{i-\Delta_y^L}, x_{j+\Delta_y^R}) \quad (10.35)$$

- $s_v = S$, W_v **is the root**: The root W_1 has $\beta(1, n, W_1) = 1$ and $\beta(i, j, W_1) = 0$ for all other values of (i, j).

- $s_v = S$, W_v **is a left child**: A left child is generated by the branching production $W_y \rightarrow W_v W_z$ where $s_y = B, s_z = S$ and W_y has $\mathcal{C}_y = \{v, z\}$. The event $W_1 \Rightarrow^* x_1..x_{i-1} W_v x_{j+1}$ corresponds with the derivation:

$$W_1 \Rightarrow^* x_1..x_{i-1} W_y x_{k+1}..x_n \Rightarrow x_1..x_{i-1} W_v W_z x_{k+1}..x_n \Rightarrow^* x_1..x_{i-1} W_v x_{j+1}..x_k x_{k+1}..x_n$$

  for some $j \leq k \leq n$. The probability of this event is:

$$\begin{aligned} P(x_1..x_{i-1} W_v x_{j+1}..x_n|\theta) = & \quad (10.36) \\ = P(x_1..x_{i-1} W_y x_{k+1}..x_n) P(W_y \Rightarrow W_v W_z) P(W_z \Rightarrow^* x_{j+1}..x_k) & \quad (10.37) \\ = \beta(i, k, W_y) \times 1 \times \alpha(j+1, k, W_z) & \quad (10.38) \end{aligned}$$

  The above derivation is not leftmost, yet the order of derivation does not affect the probability of events. Summing over all possible $j \leq k \leq n$ we get:

$$\beta(i, j, W_v) = \sum_{k=j}^{n} \beta(i, k, W_y) \alpha(j+1, k, W_z) \quad (10.39)$$

  The first term of the sum $(k = j)$ represents a case where W_z generates a null sequence.

- $s_v = S$, W_v **is a right child**: A right child is generated by the branching production $W_y \rightarrow W_z W_v$ where $s_y = B, s_z = S$ and W_y has $\mathcal{C}_y = \{z, v\}$. Following the guidelines of the previous case we get:

$$\beta(i, j, W_v) = \sum_{k=1}^{i} \beta(k, j, W_y) \alpha(k, i-1, W_z) \quad (10.40)$$

  The last term of the sum $(k = i)$ represents a case where W_z generates a null sequence.

10.10.4 Pseudocode for the specialized outside algorithm

1. Initialization:

$$\beta(1, n, W_1) = 1$$

2. for $i = 1$ up to $n + 1$ do

   for $j = n$ down to $i - 1$ do

   for $v = 2$ up to m do

$$\beta(i,j,W_v) = \begin{cases} \text{for } s_v = S, \mathcal{P}_v = y, \mathcal{C}_y = \{v,z\}: \\ \quad \sum_{k=j}^{n} \beta(i,k,W_y)\alpha(j+1,k,W_z) \\ \text{for } s_v = S, \mathcal{P}_v = y, \mathcal{C}_y = \{v,z\}: \\ \quad \sum_{k=1}^{i} \beta(k,j,W_y)\alpha(k,i-1,W_z) \\ \text{for } s_v \in P, L, R, B, D, E: \\ \quad \sum_{y \in \mathcal{P}_v} \beta(i-\Delta_y^L, j+\Delta_y^R, W_y) t_y(v) e(x_{i-\Delta_y^L}, x_{j+\Delta_y^R}) \end{cases}$$

Complexity: Time and space complexity for the specialized outside algorithm are identical to those of the specialized inside algorithm.

NOTE: For a specialized version of the EM algorithm, refer to "Biological Sequence Analysis", Durbin et al. (Cambridge, '98).

10.11 Creating a Consensus Structure From an MSA

In some cases, the consensus structure of a family of RNA sequences is unknown. For the purpose of creating a CM, we must first construct a consensus structure for the family (or training set). We assume that an MSA of all input sequences is given. Later we will show that any random MSA will suffice.

Notations for this section:

- A denotes an MSA of m RNA sequences $x^1, x^2 \ldots x^m$. We denote $n = |A|$. We assume that $x^1 \ldots x^m$ are gapped according to the MSA. A column of the MSA is considered an insertion if it contains $> 50\%$ gaps (this is a common heuristic).
- A_i denotes the i^{th} column of A. $A[i,j]$ denotes columns i through j of A.
- $P(x_i = c)$ denotes the (empirical) probability of a sequence $x \in A$ having the character c at position i. This can also be described as the relative frequency of the character c in column A_i.

- A consensus structure of A is defined by a set C of index pairs (i, j), such that $1 \leq i < j \leq n$. A pair $(i, j) \in C$ signifies that columns A_i, A_j are base paired. Since any base can only pair with one other base, an index i may only appear once in C.

10.11.1 Mutual Information

The mutual information of two random variables is a quantity that measures the mutual dependence of the two variables. Formally, the mutual information $I(X;Y)$ of two discrete random variables X and Y is defined as follows:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} P(x,y) \log \frac{P(x,y)}{P_1(x)P_2(y)} \tag{10.41}$$

where $P_1(x) \equiv P(X = x), P_2(y) \equiv P(Y = y), P(x,y) \equiv P(X = x, Y = y)$. Examples of maximal and minimal $I(X,Y)$:

- Assume X and Y are independent, then $P(x,y) = P_1(x)P_2(y)$ for all $x \in X, y \in Y$. We get:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) \log \frac{P(x,y)}{P_1(x)P_2(y)} \tag{10.42}$$

$$= \sum_{x \in X} \sum_{y \in Y} P(x,y) \log \frac{P_1(x)P_2(y)}{P_1(x)P_2(y)} \tag{10.43}$$

$$= \sum_{x \in X} \sum_{y \in Y} P(x,y) \log 1 = 0 \tag{10.44}$$

- Assume X can have n possible outcomes $\{a_1 \dots a_n\}$, each occuring with equal probability $\frac{1}{n}$, Y has n outcomes $\{b_1 \dots b_n\}$. Assume that X, Y are completely co-dependent, meaning that for all $1 \leq i \leq n$, a_i occurs iff b_i occurs. We get:

$$I(X;Y) = \sum_{i=1}^{n} \sum_{j=1}^{n} P(a_i, b_j) \log \frac{P(a_i, b_j)}{P_1(a_i)P_2(b_j)} \tag{10.45}$$

$$= \sum_{i=1}^{n} P(a_i, b_i) \log \frac{P(a_i, b_i)}{P_1(a_i)P_2(b_i)} \tag{10.46}$$

$$= \sum_{i=1}^{n} \frac{1}{n} \log \frac{\frac{1}{n}}{\frac{1}{n} \times \frac{1}{n}} = \sum_{i=1}^{n} \frac{1}{n} \log n = \log n \tag{10.47}$$

10.11.2 Mutual Information of MSA Columns

In our context, we use the mutual information bit-score to determine whether two MSA columns should base pair or not. This is reasonable since base-paired columns should show a significant correlation. Adapting the general formula to our context, let $M_{i,j}$ denote the mutual information of columns i and j of an MSA:

$$M_{i,j} = \sum_{a,b=\{A,C,G,U\}} P(x_i = a, x_j = b) \log_2 \frac{P(x_i = a, x_j = b)}{P(x_i = a)P(x_j = b)} \tag{10.48}$$

Notes and observations:

- x_i has 4 possible outcomes $\{A, C, G, T\}$ so $0 \leq M_{i,j} \leq 2$
- For some sequence x^k in the MSA, the character x_i^k or x_j^k (or both) may be deleted. In that case, the probability $P(x_i^k = a, x_j^k = b)$ is undefined. Such sequences are excluded when computing $M_{i,j}$.
- Two degenerate (constant) columns i, j have $M_{i,j} = 0$. Assume $\forall k.x_i^k =$ 'A', $\forall k.x_j^k =$ 'G' for example. Then the sum has a single term:

$$M_{i,j} = P(x_i = \text{'A'}, x_j = \text{'G'}) \log_2 \frac{P(x_i = \text{'A'}, x_j = \text{'G'})}{P(x_i = \text{'A'})P(x_j = \text{'G'})} = 1 \times \log_2 \frac{1}{1 \times 1} = 1 \times 0 = 0 \tag{10.49}$$

10.11.3 Finding Consensus Structure with Maximal $\sum M_{i,j}$

We wish to find a consensus structure C which maximizes $\sum_{(i,j)\in C} M_{i,j}$ (the sum of mutual information values over all paired columns). To accomplish this we use a DP algorithm, similar to Nussinov's maximal base-pairing algorithm; We build a DP matrix S, such that $S_{i,j}$ holds the optimal (in terms of maximum $\sum M_{i,j}$) structure for $A[i, j]$. **General flow of the algorithm:**

1. Initialize all cells $S_{i,i}$ (the leaves) to zero.
2. The matrix is then filled from leaves to root. Computing cell $S_{i,j}$, we choose the best of four options:
   (a) Add unpaired column i at the beginning of the optimal structure for $A[i + 1, j]$.
   (b) Add unpaired column j at the end of the optimal structure for $A[i, j - 1]$.
   (c) Surround the optimal structure for $A[i + 1, j - 1]$ with base paired columns i, j. This increases the structure's score by $M_{i,j}$.

(d) Concatenate two smaller structures $A[i,k], A[k+1,j]$ for some $i < k < j$. $S_{i,j}$ then contains the sum of the two scores.

3. When the matrix has been filled, $S_{1,n}$ holds the optimal score for the complete MSA.

4. Tracing back the path yielding maximal score, we find the optimal structure for A.

The recursion for $S_{i,j}$:

$$S_{i,j} = \max\left\{S_{i+1,j}, S_{i,j-1}, S_{i+1,j-1} + M_{i,j}, \max_{i<k<j}\{S_{i,k} + S_{k+1,j}\}\right\} \tag{10.50}$$

Complexity: for each of the $O(n^2)$ cells of the DP matrix we check all $O(n)$ partitions. Therefore, the total time complexity is $O(n^3)$. Each cell holds a single value, total space complexity is therefore $O(n^2)$.

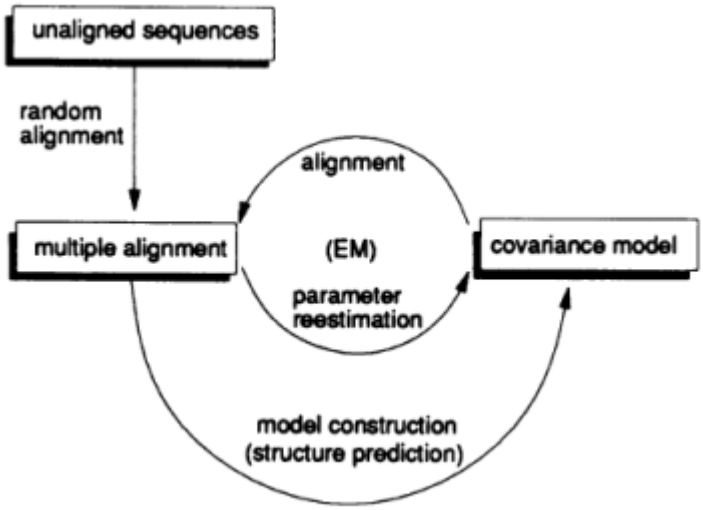

Figure 10.11: The covariance model training algorithm[3]

10.11.4 Working Without an MSA

CMs provide a powerful tool for the modeling of RNA folding. The chink in the armor, so to speak, is having to rely on an initial MSA to construct the CM. MSAs are generic and do not factor the specific behavior of RNA folding in any way. We introduce an iterative approach that allows us to construct a CM from a random MSA (sequences are gapped randomly):

1. A consensus structure is derived from the (initially random) MSA.

2. The model's parameters are optimized by EM.
3. The CM is used to align each of the sequences to the consensus → a new MSA is derived.
4. The process is repeated until convergence.

It should be noted, that although the quality of the CM is retained when using a random MSA, time to convergence increases significantly (when compared to an initial MSA of good quality).

10.12 Application of CMs – tRNA Modeling

tRNAs provide an ideal test case for CM. It is the largest gene family in most genomes, sequences exhibit significant variation while secondary structure is widely conserved. Durbin and Eddy created two covariance models for a data set of 1415 aligned tRNA sequences. 100 tRNA sequences were randomly selected to serve as a test set, an additional group of a 100 sequences was randomly chosen to serve as a training set. Two training modes were used:

- Mode A — CM was trained using a trusted alignment of the training set, incorporating data obtained from X-ray crystalography.
- Mode U — CM was trained using a random alignment of the training set.

The A model converged rapidly and had an average bit score[2] of 58.7. The U model showed a similar score of 56.7 bits, yet took greater time to converge. The results are given in the following table:

Training Mode	# of Iterations	Bit Score	Accuracy
A100	3	57.3	94%
U100	23	56.7	90%

An alignment produced by an HMM trained using the same data yields 30 bits of information, about half the information achieved by the CM alignments. Note that both models achieve at least 90% accuracy[3]. It should be stated that a degapped alignment of the sequences (where sequences are simply placed one "on top" of the other) yields only 30% accuracy.

[2]The information bit score is defined as $\log_2 \frac{P(x|\theta)}{P(r|\theta)}$, where $P(r|\theta)$ is the average probability of a random sequence r with $|r| = |x|$ being generated by the model. Intuitively, 1 information bit means that a sequence x is twice as likely to be generated by the model than a random sequence of equal length.

[3]Accuracy is defined as the percent of symbol pairs aligned according to the CM, that are also aligned according to the trusted alignment.

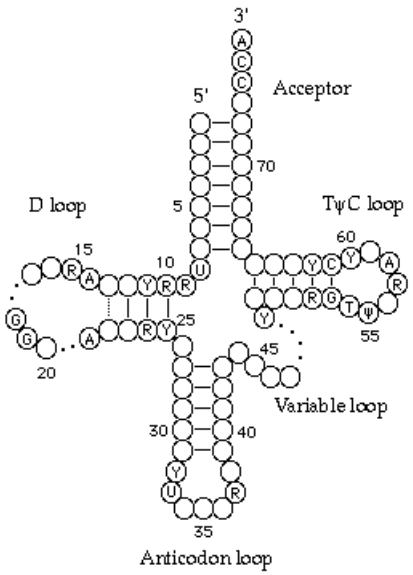

Figure 10.12: tRNA secondary structure.

Next, the A1415 model (trained using all 1415 aligned sequences) was used to classify a test set consisting of sequences from GenBank structural RNA database, and C. elegans genomic sequence (6.3 Mb total). Here is a partial list of the results:

- As seen on figure (10.13), Any cutoff score between 11.7 and 25.9 cleanly separates all the non-tRNAs from the 547 cytoplasmic tRNAs, giving a sensitivity of over 99.98%.
- All 14 C. elegans tRNAs were detected with a score greater than 31 (100% sensitivity).
- 26 out of 522 annotated tRNAs from GenBank were missed (95% sensitivity). All 26 missed tRNAs were mitochondrial, 22 of them completely lack the D-loop.
- Of all annotated non-tRNA molecules, none were mis-classified as tRNAs (100% specificity).

In conclusion, Covariance Models supply an elegant probabilistic solution for predicting RNA secondary structure, showing remarkable results.

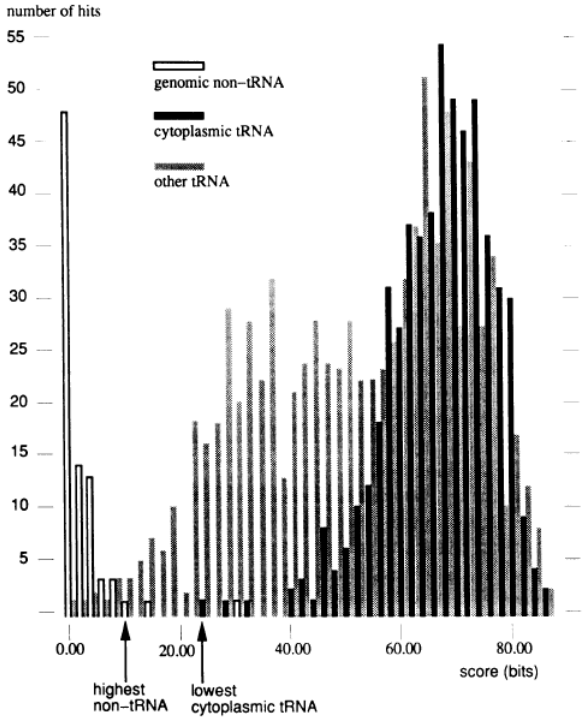

Figure 10.13: Number of hits versus score in bits, using the A1415 model to search sequences from Genbank and C.elegans genomic sequence. Non-tRNA hits are shown in white. Hits to cytoplasmic tRNAs are shown in black. 'Other' tRNA hits, in gray, are the scores of 868 mitochondrial, chloroplast, viral, and selenocysteine tRNAs. Arrows indicate the gap between the highest non-tRNA hit (with two tRNA-related exceptions) and the lowest scoring cytoplasmic tRNA.[3]

Bibliography

[1] J. Baker. Trainable grammars for speech recognition. *Speech communication papers presented at the 97th Meeting of the Acoustical Society of America*, 1979.

[2] John Cocke and Jacob T. Schwartz. Programming languages and their compilers: Preliminary notes. *Technical report, Courant Institute of Mathematical Sciences, New York University*, 1970.

[3] Sean R. Eddy and Richard Durbin. Rna sequence analysis using covariance models. *Computational Biology*, 1994.

[4] R. Durbin et al. *Biological sequence analysis*. Cambridge University Press, 1998.

[5] Tadao Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. *Scientific report AFCRL-65-758*, 1965.

[6] Ruth Nussinov and Ann B. Jacobson. Fast algorithm for predicting the secondary structure of single-stranded rna. *PNAS*, 1980.

[7] Daniel H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 1967.