13 Comparative RNA Analysis

De novo folding approaches alone are not so successful. In case additional homologous sequences are at hand, we can try a *comparative approach*. Basically, there are three ways:

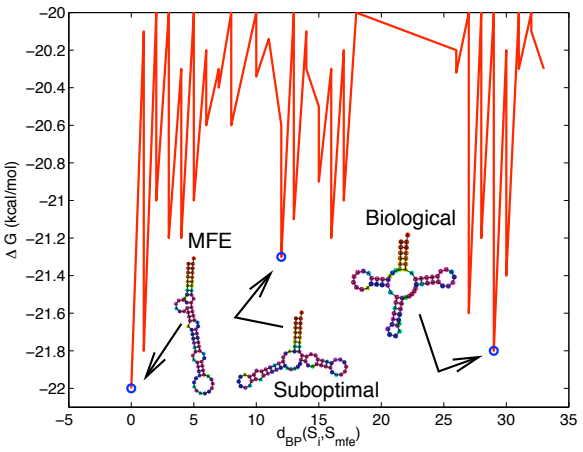
1. Plan A: Align the sequences and look for common structural elements in the alignment.
2. Plan B: Simultaneous folding and aligning
3. Plan C: Fold each sequence and align the structures.

We will discuss some aspects and give an overview of the different methods used in practice.

Given just a sequence, *de novo* foldings algorithms as, e.g., Nussinov and Zuker, compute a secondary structure.

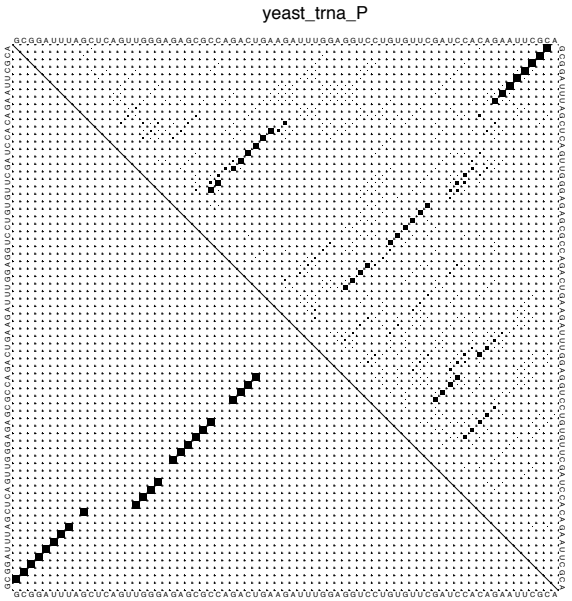
Problems with *de novo* folding:

- Often wrong (too simplistic models, too many parameters, . . .)
- *Lots* of optimal structures (and even more suboptimal ones)



[Source: lecture by Paul Gardner]

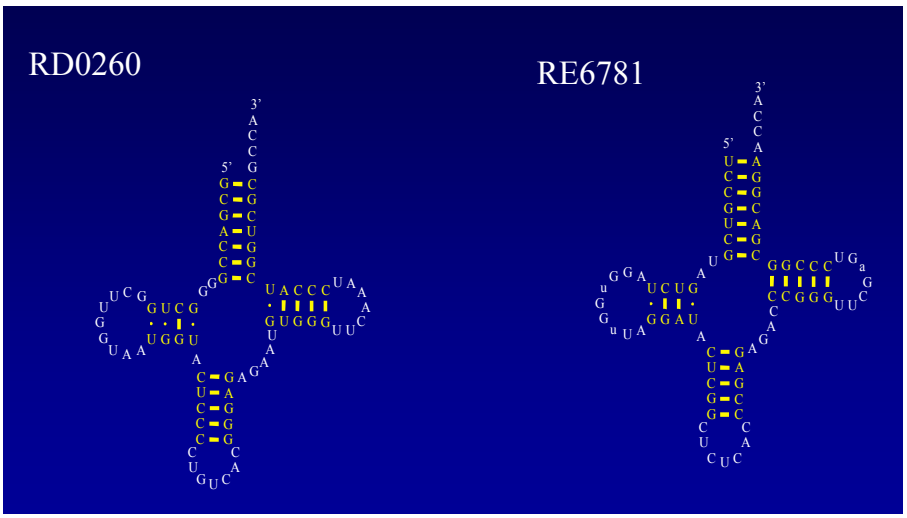
A little help: base pair probabilities (McCaskill, 1990).



But which structure is it?

More help: use biological knowledge (conservation of structure)
→ *comparative approach*

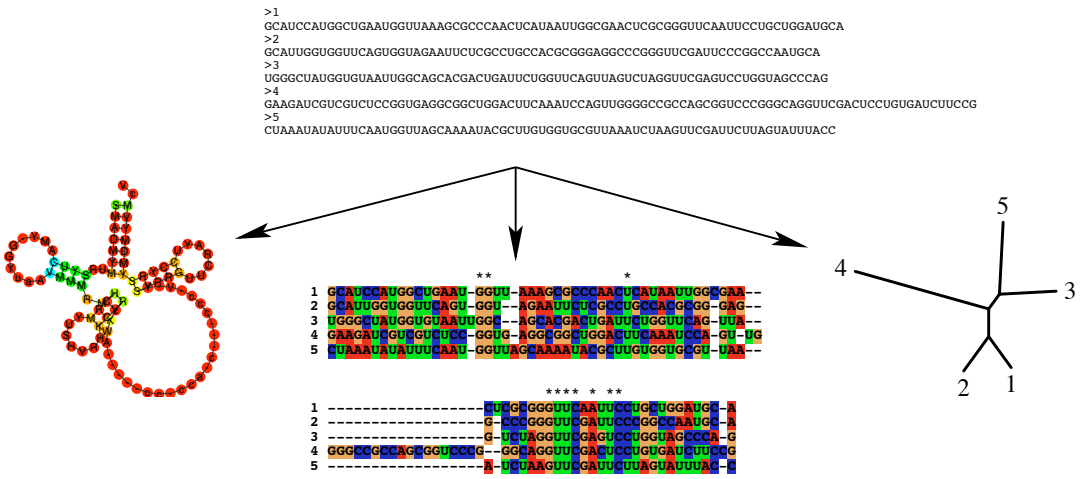
Structure is conserved although sequence is not!



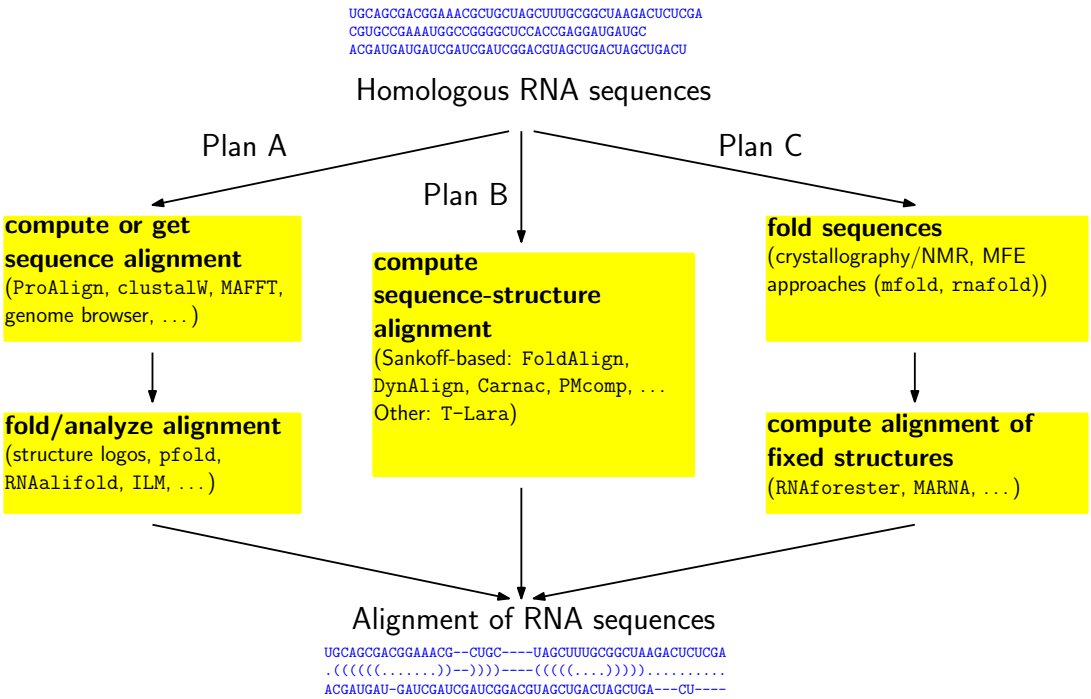
[Source: lecture by David Mathews]

13.1 Comparative RNA Analysis

- Input: set of sequences with assumed structural similarities
- Output: common structural elements, alignment, and phylogeny



lecture by Paul Gardner]



Plan A: First align, then fold

Plan B: Simultaneously align and fold

Plan C: First fold, then align.

Main problem of comparative approach:

Circular dependency of alignment and structures!

We need an accurate multiple alignment to get good structures and we need accurate structures to get a good alignment!

13 Plan A

13 Fold/Analyze Alignments

13.1 Plan A: Fold/Analyze Alignments

Derive consensus structure by

- analyzing *mutual information content*
- folding the alignment with stochastic context free grammars (SCFG)
  - extension of Hidden Markov Models
  - `pfold`, for instance, predicts most probable consensus structure of an alignment
- folding the alignment with `RNAalifold`
  - combines covariation with base-pairing energies
  - predicts consensus fold for alignment

Although energy minimization techniques are attractive, almost all trusted RNA secondary structures to date were determined using comparative analysis. However, comparative methods require many diverse sequences and highly accurate multiple alignments to work well.

The key idea is to identify the interactions (i.e., the Watson-Crick correlated positions that result from *compensatory mutations*) in a multiple alignment, e.g.:

```
seq1  GCCUUCGGGC
seq2  GACUUCGGUC
seq3  GGCUUCGGCC
```

The amount of correlation of two columns of a multiple alignment can be computed as the *mutual information content* measure:

> "if you tell me the identity of position i, how much does this reduce the uncertainty about about the identity of position j?"

13.2 Plan A: Mutual information content

A method used to locate covariant positions in a multiple sequence alignment is the mutual information content of two columns.

First, for each column i of the alignment, the frequency $f_i(x)$ of each base $x \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$ is calculated.

Second, the 16 joint frequencies $f_{ij}(x, y)$ of two nucleotides, x in column i and y in column j, are calculated.

If the base frequencies of any two columns i and j are *independent* of each other, then

$$\frac{f_{ij}(x, y)}{f_i(x) \cdot f_j(y)} \approx 1$$

If these frequencies are *correlated*, then this ratio will be greater than 1.

To calculate the *mutual information content* $H(i, j)$ in bits between the two columns i and j, the logarithm of this ratio is calculated and summed over all possible 16 base-pair combinations:

$$H_{ij} = \sum_{xy} f_{ij}(x, y) \log_2 \frac{f_{ij}(x, y)}{f_i(x) f_j(y)} \quad .$$

This measure is maximum at 2 bits, representing perfect correlation.

If either site is conserved, there is less mutual information: for example, if all bases at site i are `A`, then the mutual information is 0, even if site j is always `U`, because there is no covariance.

Examples of how to compute the mutual information content:

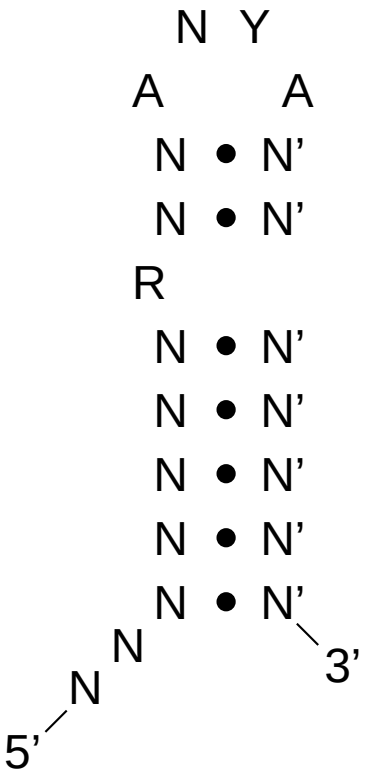
```
1  2  3  4  5  6  7  8

C  G  C  G  A  U  A  A
C  G  G  C  C  G  C  C
C  G  C  G  G  C  G  G
C  G  G  C  U  A  U  U
```

Compute:

H_{12} = _____
H_{34} = _____
H_{56} = _____
H_{78} = _____

The above example illustrates that the mutual information as given in the above formula is a general concept that not only captures the correlation between Watson crick pairs (columns 5 and 6), but also others (columns 7 and 8).

Example.



consensus binding site for R17 phage coat protein

[source: Durbin, Eddy, Krogh, Mitchison: Biological sequence analysis]

$N \in \{A, C, G, U\}$, $Y \in \{C, U\}$, $R \in \{A, G\}$

Let's first consider the purely sequence-based information content of the consensus:

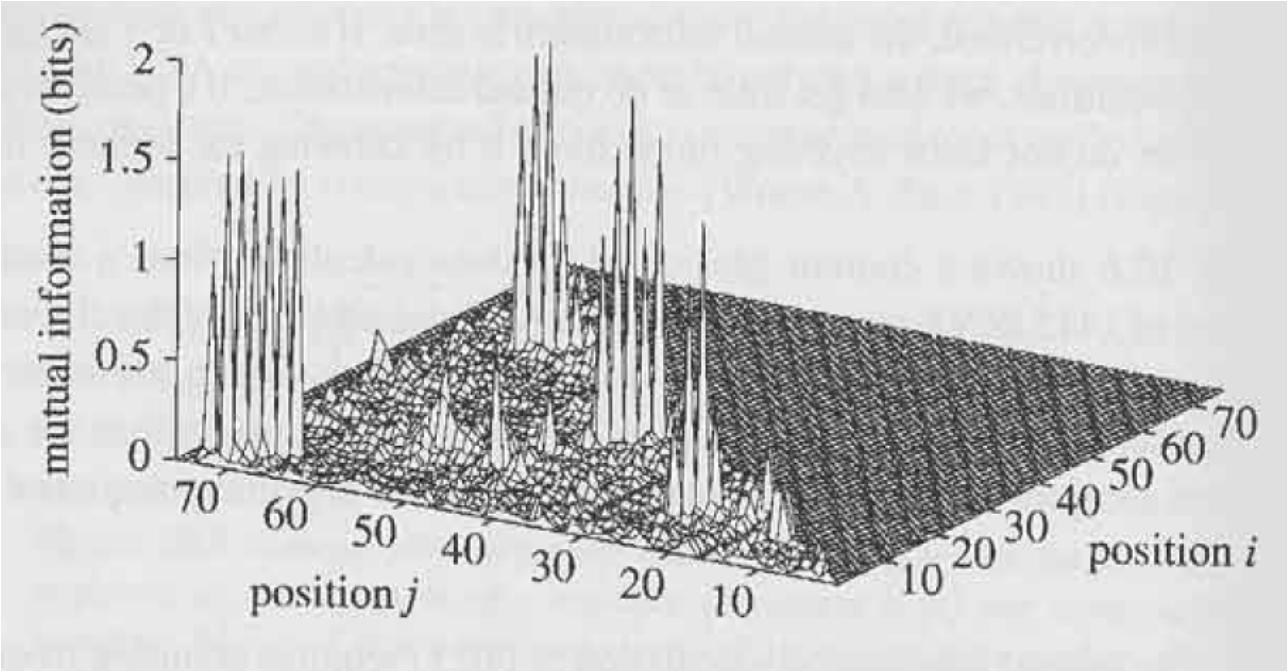
- Information conytibuted by completely conserved base (the two As) is $\sum_x p_x \log_2 \frac{p_x}{f_x} = 2$ bits assuming $f_x = 0.25$ for all $x \in N$.
- Contribution of R (or Y): $\sum_x p_x \log_2 \frac{p_x}{f_x} = \left(\frac{1}{2} \log_2 \frac{\frac{1}{2}}{\frac{1}{4}}\right) \cdot 2 = 1$ bit.
- Contribution of N: 0 bits.

Thus, the consensus 6 bits of sequence-based information. We therefore expect to find a match every $2^6 = 64$ nucleotides in random sequence, which is quite often.

Now, let's add the structural information from the base pairs:

- Contribution of $N\dot{N}'$ pair: $\sum_x \sum_y p_{xy} \log_2 \frac{p_{xy}}{f_{xy}} = 4 \cdot \frac{1}{4} \log_2 \frac{\frac{1}{4}}{\frac{1}{16}} = 2$ bits.

Including structure, the consensus conveys 20 bits of information, and we expect matches only every $2^20 \approx 10^6$ bases.



[Source: R. Durbin, S. Eddy, A. Krogh und G. Mitchison, Biological sequence analysis, Cambridge, 1998.]

13.3 Plan A: RNA analysis with stochastic context free grammars

In this lecture we will introduce *stochastic context-free grammars*, or short: *SCFGs*.

In a sense, SCFGs are for RNA sequences with their long range interactions the same as HMMs are for primary sequence analysis. Given an RNA alignment, we can design a suitable model based on covariance of the sequence positions and a SCFG to decide whether a given RNA sequence fits to the model induced by the RNA alignment.

Does it make sense to use probabilities of sentences? Noam Chomsky, the famed linguist actually thought not.

> *The notion "probability of a sentence" is an entirely useless one, under any interpretation of this term.*
>
> Noam Chomsky

However, there are other opinions, like that of Fred Jelinek (former head of the IBM speech recognition group)

> *Every time I fire a linguist, the performance of the recognizer improves.*
>
> Fred Jelinek

We start now with a little background about the language hierarchy that can be to a large extent attributed to Noam Chomsky.

13.4 Transformational grammars

Transformational grammars were introduced by Noam Chomsky. He was interested to formalize how one can enumerate all possible sentences contained in a (natural) language.

Transformational grammars are sometimes called *generative grammars*, that means they are able to generate a sentence or a string that belongs to the language they encode.

Formally a grammar consists of a set of abstract *non-terminal symbols* like $\{S\}$, a set of *rewriting rules* $\{\alpha \to \beta\}$, like ($S \to aS$, $S \to bS$, and $S \to \varepsilon$) (also called *productions*), and a set of *terminal* symbols that appear actually in a word of the language, like $\{a, b\}$.

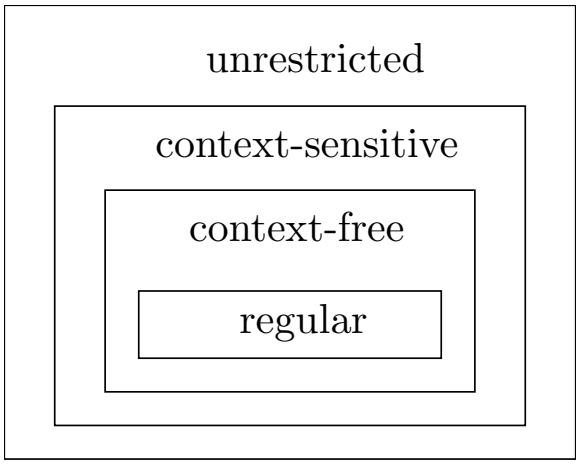
To generate a string of *a*s and *b*s we carry out a series of transformations beginning from some string. By convention we usually start from a special start symbol S. Then an applicable production is chosen which has S on its left-hand side and S is then replaced by the string on the right-hand side of the production.

The succession of strings that results from this process is called a *derivation* from the grammar. An example of the derivation in our example is $S \Rightarrow aS \Rightarrow abS \Rightarrow abbS \Rightarrow abb$.

Chomsky described four sorts of restrictions on a grammar's rewriting rules. The resulting four classes of grammars form a hierarchy known as the *Chomsky hierarchy*. In what follows we use capital letters $A, B, W, S, \ldots$ to denote nonterminal symbols, small letters $a, b, c, \ldots$ to denote terminal symbols and Greek letters $\alpha, \beta, \gamma, \ldots$ to represent a string of terminal and non-terminal letters.

1. **Regular grammars**. Only production rules of the form $W \to aW$ or $W \to a$ are allowed.
2. **Context-free grammars**. Any production of the form $W \to \alpha$ is allowed.
3. **Context-sensitive grammars**. Productions of the form $\alpha_1 W \alpha_2 \to \alpha_1 \beta \alpha_2$ are allowed.
4. **Unrestricted grammars**. Any production rule of the form $\alpha_1 W \alpha_2 \to \gamma$ is allowed.

These four classes are nested. That means a regular grammar is always also a context-free grammar and so on.



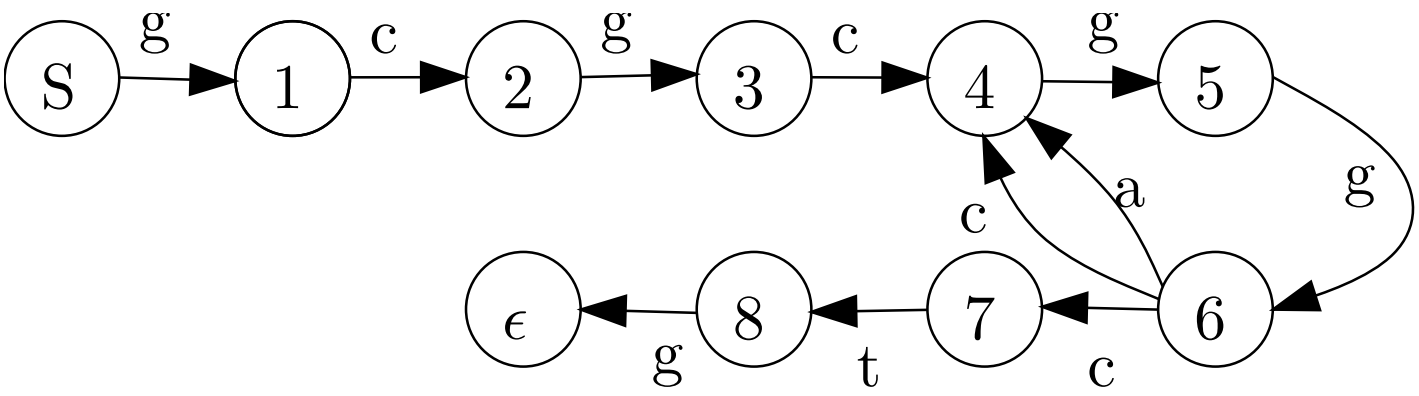
Each type of grammar has a corresponding abstract computational device associated with it, which can *parse* a given sequence and *accept* it if it belongs to the language, or *reject* it otherwise.

The corresponding machines for the Chomsky hierarchy are as follows.

1. **Regular grammars**: Finite state automaton.
2. **Context-free grammars**: Push-down automaton.
3. **Context-sensitive grammars**: Linear bounded automaton (a Turing machine with a tape of linear size).
4. **Unrestricted grammars**: Turing machine.

For example the FMR-1 gene[1] sequence contains a triplet repeat region in which the sequence *cgg* is repeated a number of times (or, as a variant *agg*). This can be modelled by a *regular grammar*, and the following finite state automaton accepts all strings starting with a *gcg*, containing a variable number of *cgg* repeats, and ending with *ctg*.

[1]The FMR-1 gene ("fragile site mental retardation 1 gene") is involved in the fragile X syndrome, the most frequent form of inherited mental retardation.



13.5 Context-free grammars

From now on we will only deal with *context-free grammars*, since with them we can conveniently model RNA sequences and their interactions. Hence it is worthwhile to write down the exact definition of a CFG.

Definition 1. A *context free grammar* G is a 4-tuple $G = (N, T, P, S)$ with N and T being alphabets with $N \cap T = \emptyset$.

- N is the nonterminal alphabet.
- T is the terminal alphabet.
- $S \in N$ is the start symbol.
- $P \subseteq N \times (N \cup T)^*$ is the finite set of all productions.

Consider the context-free grammar

$$G = \Big(\{S\},\ \{a, b\},\ \{S \to aSa \mid bSb \mid aa \mid bb\},\ S \Big).$$

This CFG produces the language of all *palindromes* of the form

$$\alpha\alpha^R \quad \text{with} \quad \alpha = \{a, b\}^* \ .$$

For example the string *aabaabaa* can be generated using the following derivation:

$$S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aabSbaa \Rightarrow aabaabaa.$$

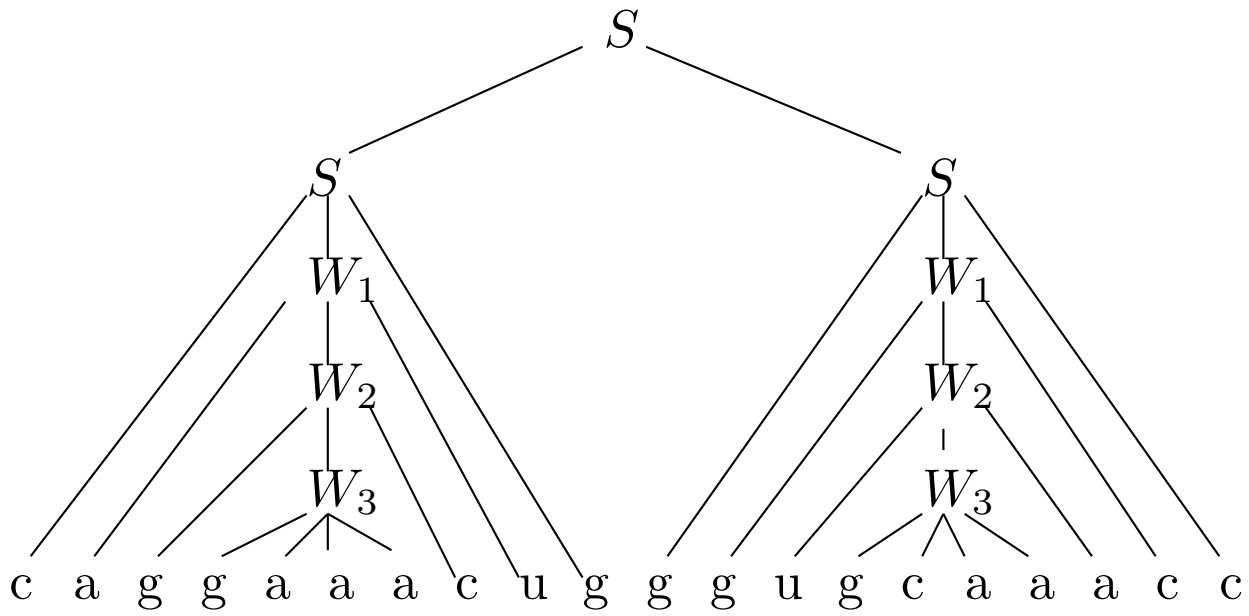
The "palindrome grammar" can be readily extended to handle RNA hairpin loops. For example, we could model hairpin loops with three base pairs and a *gcaa* or *gaaa* loop using the following productions.

$$\begin{aligned} S &\to aW_1u \mid cW_1g \mid gW_1c \mid uW_1a, \\ W_1 &\to aW_2u \mid cW_2g \mid gW_2c \mid uW_2a, \\ W_2 &\to aW_3u \mid cW_3g \mid gW_3c \mid uW_3a, \\ W_3 &\to gaaa \mid gcaa. \end{aligned}$$

(We don't mention the alphabets N and T explicitly if they are clear from the context.)

There is an elegant representation for derivations of a sequence in a CFG called the *parse tree*. The root of the tree is the nonterminal S. The leaves are the terminal symbols, and the inner nodes are nonterminals.

For example if we extend the above productions with $S \rightarrow SS$ we can get the following:



Using a pushdown automaton we can parse a sequence left to right according to the following algorithm.

The automaton's stack is initialized with the start symbol. Then the following steps are iterated until no symbols remain. If the stack is empty when no input symbols remain, then the sequence has been successfully parsed.

1. Pop a symbol off the stack.
2. If the popped symbol is a non-terminal: Peek ahead in the input and choose a valid production for the symbol. (For deterministic PDAs, there is at most one choice. For non-deterministic PDAs, all possible choices need to be evaluated individually.) If there is no valid transition, terminate and reject.

   Push the right side of the production on the stack, rightmost symbols first.
3. If the popped symbol is a terminal: Compare it to the current symbol of the input. If it matches, move the automaton to the right on the input. If not, reject and terminate.

Example. (The current symbol is written using a capital letter):

```
Input string    Stack      Operation
GccgcaaggC      S          Pop S.   Produce S->gW1c
GccgcaaggC      gW1c       Pop g.   Accept g.  Move right on input.
gCcgcaaggC      W1c        Pop W1.  Produce W1->cW2g
gCcgcaaggC      cW2gc      Pop c.   Accept c.  Move right on input.
gcCgcaaggC      W2gc       Pop W2.  Produce W2->cW3g
gcCgcaaggC      cW3ggc     Pop c.   Accept c.  Move right on input.
gccGcaaggC      W3ggc      Pop W3.  Produce W3->gcaa
gccGcaaggC      gcaaggc    Pop g.   Accept g.  Move right on input.
...             ...        (several acceptances)
gccgcaaggC      c          Pop c.   Accept c.  Move right on input.
gccgcaaggc$     -          Stack empty, input string empty.  Accept!
```

13.6 Stochastic context-free grammars

Using CFGs we not only want to define rigid patterns like in the above example, rather we want to derive a consensus pattern and

- allow approximate matches
- weight the matches somehow,

so that we can learn the weights. Similar to a Markov model we now assign probabilities to productions. This could then be used for example to compute the most probable derivation of a sequence.

Definition 2. A *stochastic context free grammar* G is a 5-tuple $G = (N, T, P, S, \rho)$ where:

- (N, T, P, S) is a CFG
- $\rho : P \to [0, 1]$
- Each production $p \in P$ has a *probability* $\rho(p)$ with

$$\sum_{A \to \alpha \in P} \rho(A \to \alpha) = 1 \qquad \forall A \in N$$

The *probability of a derivation* is the product of the probabilities of the individual productions.

Defining the SCFG would be only the first step in creating a useful probabilistic modelling system for a sequence analysis problem. Similar to HMMs we must address the following three problems:

1. Calculate an optimal alignment of a sequence to a SCFG: the *alignment* problem.
2. Calculate the probability of a sequence given a SCFG: the *scoring* problem.
3. Given a set of RNA sequences, estimate the optimal probability parameters for the SCFG: the *training* problem.

13.7 Chomsky normal form

In order to formulate algorithms for SCFGs it is very useful to choose a normalized form for the productions. One such form is the *Chomsky normal form* or CNF. This form requires that all CFG productions are of the form $W \to W_y W_z$ or $W \to a$.

Any CFG can be cast into CNF by expanding any non-conforming rules into a series of allowed productions, with the help of additional nonterminals. For example the rule $S \to aSb$ could be expanded to $S \to W_1 W_2$, $W_1 \to a$, $W_2 \to SW_3$, $W_3 \to b$.

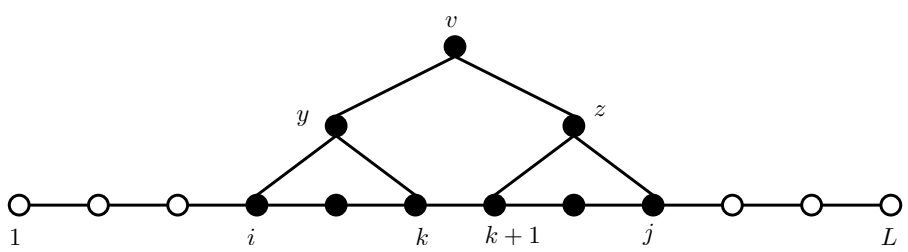
13.8 Inside and outside algorithms

Now we will formulate the *inside* and the *outside* algorithm which are the natural counterparts for SCFGs of the forward and backward algorithms for HMMs. Similarly, the *Cocke-Younger-Kasami (CYK)* algorithm is the equivalent of the Viterbi algorithm and only a variant of the inside algorithm.

The SCFG algorithms also use dynamic programming but they are computationally more expensive than the HMM counterparts. Later we will see that we can speed up the algorithms for RNA based models.

- Denote the M different nonterminals as $W_1, \ldots, W_M$, with W_1 being the start terminal. The SCFG must be in CNF with productions
  - $W_v \to W_y W_z$ (with transition probability $t_v(y, z)$) and
  - $W_v \to a$ (with emission probability $e_v(a)$).
- The sequence of terminal symbols is denoted as $x_1, \ldots, x_L$.
- The *inside* algorithm computes the array of values $\alpha(i, j, v)$, which is defined as the probability of a parse subtree rooted at nonterminal W_v for the subsequence $x_i, \ldots, x_j$.

  Hence the probability we are interested in is $P(x \mid \theta) = \alpha(1, L, 1)$.



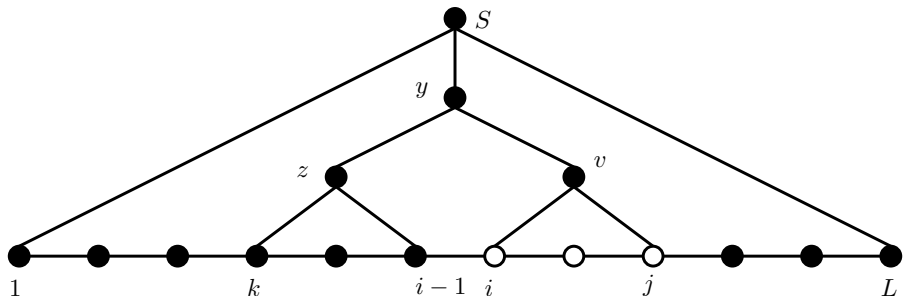
The above figure illustrates one step of the inside algorithm. The probability $\alpha(i, j, v)$ of the subtree rooted at v for subsequence $x_i \dots x_j$ is computed recursively by summing the probabilities of smaller trees, more precisely,

$$\alpha(i, j, v) = \sum_{y=1}^{M} \sum_{z=1}^{M} \sum_{k=i}^{j-1} \alpha(i, k, y)\, \alpha(k+1, j, z)\, t_v(y, z) \ .$$

Inside(G, x)**;**

(1) for $i = 1$ to L
(2) for $v = 1$ to M do $\alpha(i, i, v) = e_v(x_i)$ od;
(3) od
(4) for $\ell = 1$ to $L - 1$ do
(5) for $i = 1$ to $L - \ell$ do
(6) $j = i + \ell$;
(7) for $v = 1$ to M do
(8) $\alpha(i, j, v) = \sum_{y=1}^{M} \sum_{z=1}^{M} \sum_{k=i}^{j-1} \alpha(i, k, y)\, \alpha(k+1, j, z)\, t_v(y, z)$
(9) od
(10) od
(11) od

- The *outside* algorithm computes the array of values $\beta(i, j, v)$, which is defined as the *summed* probability of all parse trees rooted at the start nonterminal for the complete sequence $x_1, \dots, x_L$, *excluding* all parse subtrees for the subsequence $x_i, \dots, x_j$ rooted at nonterminal W_v for all i, j, and v.
- Hence $P(x \mid \theta) = \sum_{v=1}^{M} \beta(i, i, v) e_v(x_i)$ for any i.
- The computation starts from the largest excluded subsequence and recursively works its way inwards.



The above figure illustrates one step of the outside algorithm. The probability $\beta(i, j, v)$ is the summed probability of all parse trees *excluding* subtrees rooted at v that generate subsequence $x_i \dots x_j$. The top-down calculation chooses a symbol y from which v was (supposedly) produced.

The case shown here contributes a probability $\alpha(k, i-1, z)\beta(k, j, y)t_y(z, v)$ to the sum, for each $k < i$. The symmetric cases with $k > j$ (not shown) contribute $\alpha(j+1, k, z)\beta(i, k, y)t_y(v, z)$ each. And we have to sum this up over all choices of y and z.

13.9 Learning parameters

Why do we need *two* algorithms? The α and β values computed by the inside and outside algorithms (for SCFGs) can be used for parameter re-estimation using expectation maximization similar to the values computed by the forward and backward algorithms (for HMMs).

Outside(G, x, α);

(1) $\beta(1, L, 1) = 1$;
(2) $\beta(1, L, v) = 0, \quad$ for $v = 2, \ldots, M$;
(3) for $\ell = L - 2$ to 1 do
(4) for $i = 1$ to $L - \ell$ do
(5) $j = i + \ell$;
(6) for $v = 1$ to M do
(7) $$\beta(i, j, v) = \sum_{y,z} \sum_{k=1}^{i-1} \alpha(k, i-1, z)\, \beta(k, j, y)\, t_y(z, v) +$$
(8) $$\sum_{y,z} \sum_{k=j+1}^{L} \alpha(j+1, k, z)\, \beta(i, k, y)\, t_y(v, z)$$
(9) od
(10) od
(11) od

To do this we compute the expected number of times the state v is used in a derivation (of a single sequence):

$$c(v) = \frac{1}{P(x \mid \theta)} \sum_{i=1}^{L} \sum_{j=1}^{L} \alpha(i, j, v)\, \beta(i, j, v).$$

(In fact, the definitions of α and β were designed to make such an equation exist.)

We can even push this a little bit further. The expected number of times that v was occupied and then the production $W_v \to W_y W_z$ was used is:

$$c(v \to yz) = \frac{1}{P(x \mid \theta)} \sum_{i=1}^{L-1} \sum_{j=i+1}^{L} \sum_{k=i}^{j-1} \beta(i, j, v)\, \alpha(i, k, y)\, \alpha(k+1, j, z)\, t_v(y, z).$$

Then it follows that the EM re-estimation equation for the probabilities of the production $W_v \to W_y W_z$ is

$$\hat{t}_v(y, z) := \frac{c(v \to yz)}{c(v)} = \frac{\displaystyle\sum_{i=1}^{L-1} \sum_{j=i+1}^{L} \sum_{k=i}^{j-1} \beta(i, j, v)\, \alpha(i, k, y)\, \alpha(k+1, j, z)\, t_v(y, z)}{\displaystyle\sum_{i=1}^{L} \sum_{j=1}^{L} \alpha(i, j, v)\, \beta(i, j, v)}.$$

Similar equations hold for the productions $W_v \to a$. We have

$$c(v \to a) = \frac{1}{P(x \mid \theta)} \sum_{i \mid x_i = a} \beta(i, i, v)\, e_v(a)$$

and hence,

$$\hat{e}_v(a) = \frac{c(v \to a)}{c(v)} = \frac{\displaystyle\sum_{i \mid x_i = a} \beta(i, i, v)\, e_v(a)}{\displaystyle\sum_{i=1}^{L} \sum_{j=1}^{L} \alpha(i, j, v)\, \beta(i, j, v)}.$$

The EM re-estimation equations can be easily extended from a single observed sequence x to the case of multiple independent observed sequences. The expected counts are simply summed over all sequences.

13.10 The CYK algorithm

The remaining problem is to *find* an optimal parse tree for the sequence (i. e., an "alignment" against the grammar). This is solved by the Cocke-Younger-Kasami (CYK) algorithm. It is a variant of the inside algorithm in which sums are replaced by max operations.

The CYK algorithm computes a variable $\gamma(i, j, v)$ for the log value of the most probable path for the subsequence $x_i, \ldots, x_j$ starting from the nonterminal W_v. In addition we keep in $\tau(i, j, v)$ a triple of integers used for traceback. (Note that the algorithm is written in log-scale which would also be feasible for the inside and outside algorithms.)

The value of the most probable complete parse of the entire sequence is $\gamma(1, L, 1)$.

CYK(G**,** x**);**

(1) for $i = 1$ to L do
(2) for $v = 1$ to M do $\gamma(i, i, v) = \log e_v(x_i)$; $\tau(i, i, v) = (0, 0, 0)$; od
(3) od
(4) for $\ell = 1$ to $L - 1$ do
(5) for $i = 1$ to $L - \ell$ do
(6) $j = i + \ell$;
(7) for $v = 1$ to M do
(8) $\gamma(i, j, v) = \max_{y,z} \max_{k=i\ldots j-1} \gamma(i, k, y) + \gamma(k + 1, j, z) + \log t_v(y, z)$;
(9) $\tau(i, j, v) = \arg \max_{(y,z,k),k=i\ldots j-1} \gamma(i, k, y) + \gamma(k + 1, j, z) + \log t_v(y, z)$;
(10) od
(11) od
(12) od

Just as the Viterbi algorithm can be used as an approximation of the forward-backward EM training algorithm for HMMs, the CYK algorithm can be used as an approximation for the inside algorithm.

As an alternative to calculating the expected numbers of counts probabilistically using the inside-outside algorithm, one can calculate the optimal CYK alignments for the training sequences and then count the transitions and emissions that occur in those alignments.

13.11 Time and space complexity

The time complexity of SCFGs is $O(L^3M^3)$, the space complexity $O(L^2M)$. This is quite a bit, but the time complexity can be reduced for RNA SCFGs to $O(L^3M)$.

13.12 Nussinov SCFG

We come now to simple RNA SCFGs. Recall the *Nussinov* algorithm for RNA folding. We can achieve the same (if not more) by formulating it as a SCFG. Consider the following productions:

$S \to aS \mid cS \mid gS \mid uS$	i unpaired
$S \to Sa \mid Sc \mid Sg \mid Su$	j unpaired
$S \to aSu \mid cSg \mid gSc \mid uSa$	i, j pair
$S \to SS$	bifurcation
$S \to \varepsilon$	termination

Assume that the probability parameters are known.

Because the productions correspond to secondary structure elements, the maximum probability parse corresponds to the maximum probability secondary structure, if we assign base pair productions a relatively high probability.[2]

We could convert the above SCFG into CNF form and then apply the general CYK algorithm. But in this case it is not necessary and specific algorithms for specific SCFGs are typically more efficient. Hence we omit the conversion into CNF and directly write down the Nussinov-style CYK.

Let $\gamma(i, j)$ denote the log likelihood of the optimal structure given the SCFG (with estimated probability parameters) for the subsequence $x_i, \ldots, x_j$.

[2] Attention: This holds only iff each structure has an unambiguous parse tree. This is **not** the case for our Nussinov grammar. However, folding grammars, which are unambiguous in this sense, exist.

Let the probabilities of the SCFG productions be denoted by $p(aS)$, $p(aSu)$ etc.

N-CYK(G**,** x**);**

(1) for $i = 2$ to L do $\gamma(i, i-1) = -\infty$; od
(2) for $i = 1$ to L do
(3) $\quad \gamma(i,i) = \max\{\log p(x_i S) + \log p(\varepsilon), \log p(S x_i) + \log p(\varepsilon)\}$
(4) od
(5) for $\ell = 1$ to L do
(6) $\quad$ for $i = 1$ to $L - 1$ do
(7) $\quad\quad j = i + \ell$;
(8) $$\quad\quad \gamma(i,j) = \max \begin{cases} \gamma(i+1, j) + \log p(x_i S) \\ \gamma(i, j-1) + \log p(S x_j) \\ \gamma(i+1, j-1) + \log p(x_i S x_j) \\ \max\limits_{i<k<j} \gamma(i,k) + \gamma(k+1, j) + \log p(SS) \end{cases}$$
(9) $\quad$ od
(10) od

Similar to the Nussinov algorithm, we could model the *Zuker* algorithm using SCFGs and apply the inside-outside-CYK machinery with the same time complexity as the Zuker algorithm itself.

To solve our problem in Plan A (given alignment, compute consensus structure), we can build a *comparative SCFG*.

The program `pfold` [Knudsen/Hein 99], for example, parses columns from an alignment instead of symbols from a single sequence. *Covarying columns* get high emission probabilities. See also `http://www.daimi.au.dk/~compbio/rnafold/`.

13.13 Covariance models

Assume now we are given a family of related RNAs. Assume further we were able to compute a multiple RNA sequence-structure alignment (e. g. using the mutual information of columns), that exhibits a clear consensus structure like the one depicted below:

```
human  a a G A C u u c g G a U C u G G c G a c a C C C
mouse  u a C A C u u c g G a U G a C A c C a a a G U G
 worm  a g G U C u u c g G c A C g G G c A c c a U U C
  fly  c c A A C u u c g G a U U u U G c U a c c A U A
  orc  a a G C C u u c g G a G C g G G c G u a a C U C
struc. . . ( ( ( . . . . ) . ) ) . ( ( . ( . . . ) ) )
```

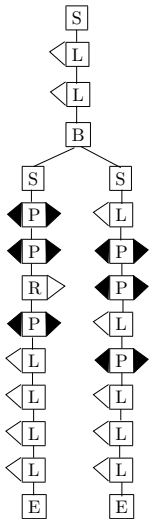
13.14 Covariance models

To describe the consensus structure with an SCFG-based model, we need several types of nonterminals to generate different types of secondary structure and sequence elements.

- for base-paired columns we need pairwise emitting nonterminals (e. g. $P \to aWb$).
- for single stranded columns we need leftwise emitting nonterminals (e. g. $L \to aW$).
- for bulges and interior loops we need rightwise emitting nonterminals (e. g. $R \to Wa$).
- we need bifurcation nonterminals (e. g. $B \to SS$).
- in addition we define a special start terminal for the immediate children and children of a bifurcation, and a special end terminal (e. g. $E \to \varepsilon$, $S \to W$).

13.15 Covariance models

The model is then best depicted by a tree. The below picture represents the RNA family in the above multiple alignment as long as no insertions and deletions occur.



We can read the productions off the tree:

$S_1 \to L_2$,
$L_2 \to (a \mid u \mid c)L_3$,
$L_3 \to (a \mid g \mid c)B_4$
$B_4 \to S_5 S_{15}$

and for the left stem:

$S_5 \to P_6$,
$P_6 \to gP_7c, \ldots,$
$P_7 \to aR_8u, \ldots,$
$\ldots,$
etc.

In the CM model we use an emit-on-state formalism (Mealy machine). Thus we have 16 pairwise emission probabilities and 4 leftwise resp. 4 rightwise emission probabilities. The transition probabilities are all 1 in the ungapped case. They become more interesting if we allow for insertions and deletions.

We discuss now how to extend the CM model to allow for insertions and deletions of arbitrary length.

13.16 Design of CM models

Gaps in CMs are modeled using a similar strategy as in profile HMMs. In profile HMMs each position in a multiple alignment corresponds to a set of three states: match, insert, delete.

Similarly, a CM expands an ungapped consensus model into a stereotyped pattern of individual states. However, symbols can be emitted to the left or to the right or to both sides.

Please note that I present here *one* possible way to construct a CM.

When pairwise nodes expand, they have several insertion and deletion possibilities. A deletion may remove both bases in the base pair or solely the 5′ or 3′ partner, leaving the remaining unpaired partner as a bulge.

Insertions in the base-paired stem may occur on the 5′ side, on the 3′ side, or both.

Hence in CMs a pairwise node expands into 6 states:

- an MP state,
- a D state (for complete deletion of a base pair),
- ML and MR states (for a single-base deletion that removes the 3′ or 5′ base, resp.), and
- IL and IR states that allow insertions on the 5′ or 3′ side of the pairs, resp.

Leftwise and rightwise nodes for single-stranded consensus positions expand to match, insert and delete states:

- ML, MR,
- IL, IR,
- D

So given an RNA consensus we could end up with the following.

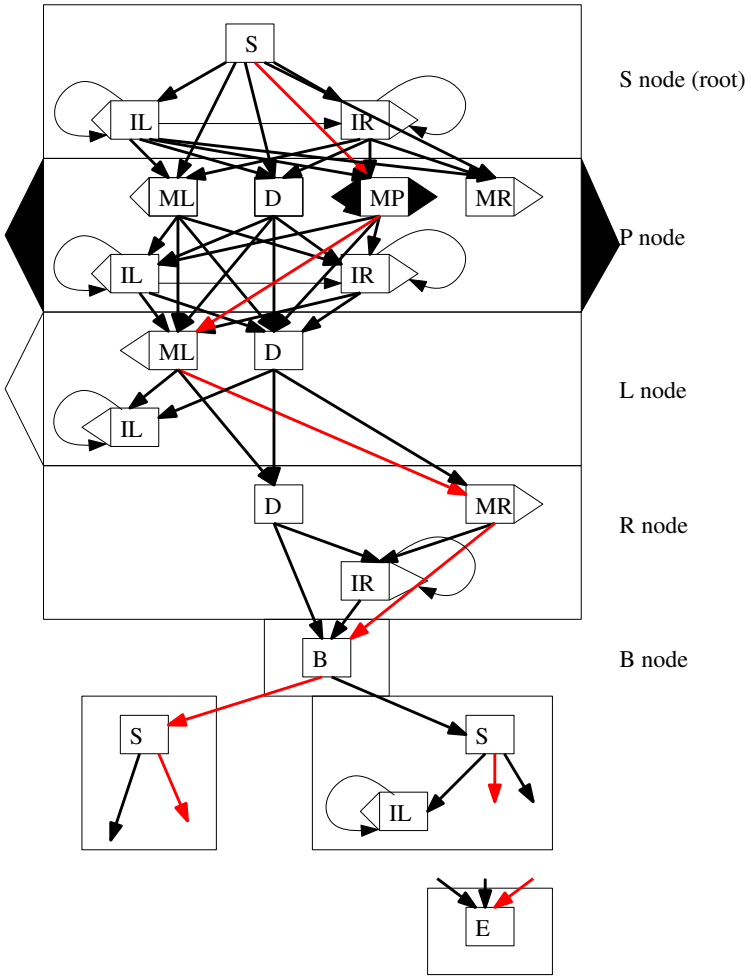
The root is expanded to a start state S and insert states for either the 5′ or 3′ side.

The left side under a bifurcation is expanded to a single S state, the right side to an S state and an insert-left IL state. This arbitrary arrangement of the insert states assures that an insertion in any position is unambiguously assigned.

Bifurcation nodes and end nodes in the consensus tree simply become B and E states in the CM.

States are connected by state transitions. As with profile HMMs, states connect all insert states in the current node to all non-insert states in the next node.

Insert states have a state transition to themselves to allow arbitrary length insertions, and IL connects to IR but not vice versa so that insertions are unambiguously assigned to a single path through the model.



Note that this is only *one* possible design of a CM model.

13.17 Constructing a CM from an RNA alignment

Assume we have an RNA alignment annotated with a consensus secondary structure. Using the structure annotation on the non-insert columns of the alignment, a consensus tree is first constructed. The nodes of this tree are "filled" with CM states, and the states are connected as discussed above.

Each sequence in the alignment can then be assigned a unique CM parse tree. Emission and transition events in these parse trees can be counted and then used for estimating emission and transition probabilities.

13.18 CM alignment algorithms

As already seen in the Nussinov-style SCFG, it would be tedious to apply the Chomsky normal form to the SCFGs for RNA analysis. We can give more specific alignment algorithms for CMs.

We denote the M different states by $W_1, \ldots, W_M$. Let v, y and z be indices for the states W_x, W_y and W_z.

There are seven types of states labelled P, L, R, D, S, B, and E, for *P*airwise emitting, *L*eftwise emitting, *R*ightwise emitting, *D*elete, *S*tart, *B*ifurcation and *E*nd states respectively. W_1 is the start state (root). The seven state types are associated with symbol emission and transition probabilities.

The Δ entries in the following table are for notational convenience and denote the number of symbols emitted on the left and right of state v.

Type	Production	Δ_v^L	Δ_v^R	Emission	Transition
P	$W_v \to x_i W_y x_j$	1	1	$e_v(x_i, x_j)$	$t_v(y)$
L	$W_v \to x_i W_y$	1	0	$e_v(x_i)$	$t_v(y)$
R	$W_v \to W_y x_j$	0	1	$e_v(x_j)$	$t_v(y)$
D	$W_v \to W_y$	0	0	1	$t_v(y)$
S	$W_v \to W_y$	0	0	1	$t_v(y)$
B	$W_v \to W_y W_z$	0	0	1	1
E	$W_v \to \varepsilon$	0	0	1	1

In addition let C_v be the children of the state v and P_v be the parents of the state. Start and delete states are treated identically in the alignment algorithms. The difference is only structurally: Start states occur as the root state or as children of bifurcations. Each bifurcation state branches into a single pair of start states. Delete states occur within R, L, and P nodes.

Apart from the fact that the bifurcation rule is the only "Chomsky" style production, there are three additional restrictions on CMs, compared to general SCFGs.

1. Each state may use *only one type of production rule*.
2. States are *not fully interconnected*.
3. States are *numbered* such that $y > v$ for all $y \in C_v$, except for insert states, where $y \geq v$ for all $y \in C_v$. This condition is important for non-emitting states, guaranteeing that there are no non-emitting cycles.

13.19 The inside algorithm

Now we turn to the scoring problem. We want to calculate the likelihood $P(x \mid \theta)$ of the sequence $x_1, \ldots, x_L$ given a covariance model θ.

Let $\alpha_v(i, j)$ be the summed probability of all parse subtrees rooted at v for the subsequence $x_i, \ldots, x_j$.

$\alpha_v(j+1, j)$ is the probability for the null subsequence of length zero; this must be included as a boundary condition since we allow non-emitting D, S, and B states.

In order to get a brief description of the recurrence, we use the notation $e_v(x_i, x_j)$ for *all* emission probabilities, even though it may not depend on x_i or on x_j. For L states we let $e_v(x_i, x_j) := e_v(x_i)$, for R states we let $e_v(x_i, x_j) := e_v(x_j)$, and for non-emitting states we (formally) let $e_v(x_i, x_j) := 1$, as in the above table.

When complete, the probability $P(x \mid \theta)$ is in $\alpha_1(1, L)$.

If there are b bifurcation states and a other states, the order of complexity of the algorithm is $O(L^2 M)$ in memory and $O(aL^2 + bL^3)$ in time.

We omit the outside algorithm and the inside-outside expectation maximization. (See Durbin book, Chapter 10.3, pp. 287). Instead we go directly to an important variant of the CYK algorithm.

13.20 The CYK algorithm for database searching

Suppose we are given a very long sequence (a complete genome, for instance) and our task is to find one or more subsequences that match the RNA model. The algorithms we have are well suited for global alignment, but still ill suited for local alignment. Most of all we do not want to spend $\Omega(L^2)$ space!

We can achieve this by limiting the length of the longest aligned subsequence to a constant D and by employing a transformation of the coordinate system. The idea is essentially the same as in the "banded" versions of other alignment algorithms, e. g., Needleman-Wunsch.

Inside-CM(CM, x**)**;

(1) for $j = 0$ to L do
(2) for $v = M$ to 1 do
(3)
$$\alpha_v(j+1, j) = \begin{cases} 1, & \text{if } s_v = \text{E}; \\ \sum_{y \in C_v} t_v(y)\alpha_y(j+1, j), & \text{if } s_v \in \{\text{S, D}\}; \\ \alpha_y(j+1, j)\,\alpha_z(j+1, j), & \text{if } s_v = \text{B}; \\ 0, & \text{if } s_v \in \{\text{P, L, R}\}\ ; \end{cases}$$
(4) od
(5) od
(6) for $\ell = 1$ to L do for $i = 1$ to $L - 1$ do $j = i + \ell$;
(7) for $v = M$ to 1 do
(8)
$$\alpha_v(i, j) = \begin{cases} 0, & \text{if } s_v = \text{E}; \\ 0, & \text{if } s_v = \text{P}, i = j; \\ \sum_{k=i-1}^{j} \alpha_y(i, k)\,\alpha_z(k+1, j), & \text{if } s_v = \text{B}; \\ e_v(x_i, x_j) \cdot \sum_{y \in C_v} t_v(y)\,\alpha_y(i + \Delta_v^L, j - \Delta_v^R), & \text{otherwise}; \end{cases}$$
(9) od
(10) od
(11) od

The dynamic programming matrix is indexed by (v, j, d) instead of (v, i, j) where $d := j - i + 1$ is the length of the subsequence $x_i, \ldots, x_j$ and $d \leq D$.

A standard CYK algorithm for SCFG alignment returns the log of the probabilities $P(S, \hat{\pi} \mid \theta)$ of the sequence S and the best parse $\hat{\pi}$ given the model θ. This is strongly a function of the length of the aligned sequences, which makes it difficult to choose the best matching subsequence among overlapping subsequences of different length.

Similar to HMMs, a nice solution to this problem is to calculate *log-odds* scores relative to a (more or less) reasonable *null model* of random sequences. For example one can assume an independent identically distributed (i.i.d.) null model in which the likelihood of a sequence is the product of individual residue frequencies f_a.

Then the log odds scores are as follows:

$$\begin{aligned} \text{for } s_v = P\text{:} \quad & \log \hat{e}_v(a, b) = \log(e_v(a, b)/(f_a f_b)) \\ \text{for } s_v = L\text{:} \quad & \log \hat{e}_v(a, b) = \log(e_v(a)/f_a) \\ \text{for } s_v = R\text{:} \quad & \log \hat{e}_v(a, b) = \log(e_v(b)/f_b) \end{aligned}$$

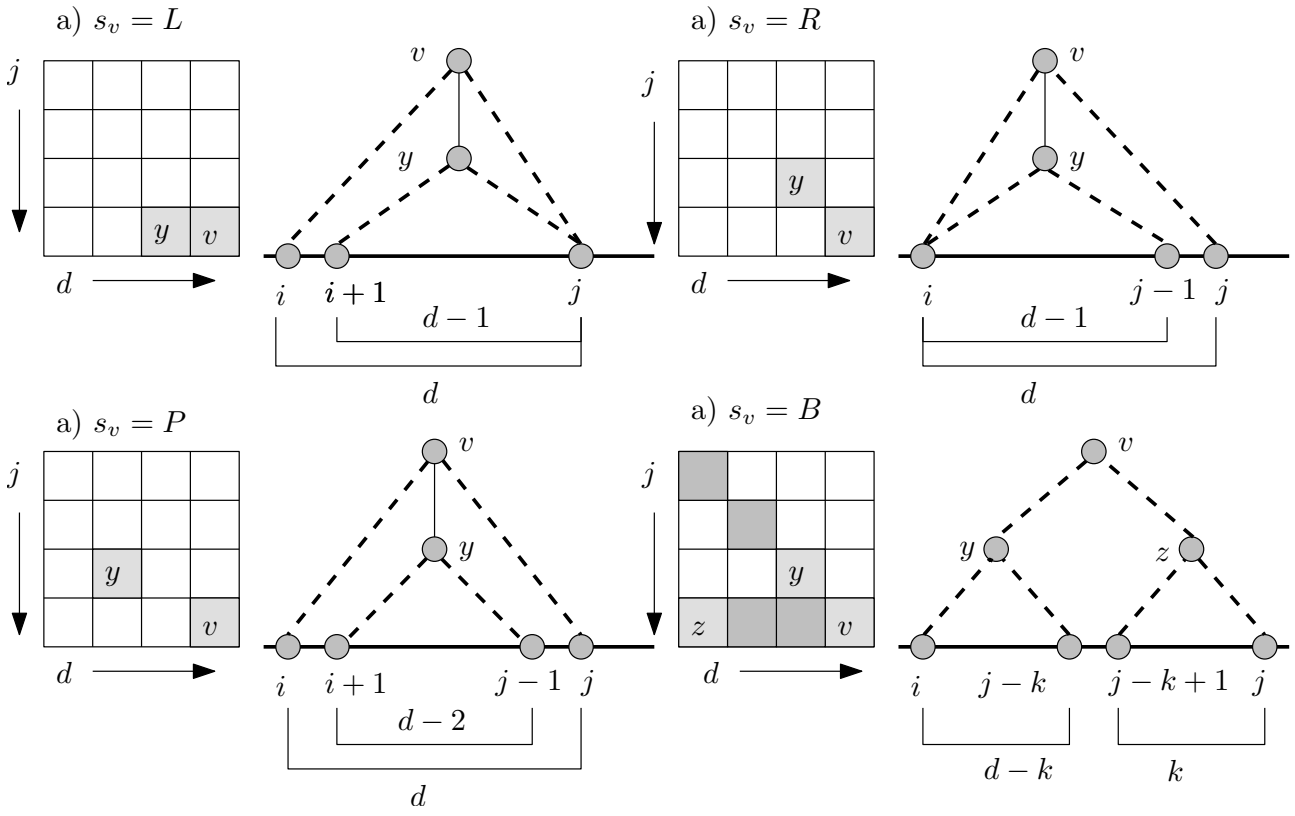
CYK-CM(CM, x**)** **Initialization:**

(1) for $j = 0$ to L do
(2) for $v = M$ to 1 do
(3)
$$\gamma_v(j, 0) = \begin{cases} 0, & \text{if } s_v = \text{E} \\ \max_{y \in C_v} \left\{ \log t_v(y) + \gamma_y(j, 0) \right\}, & \text{if } s_v \in \{\text{S, D}\} \\ \gamma_y(j, 0) + \gamma_z(j, 0), & \text{if } s_v = \text{B}, C_v = \{y, z\} \\ -\infty, & \text{otherwise} \end{cases}$$
(4) od
(5) od

The above algorithm fills a $D \times L$ matrix instead of an $L \times L$ matrix. The following picture shows the steps of the CYK recursion for the four different state types.

CYK-CM(CM, x**)** **Recursion:**

(1) for $j = 1$ to L do
(2) for $d = 1$ to $\min(D, j)$ do
(3) for $v = M$ to 1 do
(4)
$$\gamma_v(j,d) = \begin{cases} -\infty, & \text{if } s_v = \text{E} \\ -\infty, & \text{if } s_v = \text{P}, d < 2 \\ \max\limits_{0 \le k \le d} \left\{ \gamma_y(j-k, d-k) + \gamma_z(j,k) \right\}, & \text{if } s_v = \text{B}, C_v = \{y, z\} \\ \log \hat{e}_v(x_i, x_j) + \max\limits_{y \in C_v} \left\{ \log t_v(y) + \gamma_y(j - \Delta_v^R, d - \Delta_v^L - \Delta_v^R) \right\}, & \text{otherwise} \end{cases}$$
(5) od
(6) od
(7) od



13.21 Summary

- Context free grammars (CFGs) are used to model the class of context-free languages and are well-suited to describe RNA secondary structures.
- The stochastic variant (SCFGs) models the probabilities of transitions and symbol emissions.
- SCFGs can be used very similarly as HMMs. HMMs model regular languages whereas SCFGs model context-free languages.
- The forward-backward, the Viterbi, and the Baum-Welch algorithms have all corresponding algorithms for SCFGs (inside, outside, CYK, etc).
- Covariance models (CMs) are meant to model a consensus RNA structure and allow for insertions and deletions.