

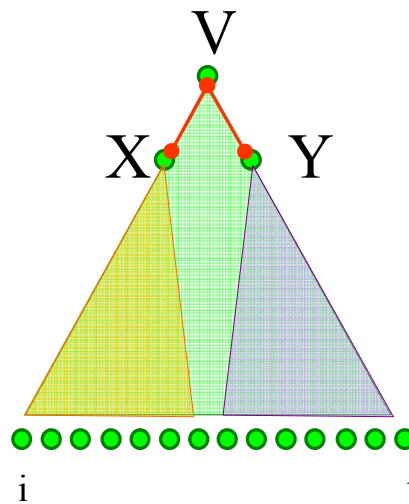
CS262 Computational Genomics
 Instructor: Serafim Batzoglou
 Scribed by: Subie Patel
 Thursday February 16, 2006

RNA Secondary Structure (cont.):

Cocke-Younger-Kasami (CYK) Algorithm:

The CYK algorithm finds the most likely parse of a given sequence if we have a Stochastic Context Free Grammar (SCFG) modeling the generation of sequences. Given a sequence $x_1 \dots x_n$ and an SCFG G , the likelihood of the most likely parse of x_i through x_j rooted at non-terminal V is denoted by the dynamic programming variable $\gamma(i, j, V)$. As our base case, we have $\gamma(i, i, V)$, which is simply the probability that non-terminal V goes to the terminal symbol x_i . Since grammars are in Chomsky Normal Form (CNF), there is only one other case for recurrence if we are not in our base case. This is the case where V goes to two non-terminals, that is, $V \rightarrow XY$. In that case, we find the most probable parse by taking the max over all possible non-terminals V could go to and all the possible ways that V can go to XY ($\max_{i \leq k < j} \gamma(i, k, X) * \gamma(k+1, j, Y)$) multiplied by the probability of the production $V \rightarrow XY$ (working in log space):

$$\gamma(i, j, V) = \max_X \max_Y \max_{i \leq k < j} \gamma(i, k, X) + \gamma(k+1, j, Y) + \log P(V \rightarrow XY)$$



We assume that recursion on gamma returns the optimal parse for the given arguments. This algorithm is analogous to the Viterbi algorithm in HMMs.

This algorithm requires the probabilities of productions which must be constructed. Consider the following grammar:

$S \rightarrow a S \mid c S \mid g S \mid u S \mid \epsilon$
 $\rightarrow S a \mid S c \mid S g \mid S u$
 $\rightarrow a S u \mid c S g \mid g S u \mid u S g \mid g S c \mid u S a$

$\rightarrow SS$
(where ϵ represents the empty string)

Notice that the series of productions – $S \rightarrow a S \rightarrow a S u$ – accomplishes the same task as the production – $S \rightarrow a S u$. We prefer the latter to the former and thus, one of the constraints on our probabilities should be that $P(S \rightarrow a S u) > P(S \rightarrow a S) * P(S \rightarrow S u)$. Another important feature of the grammar is the production – $S \rightarrow S S$. This allows the production of many distinct substructures such as two hairpins as opposed to just one.

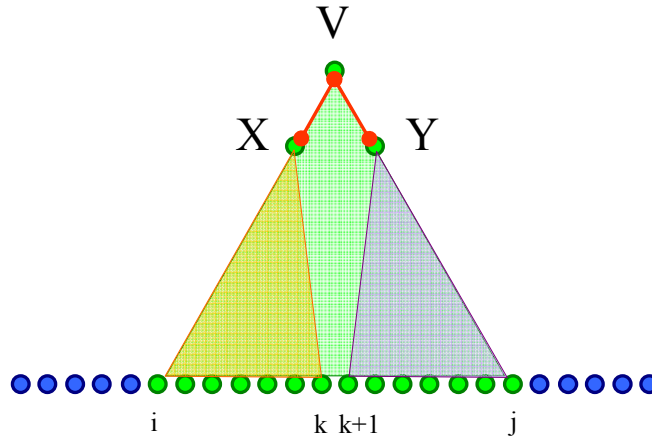
RNA structures can be evaluated by converting the grammar modeling the sequence into CNF and then directly applying the CYK algorithm, or by creating special recurrences to handle the four distinct kinds of productions in this particular grammar:

$$\gamma(i, j) = \max \begin{cases} \gamma(i+1, j-1) + \log P(x_i S x_j) \\ \gamma(i, j-1) + \log P(S x_i) \\ \gamma(i+1, j) + \log P(x_i S) \\ \max_{i < k < j} \gamma(i, k) + \gamma(k+1, j) + \log P(S S) \end{cases}$$

For the first term inside the max, we are concerned with the production – $S \rightarrow x_i S x_j$. The probability of this production at the given position in our sequence is the log probability of producing the S on the RHS of the production, which is $\gamma(i+1, j-1)$ (recursion on the interior terminals of the production), plus the log probability of the production, which is $\log P(x_i S x_j)$. Similar explanations follow for the next two cases. In the last scenario, we are concerned with the production – $S \rightarrow S S$ – for which we must take the max over all possibilities of dividing the sequence $x_i \dots x_j$ between the two non-terminal Ss on the RHS of the production while taking into account the probability of the production.

Evaluation in SCFGs:

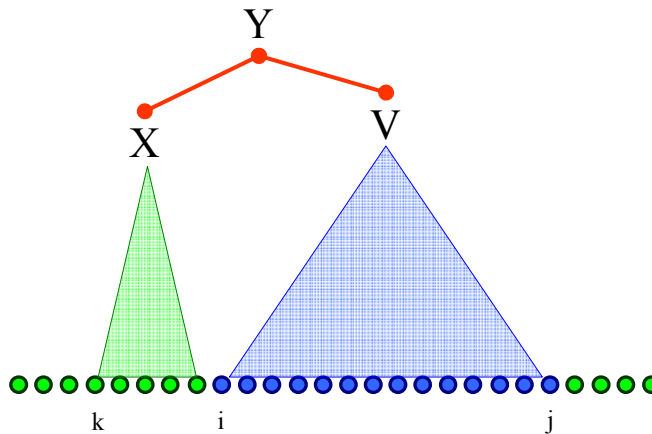
Recall our two favorite algorithms from evaluation in HMMs: forward and backward. The forward algorithm, $f_k(i)$, calculates the probability of emitting the sequence x_1 up to x_i and ending in state k while the backward algorithm, $b_k(i)$, calculates the probability of emitting the sequence x_i up to x_n having started in state k . The two analogous algorithms for SCFGs are the inside and outside algorithms. The inside algorithm, $a(i, j, V)$, calculates the probability of generating x_i through x_j rooted at a non-terminal V . The outside algorithm, $b(i, j, V)$, is the probability of generating all symbols outside x_i through x_j rooted at non-terminal V .



For the inside algorithm, we want to sum the probabilities of all the possible ways of generating the green terminals x_i through x_j , rooted at V . V must go to some non-terminals X and Y (once again, we are assuming our grammar to be in CNF). So the total probability is the sum over all X and all Y the probability of every possible partition of the terminals x_i through x_j . We also need to take into consideration the probability of the production – $V \rightarrow XY$.

$$a(i, j, v) = \sum_X \sum_Y \sum_k a(i, k, X) a(k+1, j, Y) P(V \rightarrow XY)$$

The outside algorithm is slightly trickier:



Here, we consider only the case where V is the rightmost non-terminal in the production because it's completely symmetrical to the case where V is the leftmost non-terminal. We want to calculate the probability of generating symbols x_1 through x_{i-1} and symbols x_{j+1} through x_n . This value is inside of x_k through x_{i-1} rooted at X , times the outside of x_k through x_j rooted at Y , summed over all X , Y , and k . We must also take into account the probability of the production – $Y \rightarrow XV$.

$$b(i, j, V) = \sum_X \sum_Y \sum_{k < i} a(k, i-1, X) b(k, j, Y) P(Y \rightarrow XV)$$

Adding in the case where V is on the left, we arrive at the following recurrence:

$$b(i, j, V) = \sum_X \sum_Y \sum_{k < i} a(k, i - 1, X) b(k, j, Y) P(Y \rightarrow XV) + \sum_X \sum_Y \sum_{k < i} a(j+1, k, X) b(i, k, Y) P(Y \rightarrow VX)$$

Learning for SCFGs:

Given sequences for which we have the correct parse trees, we can easily count up the frequency of every production that occurred and divide by the total number of productions to learn their probabilities. For sequences where we do not know the correct parse trees, we can apply the same technique of Expectation Maximization (EM) to SCFGs as we did in HMMs. We start at some initial values for the probabilities of productions, and then use our inside and outside algorithms to calculate the expected number of times we used each one of those rules given our training sequence. Using these values, we can re-estimate our parameters and so on until convergence.

Here is a table which highlights the similarities between algorithms for HMMs and SCFGs:

<u>GOAL</u>	<u>HMM algorithm</u>	<u>SCFG algorithm</u>
Optimal parse	Viterbi	CYK
Estimation	Forward Backward	Inside Outside
Learning	EM: Fw/Bck	EM: Ins/Outs
Memory Complexity	$O(N K)$	$O(N^2 K)$
Time Complexity	$O(N K^2)$	$O(N^3 K^3)$

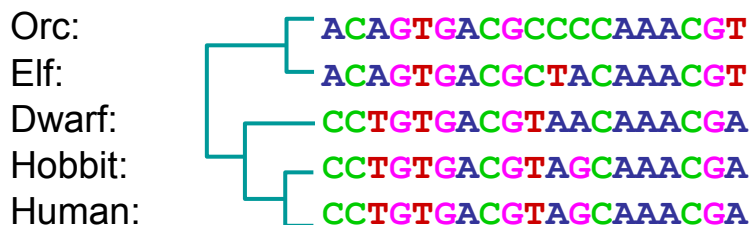
Where K: # of states in the HMM, # of nonterminals in the SCFG

Zuker Algorithm:

The Zuker algorithm is currently the best algorithm for parsing RNA structures. It is based on elaborate physical models of all possible substructures within the RNA structure. The algorithm attempts to make predictions on the energies of particular secondary structures which have been calculated through various physical experiments. Instead of only considering single pairs of letters, this algorithm models the energies of base pairs. Other secondary structures that are modeled include interactions between stems and loops, as well as loop size and composition. Lastly, bulges, places where the double helices are interrupted, are also elaborately modeled. This algorithm will not only tell you the most likely structure, but also the folding energy, that is, at what temperature the secondary structure would break down.

Phylogenetic Trees:

Phylogenetic trees are graphical models of species that are believed to have common ancestors. These trees are defined by the topology that represents which species spawned from which ancestors, as well as branch lengths representing the evolutionary distance between species and their ancestors. To infer the tree structure, we can look at the morphology of different species which is typically a poor indicator, or we can rely upon multiple alignment of genomic sequences. Looking at conserved regions of the genome across various species, we can infer tree structure by splitting the sequences into groups with common mutations.



In this example, the first split can be made by simply looking at first three letters of the sequence. Then, a split can be made on the 12th letter of the sequence to split dwarf, hobbit and human.

Modeling Evolution:

In order to quantitatively model evolution, we need to make one reasonable assumption: given a small enough period of time, only one base pair can be substituted in any a sequence. Now it is possible to estimate $P(x | y, \Delta t)$, for $x, y \in \{A, C, G, T\}$, i.e., the probability that x has mutated from y given time Δt . $P(y | y, \Delta t)$ would be the probability that y does not mutate over a given time Δt . We can define 4x4 matrix (20x20 if we are concerned with amino acids) as follows:

$$S(\Delta t) = \begin{pmatrix} P(A|A, \Delta t) & \dots & P(A|T, \Delta t) \\ \dots & \dots & \dots \\ P(T|A, \Delta t) & \dots & P(T|T, \Delta t) \end{pmatrix}$$

The diagonal of this matrix corresponds to the probabilities that there is no mutation of a given letter.

These mutations can be thought of as a Markov Process over continuous time. Let $S(t)$ be the probability of a mutation at time t . For Markov Processes, $S(t+t') = S(t)S(t')$, that is, the probability of a mutation at any time step ($t+t'$) can be decomposed into the product of a mutation occurring at time step (t) and time step (t').

The Jukes Cantor model assumes that the probability of mutation is uniformly distributed across all possible mutations. This model is popular and useful for maximum likelihood estimates of branch lengths and tree structure. In the Jukes Cantor matrix, alpha represents the probability of substituting letter, and epsilon represents a short span of time. The main idea is that if epsilon is small enough, the model can be approximated with a linear function.

$$\text{For short time } \varepsilon, S(\varepsilon) = I + R\varepsilon = \begin{pmatrix} 1 - 3\alpha\varepsilon & \alpha\varepsilon & \alpha\varepsilon & \alpha\varepsilon \\ \alpha\varepsilon & 1 - 3\alpha\varepsilon & \alpha\varepsilon & \alpha\varepsilon \\ \alpha\varepsilon & \alpha\varepsilon & 1 - 3\alpha\varepsilon & \alpha\varepsilon \\ \alpha\varepsilon & \alpha\varepsilon & \alpha\varepsilon & 1 - 3\alpha\varepsilon \end{pmatrix}$$

So $\alpha\varepsilon$ represents the probability of any mutation. To arrive at the values on the diagonal, $1 - 3\alpha\varepsilon$, we simply use the fact that the rows and columns must sum to one because they are the sum over all disjoint cases (recall the previous matrix). To model longer periods of time, we replace the values across the diagonal with a function $r(t)$ while leaving the off diagonal entries the same, $s(t)$.

$$S(t) = \begin{pmatrix} r(t) & s(t) & s(t) & s(t) \\ s(t) & r(t) & s(t) & s(t) \\ s(t) & s(t) & r(t) & s(t) \\ s(t) & s(t) & s(t) & r(t) \end{pmatrix}$$

Given our assumption that $S(t+t') = S(t)S(t')$ and the way we constructed our matrices, it can be derived that $r(t) = \frac{1}{4}(1 + 3e^{-4\alpha t})$ and $s(t) = \frac{1}{4}(1 - e^{-4\alpha t})$. So at $t = 0$, $r(t) = 1$ and $s(t) = 0$. As t approaches infinity, $r(t)$ and $s(t)$ approach $\frac{1}{4}$, a uniform distribution across mutations and conservations.

Constructing Phylogenetic Trees:

Two of the basic principles of phylogeny estimation are as follows:

- 1) The degree of sequence difference is proportional to length of independent sequence evolution
- 2) Only sequences where there is a high certainty of alignment should be used. In other words, a very well conserved gene should be chosen.

Once we have a region whose alignment we trust, we must decide upon the correct tree to relate the various species. The number of possible trees rises exponentially with the number of species so it is not possible to evaluate all trees as it is an NP-Hard problem. An easier task is to estimate branch length (the number of mutations or the length of time) given some species and a tree.

We can define the distance between two sequences a number of ways. The most obvious way is just to count up the number of letters that differ between the two sequences and divide by the total number of letters. Another way to estimate distance is to use the Jukes

Cantor model to get a measure of time between two species. We can write our frequency of mutations, $f = 3 s(t)$ because there are 3 ways a letter can mutate:

$$\begin{aligned} f &= 3 s(t) = \frac{3}{4} (1 - e^{-4\alpha t}) \Rightarrow \\ \frac{3}{4} e^{-4\alpha t} &= \frac{3}{4} - f \Rightarrow \log(e^{-4\alpha t}) = \log(1 - \frac{4}{3} f) \\ &\Rightarrow -4\alpha t = \log(1 - \frac{4}{3} f) \end{aligned}$$

Solving for t:

$$d_{ij} = t = -\frac{1}{4} \alpha^{-1} \log(1 - \frac{4}{3} f)$$

So if we want to use Jukes Cantor, given two sequences and a multiple alignment between them, we need to know the frequency of mutation, f , and also the speed at which the region of the sequence evolves, α . Given these two values, we can arrive at an estimate for d_{ij} , the time distance between two species.

Using Distances for Clustering (Average Linkage Method):

Now that we have our distances (branch lengths) between each species we wish to include in our tree, we need to decide on the topology of the tree. We can decide where to split through an iterative clustering algorithm where each cluster begins as single species and the clusters which are closest are merged until all species are in the same cluster. We define the distance between two clusters as follows:

$$d_{ij} = \frac{1}{|C_i| \times |C_j|} \sum \{p \in C_i, q \in C_j\} d_{pq}$$

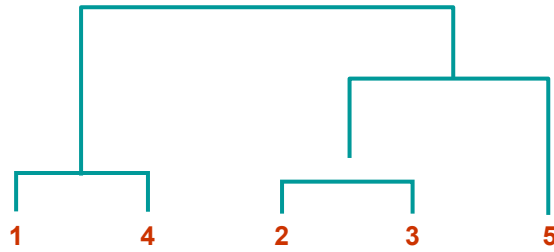
This is simply the sum of pair wise distances between all members of each cluster, divided by the number of pairs, $|C_i| \times |C_j|$. If $C_k = C_i \cup C_j$, then to save computation when clusters are merged, we can compute the distance from C_i to C_k as:

$$d_{ki} = \frac{d_{il} |C_i| + d_{jl} |C_j|}{|C_i| + |C_j|}$$

When two clusters are merged, we create a new node in our tree representing the ancestor of the two groups, placing the new node at height $d_{ij}/2$ (we're assuming both species evolve at the same rate).

Ultrametric Distances and Molecular Clock:

The previous method for creating trees is strongly related to a mathematical distance function. A distance function $d(.,.)$ is defined to be ultrametric if for any three distances $d_{ij} \leq d_{ik} \leq d_{kj}$, it is true that $d_{ij} \leq d_{ik} = d_{kj}$. In words, this means that given three objects i , j , and k , and the distances between them, d_{ij} , d_{ik} , d_{kj} (all possible pairs), two of those distances are equal, and one is less than or equal to the other two. Looking at a sample tree, these relationships make more sense:



Picking any three leaves of this tree, two of those are going to be closest to each other (twice the height to their closest common ancestor). Since those two are closer to each other than either of them is to the third, the common ancestor between the two closest and the third must be higher up in the tree. Because we defined the height of a common ancestor to be $d_{ij}/2$ when merging clusters C_i and C_j , the two closest nodes will have exactly the same distance to the third node. Thus, our ultrametric constraint is met.

The molecular clock model simply states that all species start at height zero of the tree and that evolution occurs at the same rate in all species. Since the evolutionary rate is the same, we divide the distance between two clusters equally amongst the two branches which merge those clusters.

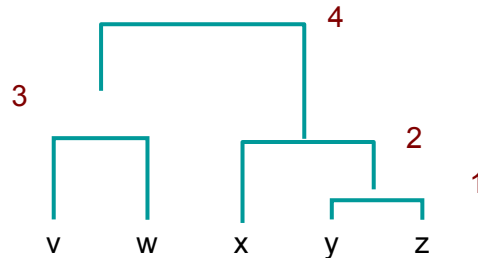
AL Example:

1	v	w	x	y	z
v	0	6	8	8	8
w		0	8	8	8
x			0	4	4
y				0	2
z					0

3	v	w	xyz
v	0	6	8
w		0	8
xyz			0

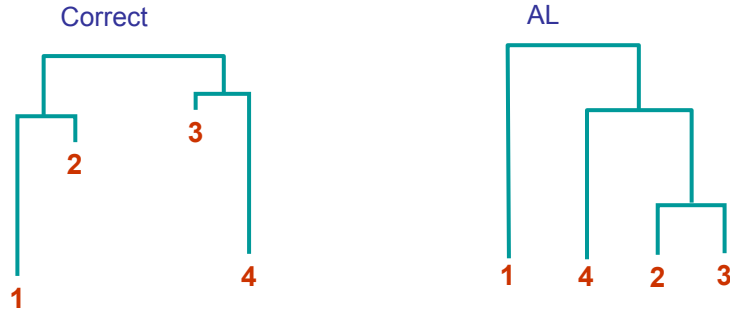
4	vw	xyz
vw	0	8
xyz		0

2	v	w	x	yz
v	0	6	8	8
w		0	8	8
x			0	4
yz				0



Problem with Average Linkage Method:

Average linkage is a poor model because of the molecular clock assumption. Different species simply do not evolve at the same rates due to generation time as well as other factors. Suppose we have the following correct phylogenetic tree to the left:



AL will assume the same evolution rates across species and as a result infer the incorrect tree.

Additive Distances:

Another less restrictive way to construct the tree is to constrain our distances to be additive, not ultrametric. Distances are additive if the tree can be shown geometrically; so if b along the path from a and c in our tree, then $d_{ac} = d_{ab} + d_{bc}$. Given a tree and additive distances, it is possible to uniquely reconstruct all the lengths of the edges using the formula $d_{km} = \frac{1}{2} (d_{im} + d_{jm} - d_{ij})$.

AD Example:

D_1

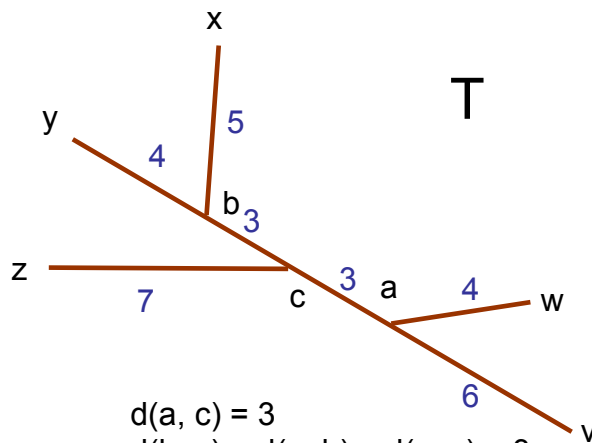
	a	x	y	z
a	0	11	10	10
x		0	9	15
y			0	14
z				0

D_2

	a	b	z
a	0	6	10
b		0	10
z			0

D_3

	a	c
a	0	3
c		0



$d(a, c) = 3$
 $d(b, c) = d(a, b) - d(a, c) = 3$
 $d(c, z) = d(a, z) - d(a, c) = 7$
 $d(b, x) = d(a, x) - d(a, b) = 5$
 $d(b, y) = d(a, y) - d(a, b) = 4$
 $d(a, w) = d(z, w) - d(a, z) = 4$
 $d(a, v) = d(z, v) - d(a, z) = 6$
Correct!!!

Neighbor-Joining:

Using additive distances and given a tree structure T , we can come up with all the branch lengths. However, it is usually the case that we do not know the structure but we do know the pair wise distances between organisms. Now we will briefly look at an algorithm which can infer topology. This algorithm is guaranteed to produce the correct tree when distances are additive and it also does a decent job when distances aren't additive.

Neighbor-Joining correctly reconstructs – in cubic time – all branch lengths given only pair wise distances between organisms. The algorithm is as follows:

Define:

$$D_{ij} = d_{ij} - (r_i + r_j)$$

Where:

$$r_i = \frac{1}{|L| - 2} \sum_k d_{ik}$$

Claim: The above “magic trick” ensures that D_{ij} is minimal **iff** i, j are neighbors

Initialization:

Define T to be the set of leaf nodes, one per sequence
Let $L = T$

Iteration:

Pick i, j s.t. D_{ij} is minimal
Define a new node k , and set $d_{km} = \frac{1}{2} (d_{im} + d_{jm} - d_{ij})$ for all $m \in L$

Add k to T , with edges of lengths $d_{ik} = \frac{1}{2} (d_{ij} + r_i - r_j)$
Remove i, j from L ;
Add k to L

Termination:

When L consists of two nodes, i, j , and the edge between them of length d_{ij}