

MS2a, Week 2, Model Solution

Rune Lyngsø

October 28, 2011

A Counting of trees

- a. No root inner nodes have 3 edges, only leaves are labelled. How many distinct trees with 10 leaves?

Use formula

$$T_{\text{unrooted}}(n) = \prod_{i=3}^n (2i - 5)$$

with $n = 10$. This yields 2,027,025 distinct unrooted trees with all internal nodes of valence 3.

- b. No root and only leaves are labelled. How many distinct trees with 8 leaves.

Without a bound on internal node valence we need to proceed by the recursion

$$R(n, k) = \begin{cases} 1 & \text{if } k = 1 \\ T_{\text{unrooted}}(n) & \text{if } k = n - 2 \\ k \cdot R(n - 1, k) + (n + k - 3) \cdot R(n - 1, k - 1) & \text{otherwise} \end{cases}$$

(n is number of leaves, k number of internal nodes, and a new leaf is attached either to an existing internal node or to an edge, thereby creating an extra internal node) and sum $R(n, k)$ for all $k = \{1, \dots, n - 2\}$, in this case for $n = 8$. This yields 39,208 distinct trees.

- c. No root inner nodes have 3 edges all nodes are labelled. How many distinct trees with 4 leaves?

We know there are $T_{\text{unrooted}}(n)$ trees when only leaves are labelled. When the $n - 2$ internal nodes are also labelled, we can for each of these trees construct new trees by

- first choose a subset of size n of the $2n - 2$ labels to label the leaves

- choose any permutation of the remaining $n - 2$ labels to label the internal nodes

Hence, the total number of trees is

$$\binom{2n - 2}{n} \cdot (n - 2)! \cdot T_{\text{unrooted}}(n)$$

which for $n = 4$ yields 90 distinct trees.

Show 2 trees that would be identical if inner nodes were unlabelled.

Assume the set of labels is $\{1, \dots, 6\}$. With only 4 leaves the topology is fixed. Leaf labelling is also fixed to be the same for the two trees, by the requirement for identity. So the only remaining freedom is to swap the labels of the two internal nodes:



- d. No root, inner nodes have 3 edges – no nodes are labelled. How many trees with 6 leaves?

With 6 leaves there will be 4 internal nodes. Once we have the topology of how the internal nodes are connected, we will have to add leaves to each internal node to make it have exactly three incident edges. Furthermore, we should only consider topologies of internal node connectivity where each node has valence at most three, also known as trivalent trees. In the lecture material, there are only two topologies on four nodes, both trivalent, so there are 2 distinct trees.

- e. No root – no nodes are labelled. How many trees with 6 leaves?

We already found the two trees where all internal nodes have valence three. If internal nodes are only restricted to have valence at least three, we also need to consider the topologies for less than four internal nodes, and the distinct ways we can attach leaves to these topologies. For each possible choice of less than four internal nodes, there is only one possible topology. With three internal nodes we can attach the one surplus leaf to either the central internal node, or one of the flanking nodes (by symmetry, it doesn't matter which one we choose), for two more distinct trees. With two internal nodes we can either have a balanced attachment of three leaves to each, or split the leaves 4-2, for a further two distinct trees. Finally there is just one possibility with

only one internal node. In total, we get 7 distinct trees. We could also just look it up as sequence A007827 on <http://www.research.att.com/~njas/sequences/>.

B Ancestral nucleotides

Let n be a node in a binary (internal nodes have two children) tree with a root, let n_L be the left child, n_R the right child. Let $d(,)$ be a distance function on nucleotides, with the distance between identical nucleotides being 0, the distance between nucleotides separated by a transition being 2, and the distance between nucleotides separated by a transversion being 5. $w(n, N)$ is the total cost of the evolution in the subtree hanging from n if the nucleotide N must be at node n .

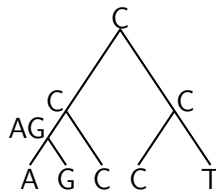
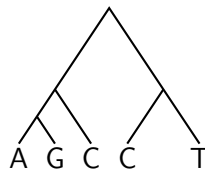
Basic recursion:

Initial condition $w(\text{leaf}, N) = 0$ if N is actually at this leaf, infinity if not.

$$w(n, N) = \min\{w(n_L, N_L) + d(N, N_L)\} + \min\{w(n_R, N_R) + d(N, N_R)\}$$

First min is taken over N_L element in $\{A, C, G, T\}$, the second min is taken over N_R element in $\{A, C, G, T\}$.

- f. Find the cheapest assignment of nucleotides to internal nodes in the tree below. What is the evolutionary cost of the tree with this assignment? Are the nucleotides assigned unambiguously?



Any of the indicated assignments will incur a cost of two transitions (one between purines and one between pyrimidines) and one transversion (from cytosine to a purine), for a total cost of 9. As can be seen, in one of the tree nodes the assignment cannot be done unambiguously.

- g. Why does this recursion work?

Let's proceed with an induction argument. For a leaf n , if the nucleotide observed at the leaf is identical to N in $w(n, N)$, then no

evolution is required to effect the observation; if the nucleotide observed is not identical to N , then N cannot be observed at n and no finite amount of evolution would suffice.

Now assume that we are at non-leaf node n and the recursion has worked for the subtrees of n , *i.e.* the subtrees rooted at n_L and n_R . There must have been a nucleotide at n_L , and we will denote this by N_L . The minimum amount of evolution on the branch from n to n_L is $d(N, N_L)$ (assuming $d(\cdot, \cdot)$ obeys the triangle inequality), and by induction hypothesis the minimum amount of evolution in the subtree rooted at n_L has been correctly computed for $w(n_L, N_L)$. The minimum total amount of evolution in this part of the tree is thus $d(n_L, N_L) + w(n_L, N_L)$. Similarly, the minimum total amount of evolution in the other part of the tree is $d(n_R, N_R) + w(n_R, N_R)$, where N_R is the nucleotide at n_R . We observe that the two contributions to the amount of evolution in the tree rooted at n are independent, so we can choose nucleotides N_L and N_R leading to the smallest minimum amount of evolution at respectively n_L and n_R independently of each other.

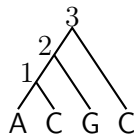
- h. Can you come up with a simple example where the method would fail if we used $w(n)$ instead of $w(n, N)$ (And also ignored N_L and N_R)?

We cannot completely ignore N_L and N_R – if we just compiled minimum scores $w(n)$ without remembering anything about possible nucleotides at n we could always postulate a match to achieve a final minimum amount of evolution of 0. So assume that apart from $w(n)$ we also keep track of the set $W(n)$ of nucleotides at n that would allow the minimum score to be $w(n)$. So $w(\text{leaf}) = 0$ and the set of minimum scoring nucleotides at a leaf would just be the observed nucleotide. The recursion would be modified to

$$w(n) = \min_N \left\{ \min_{N_L \in W(n_L)} \{d(N, N_L) + w(n_L)\} + \min_{N_R \in W(n_R)} \{d(N, N_R) + w(n_R)\} \right\}$$

with $W(n)$ being the set of all N attaining this score.

Consider the following situation:



After initialisation we would first get $w(1) = 5$ and $W(1) = \{A, C\}$. At 2 we would observe that choosing a purine would result in a score

of 7 ($w(1)$ and an additional transition) while choosing C would result in a score of 10 ($w(1)$ and a transversion on the branch to the G). So we would get $w(2) = 7$ and $W(2) = \{A, G\}$. Finally at 3 we could choose either A, G or C for a total score of 12 ($w(2)$ and an additional transversion). But choosing C at all internal nodes just requires two transversions (on the branches to the A and the G) for a total score of 10. Evidently the method using $w(n)$ fails in this case.

- i. Could this algorithm be modified so it could handle ambiguity in sequencing (say we only knew that the nucleotide at the first leaf was purine)?

If we only knew that the nucleotide at the first leaf was a purine, both A and G would be possible nucleotides that would effect this observation, so in the initialisation we should assign scores of $w(\text{first leaf}, N) = 0$ for both these cases. More generally, the only thing that would need to be changed for the recursion to properly deal with sequencing and other ambiguities would be to initialise $w(\text{leaf}, N) = 0$ for all nucleotides consistent with the observed state at each leaf.

- j. How would the recursion look if we were analyzing proteins?

Much the same, except that there would be twenty instead of four choices for N (and N_L and N_R), and we would probably call them A , A_L and A_R instead. But choices in sibling subtrees would still be independent and we would just have to minimise over all possibilities for each choice.

- k. Could you make an algorithm that would minimize the number of amino acid changes if we had codons at the leaves?

Yes. If all we care about is amino acid changes, we could just translate each codon to its corresponding amino acid and proceed as above (where the distance function $d(\cdot, \cdot)$ reflects the minimum number of amino acids that would have to be visited when transforming any codon for one amino acid into any codon for the other amino acid by changing only one nucleotide at a time).

It is probably a bit more realistic to assume that we would want to minimise some score that was a sum of a nucleotide evolution score and an amino acid change score. Extending the set of possibilities to the 64 possible codons we could use the same recursion, with the distance function $d(\cdot, \cdot)$ capturing the dual cost of changes along each branch.

- l. Given an alignment of 10 sequences, 100 nucleotides long how could the most parsimonious phylogeny be found?

One possible approach is to run through all possible phylogenies for the 10 sequences and for each use the recursion described here to compute the parsimony cost for each phylogeny. The phylogeny with the lowest parsimony cost is the most parsimonious phylogeny.

How much computation would be involved?

We would need to run through the $T_{\text{rooted}}(n) = \prod_{i=3}^n (2n-3)$ rooted phylogenies (if the distance function is symmetric, as in this case, it will suffice with just the $T_{\text{unrooted}}(n)$ unrooted phylogenies). For each of the leaves we need to set four values of $w(\cdot, \cdot)$ and for each of the internal nodes we need to minimise over four choices twice for each possible nucleotide. This has to be done for each column in the alignment. In total, we would need to perform

$$T_{\text{unrooted}}(10) \cdot 100 \cdot (10 \cdot 4 + 8 \cdot 4 \cdot 4 \cdot 2) = 59,999,940,000$$

relatively simple computational steps.

Would it be slower if we had had proteins?

If we denote the number of sequences by n , the number of columns in the alignment by m , and the number of alphabet letters by s (hence $s = 4$ for the nucleotide case and $s = 20$ for the protein case), then the above expression symbolises into

$$T_{\text{unrooted}}(n) \cdot m(ns + 2(n-2)s^2) = T_{\text{unrooted}}(n) \cdot m(n + 2(n-2)s)s$$

We immediately see that the computational burden increases with increasing alphabet size, in this particular case somewhere between 5 and 25 fold (a slightly more detailed analysis yields an increase by a factor converging to $55/3$ for $n \rightarrow \infty$). One could argue that a nucleotide sequence of m positions translates into a protein sequence of only $m/3$ positions, so if this is taken into account the increase in computational burden is only by a factor between $5/3$ and $25/3$. If we wanted to take changes at both the nucleotide level and the protein level into account, *i.e.* working with codons, the alphabet size increases by a factor of 16 while the sequence length only decreases by a factor of 3, resulting in an increase in computational burden by a factor between $16/3$ and $256/3$ (converging to $176/3$ for $n \rightarrow \infty$).