COSC 348:
Computing for Bioinformatics

Lecture 12:
Phylogenetic tree inference: calculation of cost

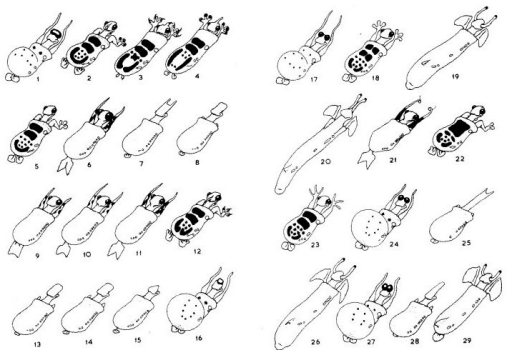
Lubica Benuskova

Prepared according to the notes of Dr. Richard O'Keefe

http://www.cs.otago.ac.nz/cosc348/

Phylogeny based on cost: parsimony approach

- Let's have an imaginary species called *Caminalcules*. We want to derive a phylogenetic tree for its subspecies (labelled by numbers).



Definition: character

- Let us construct phylogeny based on characters.
- A **character** is a measurable property of a taxonomic unit.
- Characters are chosen because we believe they are informative. We can be wrong about that.
- A character **value** or character **state** is a value that the character may have.
- Based on the values, a character may be
  - Boolean or binary
  - Discrete
  - Continuous

Characters: example

- Head_junction: simple, complex
- Horn: absent, present, horn flattened, horn pointed
- Head_length: 9--10.9, 10.9--12.8, 12.8--14.7, 20.4--22.3, etc.
- Anterior_of_head: concave, flat, convex
- Anterior_projections: absent, present
- Eyes: absent, present
- Eye_stalks: absent present
- Length_of_stalks: 3--4.5, 4.5--5.9, 6--7.5, 10.5--12, etc.
- Top_of_head: depressed, flat, crested, headcrest, single, lobate
- etc. More characters will be used in the lab.

Discrete characters

- *Binary*: there are just two possible values, which we may represent by 0 and 1. The only question you can ask is "are these the same or different?"
- *Nominal*: there are two or more possible values, which we may represent by integers: 0 ... v-1. There is no ordering; we could mix up the numbers any way we like and it would make no difference. The only question we can ask is "are these values the same or different?" There is no notion of some values being closer than other values.
  - We might choose to treat nucleotides (A, C, G, T) as nominal values. However, when we do that, we loose sight of the fact that they fall into two groups: the pyrimidines (C, T) and the purines (A, G).
  - In the same way, we could treat amino acids as nominal values, but we'd lose some similarity detail.

Discrete characters contd.

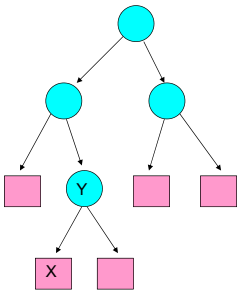
- *Ordinal*: there are several possible values which are in a definite order. We can represent the values by 0 ... v-1, or by any other series of numbers we like, but we cannot scramble the order because ***the order matters***. If a character is nominal, a value x may be closer to y than to z, but we cannot say ***how much*** closer.
  - For instance, we have a distinct number of possible colours, where these values have an ordered relationship, e.g., light brown < medium brown < dark brown < black.
- *Counts*: the values are the whole number counts. For example, we might take the number of toes on the forelimb: 0 for a snake, 1 for a horse, 2 for a cow, 3 for a three-toed sloth, 4 for a frog, 5 for a mouse. The values are fixed. If a character is a count, we can make quite *precise quantitative comparisons*.

Continuous characters

- Physical measurements such as length, weight, and so on must always be strictly positive. They are totally ordered.
  - If *x* lies between *y* and *z*, we cannot determine whether *x* is closer to *y* or *z* without deciding whether to use raw numbers (difference, ratio, etc), square roots, or logarithms, or some other transformation.
  - Differences between measurements can be zero.
- Continuous characters are often "binned". That is, the range is divided into blocks and the block number recorded as an *ordinal value*. For example, in the *Caminalcule* characters we find:
  - `13. If eyes on stalks, length of stalk (excluding eye) in mm, recorded as (0) 3-4.5mm; (1) 4.5-5.9mm; (2) 6-7.5mm; (3) 10.5-12mm; (4) 13.5-15mm; (5) 16.5-18mm.`
- This converts a continuous measurement to an ordinal value.

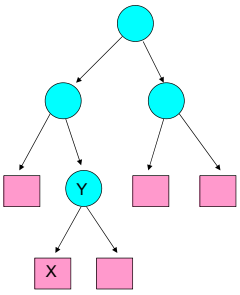
Calculating the cost

- Let us ask the question "how hard is it to get from state X to state Y?".
- We will compute the ***cost of change*** for a single character in a given tree, working from the leaves along the branches towards the root.
- This can easily be applied to any number of characters, either
  - by making a separate pass over the tree for each character and adding up the results,
  - or by making a single pass over the tree adding up for all char's as we go.



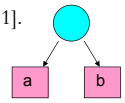
The Fitch algorithm for cost calculation

- The cost of a node is the *smallest number of changes* for the subtree rooted at that node.
- This algorithm is suitable for binary and nominal characters, because it only asks "are these two values the same or different"? Any difference counts as a cost of 1.
- Author: Walter M. Fitch, Prof. of molecular evolution, Irvine, USA.



The Fitch algorithm

- Lets' assume we have only one binary character [0, 1].

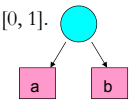


- The Fitch of a leaf with value *x* is (0, *x*)
- The Fitch of an internal node with children *a, b* is
  - let (*cost_a*, *value_a*) be the Fitch of child *a*
  - let (*cost_b*, *value_b*) be the Fitch of child *b*
    - if *value_a* intersect *value_b* is non-empty, return (*cost_a* + *cost_b*, *value_a* **intersect** *value_b*)
    - if *value_a* intersect *value_b* is empty, return (*cost_a* + *cost_b* + 1, *value_a* union *value_b*)

Note on intersection and union: let one set be {2,3,5} and another set be {1,2,4}, then the intersection would be {2} and the union would be {1,2,3,4,6}.

The Fitch function: example

- Lets' assume we have only one binary character [0, 1].

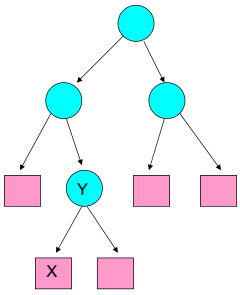


1. The Fitch of a leaf *a* is (0, [0 or 1]).
2. The Fitch of a leaf *b* is (0, [0 or 1]).
3. The Fitch of an internal ancestor node with children *a, b* is

- ➢ If *value_a* = *value_b* = 0, the Fitch of their ancestor = (0, 0).
- ➢ If *value_a* = *value_b* = 1, the Fitch of their ancestor = (0, 1).
- ➢ If *value_a* = *0, value_b* = 1, the Fitch of the ancestor = (1, [0 or 1]).
- ➢ If *value_a* = *1, value_b* = 0, the Fitch of the ancestor = (1, [0 or 1]).

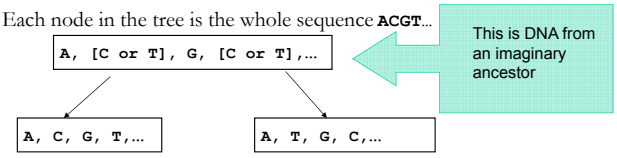
The Fitch cost of a tree

- Computes the cost and value for each node starting from the bottom of the tree for each branch.
- The final cost of the whole tree is the **cost component at the root**.
- Generalisation to nominal values,
  - for instance Anterior of head: concave (0), flat (1), convex (2).
  - then possible unions of values for the ancestor are: flat or convex (1 or 2), concave or flat (0 or 1), or concave or convex (0 or 2).



Fitch cost for biomolecules

- It can easily be applied to any number of characters. Possible values for characters in DNA are {A, T, C, G}, for proteins there are 20 different AA, plus the gap.
- We have a set of *n* globally aligned sequences and the task is to apply the Fitch algorithm. If the letter is the same, cost = 0, if the letter is different, cost = 1. We treat each letter as one character.
- Each node in the tree is the whole sequence **ACGT**...



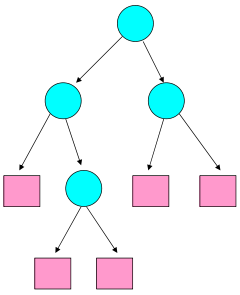
- We'll deal with the question how to order the nodes later.

The Fitch algorithm: evaluation

- In reporting the run-time of these algorithms, I use the following variables:
  - *n* = the number of OTUs
  - *k* = the number of characters measured
  - *v* = the number of values for a character
- The cost of the node cost calculation is *O(v)*. We visit each node exactly once, so the cost for a single character is *O(nv)*. Since we have to do this for each character and combine the results, the total cost is *O(nkv)*.
- In short, the Fitch algorithm is simple and fast, but crude.

The Sankoff algorithm for cost calculation

- The cost is based on an evolutionary change scoring scheme (magnitude of change matters).
- We proceed from the leaves to the root. The final cost is the *minimal* cost of the root.
- This algorithm is suitable for any characters.
- David Sankoff, professor at Montréal University, Canada



The Sankoff evolutionary cost

- Let C_{ij} be the evolutionary cost of going from state *i* in an ancestor to state *j* in an immediate descendant.
- Let's take a tree-structured character as an example:
  - **(Horn is absent (0)**
  - **, (Horn is flattened (1)**
  - **, Horn is pointed (2)**
  - **)**
  - **)**

	j=0	j=1	j=2
i =0	0	3	3
i =1	3	0	1
i =2	3	1	0

- Common sense: it is easier to change the shape of a horn than to acquire or lose one. So we might have a change cost matrix like this:
- If our characters are letters in biosequences, we might use the same cost matrices that we use in doing alignments (PAM, BLOSUM, etc).

The Sankoff algorithm for cost calculation

- Let w_i is the least cost of the subtree rooted at the node if this node had the value *i*.
- Sankoff of a leaf with value *x* is $w_x = 0$ (or infinity).
- Sankoff of an internal node with children *a, b* is
  - let (w_a) be Sankoff of child *a*
  - let (w_b) be Sankoff of child *b*
  - Return: $w_i = \min_j (C_{ij} + w_{a_j}) + \min_k (C_{ik} + w_{b_k})$

Calculated recursively for ALL concrete values *i, j* and *k*, respectively

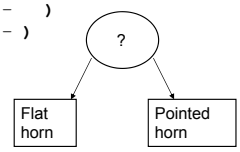
Cost of changing value *j* to value *i*

Cost of changing value *k* to value *i*

The Sankoff cost: example

- Let's take a tree-structured character as an example:
  - **(Horn is absent (0)**
  - **, (Horn is flattened (1)**
  - **, Horn is pointed (2)**
  - **)**
  - **)**

	j=0	j=1	j=2
i =0	0	3	3
i =1	3	0	1
i =2	3	1	0



$$w_i = \min_j (C_{ij} + w_{a_j}) + \min_k (C_{ik} + w_{b_k})$$

$$\left.\begin{aligned} w_0 &= (C_{01} + w_{a_1}) + (C_{02} + w_{b_2}) = (3+0)+(3+0) = 6 \\ w_1 &= (C_{11} + w_{a_1}) + (C_{12} + w_{b_2}) = (0+0)+(1+0) = 1 \\ w_2 &= (C_{21} + w_{a_1}) + (C_{22} + w_{b_2}) = (1+0)+(0+0) = 1 \end{aligned}\right\}$$

Minimum cost = 1

Not observed / not observable values

- The problem that plagues us in this data set is the "not observed/not observable" the "x" values in our *Caminalcules* example.
- One way to handle them is to eliminate them. You can remove all characters for the Caminalcules that have only "x" values for all subspecies. However, there doesn't seem to be any way to get rid of all of them.
- Another way is to treat changes involving an "x" as costing nothing, unless there's an implied change between two non-"x" values.
- **By far the simplest way is this one: just treat "x" as another value. Treat change to or from "x" just like any other change. Such a change is generally caused by a change in the character, so we can't be too far out.**

Multiple characters

- Compute tree cost for each character independently, and add them up.
- We do not have to assume that the characters are equally important: we could assign a possibly different weight to each character and calculate a weighted sum.
- There is a *big approximation* here, which is that the characters are independent, which may not be the case.
  - We can solve this by merging the characters: e.g. physical measurements are often not independent: body size and body mass are related.

Evolutionary cost for biomolecules

- If the phylogenetic tree contained DNA sequences in each node then we can treat each letter {A, T, C, G} as a one character.
- For DNA we have the following *substitution matrix*: '+2' as a reward for match, '+1' for substitution of pyrimidine for pyrimidine and purine for purine, respectively, and '–1' as the penalty for mismatch (+some gap penalty):
- Thus for each node we can calculate the alignment score and by taking its inverse, we get the value of the evolutionary cost C_{ij} of going from DNA sequence i to DNA sequence j
  - (the more similar the sequences are the smaller evolutionary change there was).

Evolutionary cost = inverse of the alignment score

- For proteins we'd use protein substitution matrices (PAM, BLOSUM)
- The scoring scheme consists of character *substitution scores* (i.e. score for each possible character replacement) plus penalties for gaps (constant, linear, additive).
- The *alignment score* is the sum of substitution scores and gap penalties. The alignment score thus evaluates **evolutionary similarity of sequences**.
- **The cost of evolutionary change is the inverse of the alignment score.**

The Sankoff algorithm: inifinity

- Interestingly, the Sankoff algorithm can handle the cases where we are not exactly sure what the character value for some leaf is, only that it belongs to some subset.
- We assign the cost 0 to each possible value and infinity to the impossible values.
- "Infinity" does not have to be an IEEE infinity value and it does not need any special magic to handle it. If n is the number of OTUs, then the tree can be at most n levels deep, and the biggest possible increment is twice the maximum entry in the C matrix, so $2n\max(C_{max})+1$ will suffice as "infinity", and this is finite.

The Sankoff algorithm: evaluation

- In reporting the run-time, I use the following variables:
  - n = the number of OTUs
  - k = the number of characters measured
  - v = the number of values for a character
- The algorithmic cost here is higher. It is $O(n.v^2)$ for a single character, so $O(n.k.v^2)$ for all the k characters. In practice this algorithm is **$O(v^2)$ times slower** than the Fitch algorithm. We put up with this because it can give us better results.
- The Sankoff algorithm for cost calculation is smarter, but you have to tell it more, and it costs more than the Fitch algorithm.