COSC 348:
Computing for Bioinformatics

Lecture 13:
Building a (phylogenetic) tree

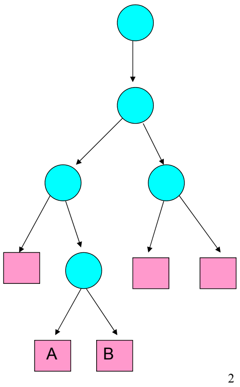
Lubica Benuskova

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

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

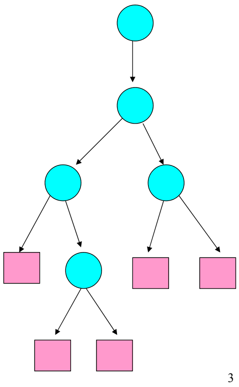
Phylogeny

- Phylogenetic tree is a tree with two kinds of nodes:
  - A **leaf** has a parent and no children. It stands for an observed taxonomic unit (OTU).
  - An **internal node** has a parent and *two* children. It stands for a hypothetical taxonomic unit (HTU), i.e. an imaginary ancestor.
  - The two children are not ordered: (A,B) and (B,A) are the same split.



Building a tree

- In this lecture we'll look at how many trees there are for given *n*, how to build a tree from scratch and start looking at how to find the tree with the minimal cost .
- The tree construction problem is not only of interest in bioinformatics.
- Suppose, for example, that you want to optimise database queries. There are as many ways to join (or unite, or intersect) *n* tables as there are phylogenies for *n* species.



Exhaustive enumeration

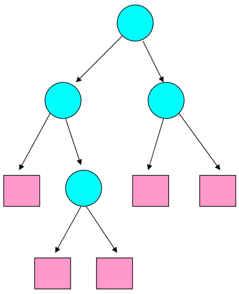
- Let's look at how we can infer a phylogenetic tree using the parsimony idea. Having defined the cost *f*(*x*) of a tree *x*, we now have to find the tree or trees with the least cost *v*.
- The simplest possible algorithm is *exhaustive enumeration:*

```
To find tree x in set X for which cost f(x) is least:
  1. best_x = last_tree, best_f_x := cost_of_last_tree
  2. for each x in X
          a. cost := f(x)
          b. if cost < best_f_x then
            I. best_x = x, best_f_x := cost
           II. Else go to step 1
```

- This is a very practical algorithm, as long as *X* (the set of all phylogenetic trees) is small and as long as we are able to construct them all.

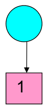
Counting the trees: rooted trees

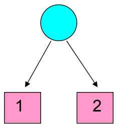
- Thus, we are interested in how many trees there are for given *n*.
- A rooted phylogeny means that we not only know how much change there has been, but *also* which way it has gone.
- A phylogenetic tree with *n* leaves will have *n*–1 internal nodes and 2*n*–2 edges.
  - Example: let *n* = 5, then we have 4 internal nodes and 8 edges.
  - An exception is *n* = 1.



Counting trees: towards Newick format

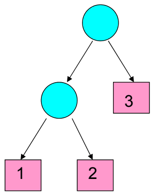
- Let's start by making a tree containing leaf 1. There is only one such tree, and it has one internal node and one edge.
- Let us have two leafs 1 and 2, then we have one internal node and since two children are not ordered, we have only one tree: (1, 2).
- Here '(' stands for the left edge, ')' stands for the right edge, and ',' stands for an imaginary ancestor (internal node).



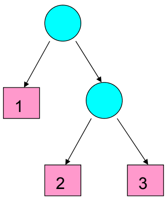


Counting trees: Newick format

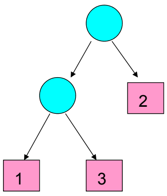
- Let's have 3 leaves: 1, 2 and 3. We have three different trees:
  - Convention: child, which has the smallest left subtree goes first and order of children does not matter.



((1, 2), 3)



(1, (2, 3))



((1, 3), 2)

Newick format: unique representation of a tree

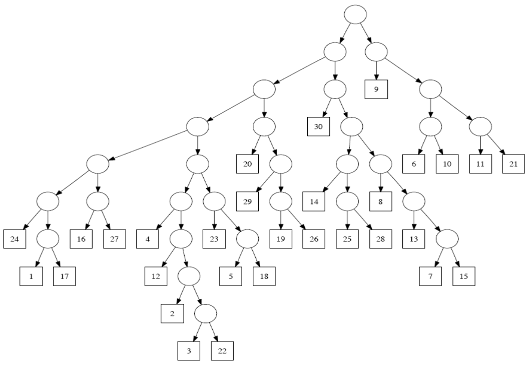
1. One rooted tree with 1 leaf: 1
2. One rooted tree with 2 leaves: (1,2)
3. Three rooted trees with 3 leaves: ((1,2),3) ((1,3),2) (1,(2,3))
4. Fifteen rooted trees with 4 leaves:

(((1,2),3),4)
(((1,2),4),3)
(((1,3),2),4)
(((1,3),4),2)
(((1,4),2),3)
(((1,4),3),2)
((1,(2,3)),4)
((1,(2,4)),3)
((1,(3,4)),2)
((1,2),(3,4))
((1,3),(2,4))
((1,4),(2,3))
(1,((2,3),4))
(1,((2,4),3))
(1,(2,(3,4)))

Note: The order of the children of a node is not significant, so there may be many strings that represent the same tree. One way to pick a unique string is to order the children A B of a node so that the child, which has the smallest left subtree should go first. Having unique representations makes it easier to see if we all find the *same* tree.

Tree of *Caminalcules* in Newick format

- (((((((1,17),24),(16,27)),((((2,(3,22)),12),4),((5,18),23))),(((19,26),29),20)),(((((7,15),13),8),(14,(25,28))),30)), (((6,10),(11,21)),9))



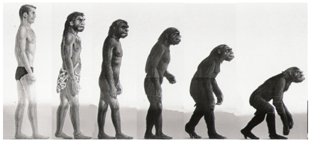
Number of trees grows exponentially

- Here is the table how fast the # of trees grows with *n*
- It can be shown the number of trees with *n* leaves is the recursive product of $1 \times 3 \times 5 \times ... \times (2n-3)$.
- This calculation holds for rooted trees.
- A rooted phylogeny means that we not only know how much change there has been, but which way it has gone (from a common ancestor).

n	# of trees
1	1
2	1
3	3
4	15
5	105
6	945
7	10,395
8	135,135
.	.
.	.
.	.
20	8200794532637891559375

Unrooted trees

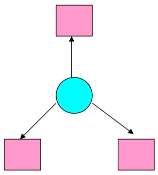
- In the Agatha Christie detective story "The Man in the Brown Suit", the heroine's father was an eccentric anthropologist whose theory was that humans were not descended from chimpanzees, no, chimpanzees were degenerate humans!



- If the only OTUs you have are chimpanzees and humans (i.e no other data like fossils), that's not as crazy as it sounds. All you have is a difference, not a direction.
- Many phylogeny inference algorithms construct *unrooted* trees.

Unrooted trees

- Thus, in the case of unrooted phylogenies, there are two kinds of nodes: leaves are on the "outside" of the tree and internal nodes are on the "inside" of the tree.
- Given an unrooted tree, we can construct a rooted tree by choosing one of its leaves, ripping the leaf off, and making that edge the root edge.
- Conversely, given a rooted tree, we can make an unrooted tree by pasting a new leaf onto the root edge.
- So the number of unrooted trees with *n* leaves is the same as the number of rooted trees with *n*–1 leaves.

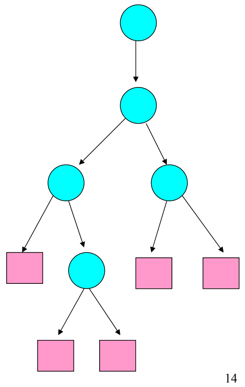


Rooted tree: an outgroup

- As computer scientists, we prefer to work with rooted trees. The simplest way to ensure that we can do this is to use an *outgroup*.
- An outgroup is an OTU, which is similar enough to the OTUs we are really interested in, to make comparisons possible, but different enough that it's obviously outside the family.
- Finding an outgroup requires biological knowledge and judgement.
  - For example, if we wanted to find a phylogeny for the apes (people, chimps, bonobos, gorillas, orang-utans) we might choose Rhesus monkeys.
- Computationally, we add the outgroup to the tree last, forcing it to join up at the root.

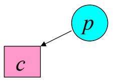
Building a tree

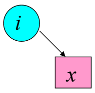
- It is useful to have a dummy "header" node at the very top of the tree, which counts as the parent of the root node – each node now has a parent.
- When we insert the OTU, we do so by splitting one of the existing edges and inserting an internal node with the new leaf as one of its children and the old subtree as its other child.
- Splitting an edge and splitting just above a subtree amount to the same thing; there are 2*n*-1 subtrees.

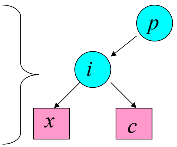


Algorithm for building a tree

1. Pick an existing node *c* (for "child").
2. Let node *p* be the parent of *c*.
3. Let *i* be a new internal node.
4. Let *x* be a new leaf (eXternal node) holding the OTU that is to be added to the tree.
5. Make *i* be the *x*'s parent and *x* be *i*'s right child.
6. Make *i* be the *c*'s parent and *c* be *i*'s left child.
7. Make *i* be *p*'s left child if *c* was *p*'s left child or *p*'s right child if *c* was *p*'s right child.
8. Make *p* be *i*'s parent.
9. Swap *x* for *c*.

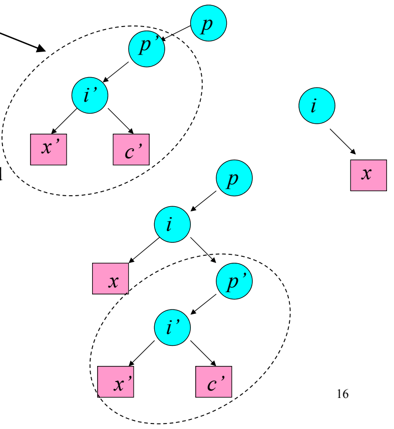






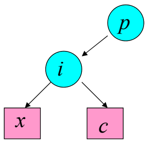
Algorithm for building a tree

1. Pick an existing **subtree** *c* (for "child").
2. Let node *p* be the parent of *c*.
3. Let *i* be a new internal node.
4. Let *x* be a new leaf holding the OTU that is to be added to the tree.
5. Make *i* be *x*'s parent and *x* be *i*'s right child.
6. Make *i* be *c*'s parent and *x* be *i*'s left child.
7. Make *i* be *p*'s left child if *c* was *p*'s left child or *p*'s right child if *c* was *p*'s right child.
8. Make *p* be *i*'s parent.
9. Swap *x* for *c*.

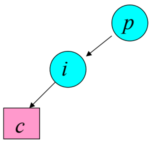


Undoing changes

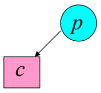
- To undo the most recent addition, you start from *x*.



- From that, *p* is *i*'s parent, and *c* is *i*'s left child.



- As you are about to discard *i*, you only have to worry about restoring the link between *p* and *c*.



How to build the best tree?

- We know how to calculate cost.
- We know how to build a tree.
- We know the problem is NP complete. There is no efficient way to locate a solution.
- How do we know at least in principle, in which order to add OTUs so that the tree cost would be minimal?
- To find the "best" tree we have to use one of the optimisation algorithms: greedy algorithm with random restarts, hill-climbing, simulated annealing, genetic algorithm.