COSC 348:
Computing for Bioinformatics

Lecture 14:
Phylogenetic tree inference: optimisation

Lubica Benuskova

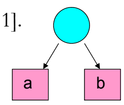
Prepared according to the notes of Dr. Richard O'Keefe

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

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?
- For example, suppose we have measured just one character with *k* values, and that we're using the Fitch algorithm to determine the cost of a tree.

Recall 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*)

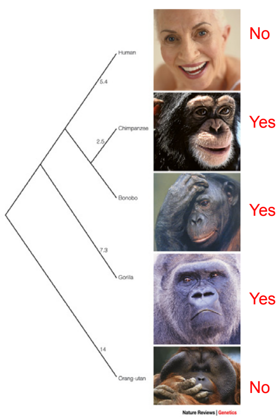
Example:

- If *value_a* = *value_b* = *k*, then Fitch of their ancestor node = (0, *k*).
- If *value_a* = k_a, *value_b* = k_b, then Fitch of the ancestor node = (1, [k_a or k_b]).

How to build the best tree: simple way

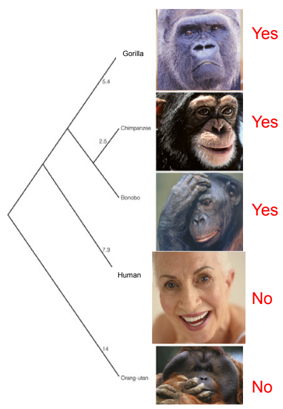
- We have measured just one character with *k* states (character values), and we're using the Fitch algorithm to determine the cost of a tree.
- Then we can build an optimal tree this way:
  - Collect all the OTUs with state 1 and build **any** tree over them. ...
  - Collect all the OTUs with state *k* and build **any** tree over them. (All these subtrees have cost 0.)
  - Build **any** tree over these subtrees. This tree will have cost *k*–1, which is the optimum.
- We learn a number of things from this example:
  - There are special cases, which can be done quickly.
  - There is, in general, no such thing as the *best* tree; there may be many trees with the same optimal cost.
  - Relying on just one character probably leads to nonsense.

Phylogeny of great apes & knuckle walking



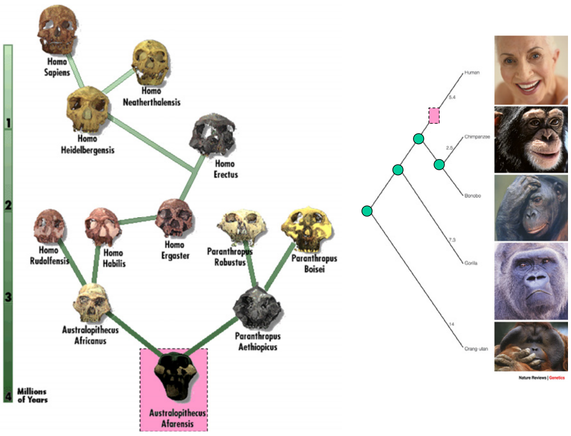
- Chimps and bonobo walk on their knuckles, and so do gorillas.
- We don't, and the people who study fossil hominids tell us that none of our ancestors ever did.
- So the "walks-on-knuckles" character doesn't fit the evolutionary tree at all well.
- It would be OK if the common ancestor of gorillas, chimpanzees, and humans was a knuckle-walker and we'd lost that feature. But the fossils say we never had it.

The most parsimonious tree



- Knuckle-walking is extremely unusual; chimps, bonobo and gorillas are the only creatures that do it at all.
- If we considered just knuckle-walking character then the most parsimonious tree would be one in which knuckle-walking evolved just once, and stayed.
- However, it seems this unusual feature has evolved and then it was lost again because other anatomical and molecular characters are held to favour the previous tree.

Evolution of hominids (i.e. us)



Parsimony method = optimisation problem

- Parsimony method for phylogeny inference:
  - Step 1: define a **cost** measure for phylogenies.
  - Step 2: announce that the best tree(s) is(are) the one(s) having the least cost.
- What we have here is a combinatorial optimisation problem: “find a tree having minimal total evolutionary cost".
- The problem is NP complete. There is no efficient way to locate a solution.
- Optimisation algorithms: greedy algorithm, random search, greedy algorithm with random restarts, hill-climbing, simulated annealing, genetic algorithm.

Greedy algorithm

- The basic idea is simplicity itself: make the *locally* best choice.
- In general, greedy algorithms do not find *global* minimum, because their focus on *local* decisions makes them "blind" to the global consequences of their decisions.
- However,
  - greedy algorithms are usually easy to develop,
  - they are usually fast to run,
  - they often give good results, and
  - they sometimes give the global optimum

Greedy algorithm

```
1. Start with a trivial (one-leaf) tree.
2. For each remaining OTU,
   1. Try inserting it in each possible place of an
      already existing tree.
   2. Commit to the one which results in the least cost.
```

- Since there are n OTUs to insert, the outer loop iterates n–1 times. When there are n leaves in the tree, there are $2n$–1 places to insert the new OTU, and determining the cost of the result requires $O(n)$ time. Thus, a greedy algorithm for inferring a phylogeny is $O(n^3)$.
- The basic operations we need for the greedy algorithm are
  - make a one-leaf tree
  - insert a new OTU at any given place
  - delete the OTU that was last inserted

Random search

- Greedy algorithm is expensive, for large n we cannot afford to try insert every OTU at every possible place.
- A simple alternative to greedy search is random search: i.e. sampling solutions at random.
- We cannot expect blind random search to land on a minimum.
- We can expect that after applying the random search many times, we’ll find out something about "average" properties of the solution space.
- We can expect to get some idea of the distribution of costs, so that we can tell where the good solutions are.

Random search algorithm

- We can make a random phylogenetic tree by:

```
1. start with a one-leaf tree
2. insert each remaining OTU at a randomly
   chosen point
```

- It is not necessary to process the OTUs in a random order, although that would do no harm.
- The operations we need for random search are
  - make a one-leaf tree
  - insert a new OTU at any given place
- Generate several thousand random “solutions”, i.e. trees and pick the best one (i.e. with the least cost).

Greedy search with random restarts

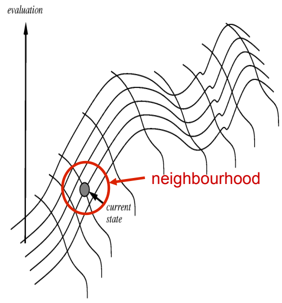
- The idea here is to combine the Greedy Algorithm's simplicity with Random Search's immunity of being trapped in local minima.
- For hundreds or thousands of times:

```
1. generate a random permutation of the OTUs
   (i.e., different random orders of OTUs).
2. use the greedy algorithm to build trees for
   these  permutations
3. pick the best result.
```

- It is similar to the algorithm called hill-climbing, but there are differences.

Hill-climbing search

- Idea of hill-climbing is this: imagine the search space as a hilly landscape, with the height at each point given by the *objective function* (evaluation = inverse of cost).
- You want to find the maximum. You start at some random point.
- At each step, in the neighbourhood of the current state you find the direction which leads upwards most steeply, and move up-hill.
- Eventually you reach a point where all directions are down, and you are at the top of a hill, but not necessarily the ***highest*** hill.



Hill-climbing for the tree search

- The greedy algorithm builds a single tree one node at a time, while the hill-climbing algorithm starts from a complete tree and at each step moves from a complete tree to a "neighbouring" complete tree.
- Hill-climbing algorithm:

```
1.  Make a complete tree.
2.  Repeat
    1. Examine all the neighbours of the current tree.
    2. Move to the best neighbour until no neighbour is
       an improvement.
```

- This leaves us with the question "what is a neighbour of a tree?"

Nearest neighbour interchanges

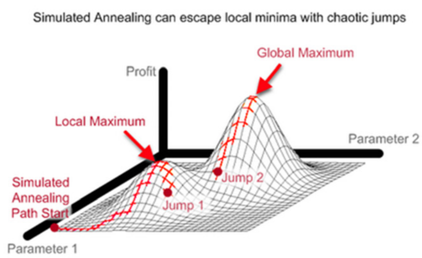
- We are looking for is a way of "stepping" from one tree to "nearby" trees such that every tree can be reached from every tree.
- Subtree pruning and re-grafting: remove any subtree (except for the whole tree, of course), remove it, and try reinserting it in all possible places. For each new tree calculate the cost of the whole tree. Keep the best tree.
- If a tree with n_1+n_2 leaves has a subtree with n_2 leaves removed, there are $2n_1-3$ places it can be put back, but one of those is the place it came from. It turns out that a tree with n leaves has about $4(n-2)(n-3)$ neighbours under this definition. For example, with n=29 there may be up to 2808 neighbours.

Variations to hill-climbing scenario

- *Random-restart hill climbing*: The idea is to repeatedly generate a random tree and do hill-climbing from there, retaining the best solution. The hope is that one of the random restarts will land us close enough to the global optimum.
- *First-choice hill climbing*: generates successors randomly until one is generated that is better than the current state *(good strategy when a state has thousands of successors)*. Then we can continue from this new state.
- *Stochastic hill climbing*: chooses the next solution with a probability proportional to the steepness of an uphill move (i.e. difference between the costs of an old and new tree).

Simulated annealing

- We always accept a solution if it improves the profit (= 1/cost).
- If the new solution makes things worse, then we accept the new solution with certain nonzero probability.
- However, once we are getting close to a good place, we want to stay there, so we keep lowering the probability of accepting bad moves.



Simulated annealing algorithm

- Here's a pseudo-code:

```
tree := random initial tree.
RND := uniform random number from interval (0, 1].
repeat until good enough result or time runs out:
  T := new temperature.
  repeat until convergence or iteration limit:
    new tree := random neighbour of a tree.
    delta := cost(new tree) - cost(tree).
    if delta <= 0: tree := new tree
       else RND < exp(-delta/T):
       tree := new tree
```

- This version is written as choosing a random neighbour, rather than exploring all the neighbours, as hill-climbing would. We thus have two kinds of randomness: random choice of a neighbour and random acceptance of "backwards" steps.

Simulated annealing

- The (metaphorical) temperature T is gradually decreasing over the course of the search. As T decreases, **`-delta/T`** gets more negative, so **`exp(-delta/T)`** gets closer and closer to 0.
- This, the so-called cooling schedule, is an important part of doing a simulated annealing search. T can decrease linearly, piece-wise linearly or exponentially (proper schedule chosen by experimentation).
- It has been shown theoretically if T decreases slow enough, algorithm converges to global optimum.
- Applied to the phylogeny problem, you need to do random subtree pruning (pick a subtree other than the root at random) and re-grafting (put it back at a random place other than where it came from).

Genetic algorithm (fitness = 1/cost)

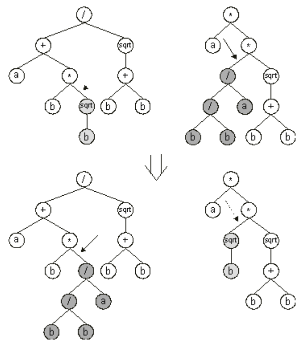
```
BEGIN
  Generate initial population of random trees;
  Compute fitness (cost) of each individual;
  REPEAT /* New generation /*
    FOR population_size DO
      Compute cost of individuals;
       Select parents from old generation;
         /* biased to the ones with smaller cost */
      Recombine parents for offspring;
      Insert offspring into new generation;
      Mutate offspring;
    END FOR
  UNTIL population has converged
END
```


Recombination of trees

- This example is taken from genetic programming webpage (hence the symbols) but it's the same idea for phylogeny trees: http://www.gene-expression-programming.com/GepBook/Chapter1/Section5.htm

Parent trees

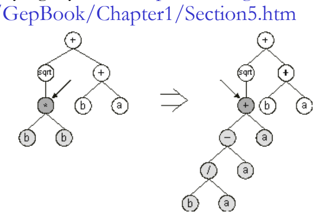
Offspring trees



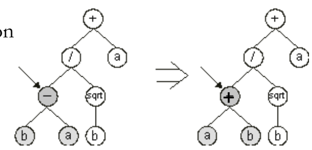
Mutation of trees

- This example is taken from genetic programming webpage (hence the symbols), but applies also for phylogeny trees: http://www.gene-expression-programming.com/GepBook/Chapter1/Section5.htm

Replaces the subtree at randomly selected node by a randomly generated subtree



Point mutation



Caveats of parsimony methods

- The cost reflects the amount or difficulty of the evolutionary change implied. Parsimony or Occam's razor in action: if there is a tree with cost c which can account for the data, it would be unscientific to propose a tree with cost $c' > c$.
- It is important to understand, however, that there may be more than one most parsimonious phylogeny, and that the true phylogeny is not necessarily the most parsimonious (i.e. with the least cost).
- In real problems, we do not know the real phylogeny. So we can never *know* that the phylogeny we compute matches what really happened; nor can we expect nature to follow our ideas of elegance. We do the best we can…