Finding all or best or some (hopefully best) trees

Peter Beerli and Michal Palczewski

September 6, 2011

1 Counting trees [JF:3; Yang:75]

1.1 Counting rooted trees

Cayley (1857) seems to be the first one to publish tree counting methods and many followed, it seems that many authors reinvented formulas for counting several times in the biological literature and the computer science literature. Counting trees might be futile exercise but it will give a good impression about how many things are contributing to a solution. It is also directly connected to the *exhaustive search* method.

Starting with a tree with two tips we easily recognize that there this only possible tree. We can add a third tip either into the branch from A to the root, into the branch from B to the root, or below the current root. There are 3 possible rooted bifurcating trees with 3 tips: ((a,b),c), ((a,c),b), ((c,b),a). We just added a node and and an additional branch. A fourth tip can be added at all 4 branches above the root and also 1 below the root, there are 5 possible branches to insert. this leaves us wit 3×5 possibilities for a rooted 4-tip tree.(figure 1).

We outlined an algorithm to calculate the total number of possible unrooted trees. We need only to realize that there are $2n-3$ possible branches were we can insert a new branch. The $2n-3$ we can get by recognizing that there are n tips and $n-2$ interior branches and one root branch. Once we recognize the series it is even simpler

$$1 \times 3 \times 5 \times 7... \times (2n-3) \tag{1}$$

We can express this in a more compact formula

$$\text{number of rooted trees} = \frac{(2n-3)!}{2^{n-2}(n-2)!} \tag{2}$$

The number of trees increase rapidly (see Table 3.1 in JF page 24).

Figure 1: A sample of all different tree shapes with 4 tips, when the trees are labeled, See Figure 3.2 on page 21 of JF

Algorithm 1 Counting rooted trees with n tips

```
S ← 1
k ← 2
while k ≤ n do
    S ← S × (2k − 3)
    k ← k + 1
end while
return S
```

1.2 Counting unrooted trees

Once we recognize that there is nothing special about the root and that any tip can be treated as a root we can apply the same formula (3) with $n-1$ tips. We can also rewrite the formula to reflect the counting on unrooted trees to

$$\text{number of unrooted trees} = \frac{(2n-5)!}{2^{n-3}(n-3)!} \tag{3}$$

1.3 Counting multifurcations

JF:25-28

When we want to count multifurcating trees there arises some ambiguity because the number of

nodes depends not only on the number tips, but also on the number of multifurcations in the tree. To get a count we simple count over all possible multifurcating nodes up to to the $n-1$ internal nodes for a given number of n tips.

1.4 Counting labeled histories

JF:35-36

For most phylogenetic purposes we are not interested in labeled histories, except we can pinpoint some internal nodes with some fossil data. In population genetics labeled histories have made

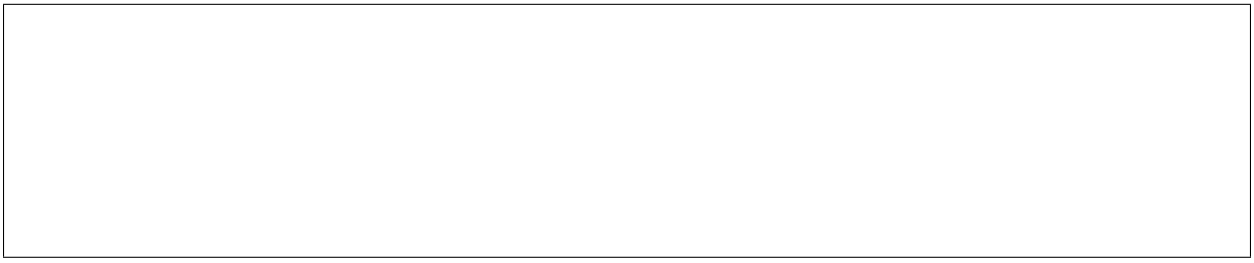

Figure 2: Example of different labeled histories

a huge impact as they are used in the framework of the coalescence theory. On rooted labeled histories we count pairs of nodes:

$$\frac{n(n-1)}{2} \times \frac{(n-1)(n-2)}{2} \times .... \times \frac{2(1)}{2} = \frac{n!(n-1)!}{2^{n-1}} \tag{4}$$

This numbers for labeled histories raise much faster than the rooted and even faster than the multifurcating trees.

2 Counting tree shapes

Computer scientist or mathematicians are often interested in counting different tree shapes or unlabeled trees (Figure 3).

Table 1: Number of trees for 2 to 20 tips

Tips	Number of labeled histories	Number of rooted trees
2	1	1
3	3	3
4	18	15
5	180	105
6	2700	945
7	56700	10395
8	1587600	135135
9	57153600	2027025
10	2571912000	34459425
11	141455160000	654729075
12	9336040560000	13749310575
13	728211163680000	316234143225
14	66267215894880000	7905853580625
15	6958057668962400000	213458046676875
16	834966920275488000000	6190283353629375
17	113555501157466368000000	191898783962510625
18	17373991677092354304000000	6332659870762850625
19	2970952576782792585984000000	221643095476699771875
20	564480989588730591336960000000	82007945326378915593 75

3 Exhaustive search

An exhaustive search simple make sure that we look at every tree for a given number of tips, this is the same approach as if we would count trees.

Begin not covered in class: In many cases it is not practical to store all the trees before applying the optimality criterion. A better approach is then to generate one tree, calculate the optimality score of that tree, generate the next tree, etc. A simple way of accomplishing this is to use a predefined order of tips and an indexing scheme specifying how the tree should be built. Assume we have m tips and that they are ordered in some predefined series. Then use m indices i_j, each taking on values from 1 to $2 * j - 5$ (the number of branches for an unrooted tree with $j - 1$ tips). The value of each index specifies how to construct a particular tree. For instance, the index series {1,1,1,3,5,7,9,11} would give us the tree resulting from combining the first three tips, then joining

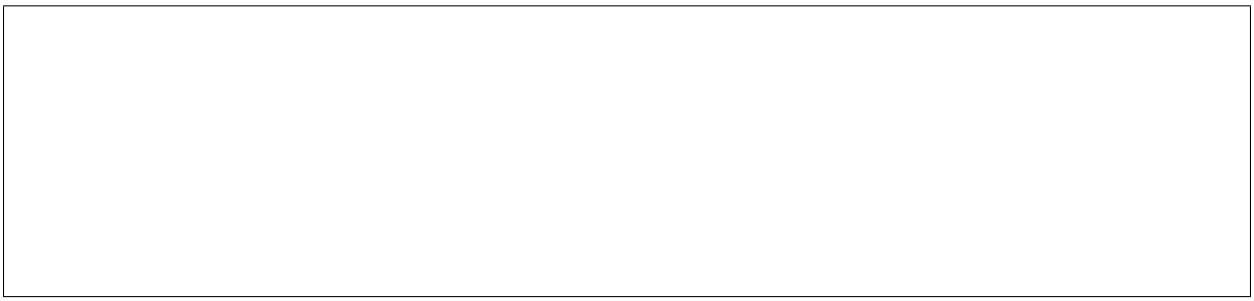

Figure 3: a sample of all different tree shapes with 4 tips, when the trees are unlabeled, page 31 JF

Algorithm 2 Exhaustive search on unrooted trees (breadth first method)

```
Tree T_3 ← join first 3 tips
for k = 4 to n do
  for all trees in T_{k-1,.} do
    for all branches in tree T_{k-1,j} do
      T_{k,i} ← add tip to branch i in tree T_{k-1,j}
    end for
  end for
end for{Now we have collected all trees T_{n,.}}
for all  all trees T_{n,.} do
  ω_i ← calculate optimality criterion for T_{n,i}
end for
sort T_{n,.} according to ω.
```

the next tip to branch 3 in that tree, the next tip to branch 5 in the resulting tree, etc. It should be relatively easy to see that each sequence of numbers designates a unique tree. For this to work, we need to have a systematic way of numbering the branches we add to a growing tree. A simple scheme is to use the next two available indices for the two branches being added to the tree. For instance, when adding tip p to a tree with n branches, we would use the index $n+1$ for the branch leading to p and the index $n+2$ for the lower part of the interior branch that is bisected when the tip is added.

Using this indexing scheme, we can exhaustively enumerate the index combinations corresponding to all possible trees by simply adding one to the index series, as if the indices represented the digits in a counter where the base was different for each digit. Such a counter algorithm is easy to formulate (Algorithm 3). Once we have the index combination, the algorithm for constructing the tree is also straightforward (Algorithm 4).**End of not covered in class**

Algorithm 3 Not covered in class: Algorithm for generating tree building index numbers

for all j **do**
  $i[j] \leftarrow 1$
end for
while $i[3] < 2$ **do**
  Do something with the index series
  $i[n] \leftarrow i[n] + 1$
  $j \leftarrow n$
  while $i[j] > 2 * j - 5$ **do**
    $i[j] = 1$
    $j \leftarrow j - 1$
    $i[j] \leftarrow i[j] + 1$
  end while
end while

Algorithm 4 Not covered in class: Algorithm for building a tree from an index series

Construct tree with 3 tips
$k \leftarrow 4$
while $k \leq n$ **do**
  Set p to tip k
  Create new node q
  Attach p to q
  Attach q underneath node with index $i[k]$
  Set index of p to $2 * k - 4$
  Set index of q to $2 * k - 3$
end while

4 Branch and bound search

This method was developed around 1960 (Computer science , who?) and was described for phylogenies by Hendy and Penny (1982). We start with a 3-tip tree (level 1) and add another tip and evaluate all 3 four-tip trees ($T_{4,.}$) (level 2) using a optimality criterion, for example parsimony. We might find that some trees have fewer changes than others. We pick next the tree with the fewest changes, $T_{4,best}$ in level 2 and add the next tip to it and evaluate the score for these trees on level 3. We move now to the second best tree on level 2 and add a tip and evaluate the scores. If a score got bigger than the lowest score on level 3 already evaluated, we can stop searching that tree-set further as all of it will have larger scores.

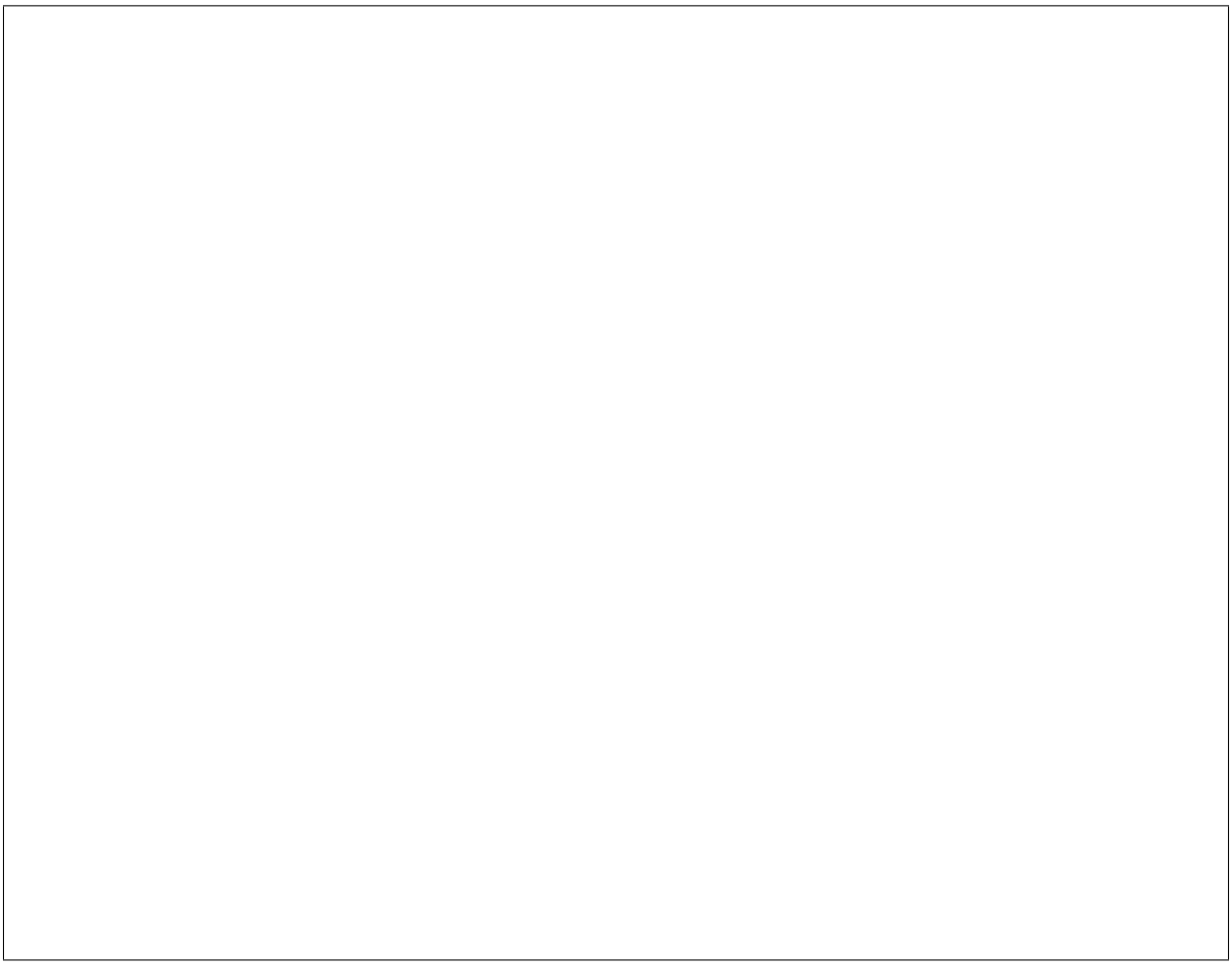

Figure 4: Branch and bound method

5 Heuristic search

There are several ways one can search the space of all possible trees. Only a limited number of possibilities are shown. We can divide heuristic methods into at least two classes:(1) greedy hill-climbing methods, (2) methods based on Markov chain Monte Carlo (MCMC) methods. Here will cover mostly the (1), the MCMC methods gets their fair share later in the course.

5.1 Stepwise addition

This is perhaps the most simple procedure to get to a "reasonably" good tree. One branch is added at a time and each position is evaluated and the best tree for each round is the base for the next round of adding another tip. Problematic for this approach is that the order of the taxa that are added to the tree matters because only a small number of trees are evaluated. This approach is "greedy" because it will climb to closest optimality peak. It is easy to see that once one has made

Algorithm 5 ATTEMPT of Branch and bound method on unrooted trees

Generate tree with 3 tips
$k \leftarrow 4$
while $k < n$ **do**
  Generate tree with k tips
  Score all k-tip trees, using an optimality criterion, for example parsimony
  Pick the tree T_b with the lowest score
  Generate trees with $k+1$ tips based on this tree T_b
  Score all $k+1$-tip trees
  Remove all the k-tip trees from the list that have lower scores than these $k+1$-tip trees.
  $k \leftarrow k+1$
end while

a bad decision one will end on a suboptimal tree (climbing a mole-hill instead of Mount Everest).

Algorithm 6 Stepwise addition

Tree $B_3 \leftarrow$ join 3 tips for first "best" tree
Optimality criterion $\omega_0 \leftarrow 0$
for $k = 4$ to n **do**
  for all branches in tree T_{k-1} **do**
    $T_{k,i} \leftarrow$ add tip to branch i in tree T_{k-1}
    $\omega_i \leftarrow$ calculate optimality criterion for $T_{k,i}$
    if $\omega_i > \omega_{i-1}$ **then**
      $B_k \leftarrow T_{k,i}$
    else
      $\omega_j = \omega_{i-1}$
    end if
  end for
end for
return B_n

5.2 Star Decomposition

This is an alternative to the *Stepwise addition* method. Instead of building up a tree, one is starting with a star tree with a single multifurcating node and calculates the optimality criterion for all trees that have two tips joined into a new group. The tree with the best optimality score is used for the next step, where one tries again to two group two nodes, and so on until the multifurcation is

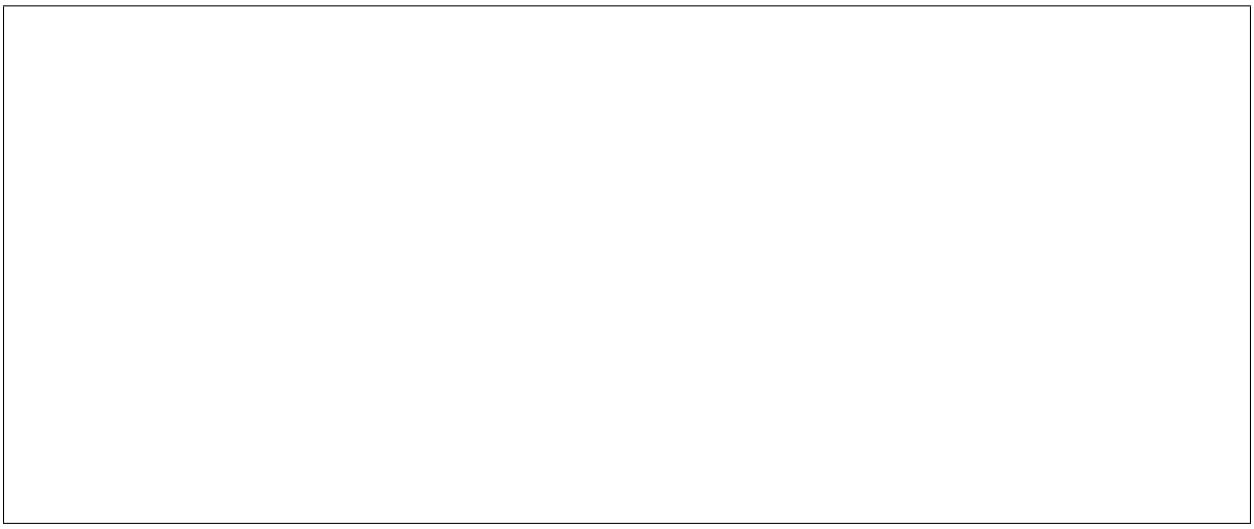

Figure 5: Stepwise addition

Figure 6: Star decomposition

resolved.

5.3 Nearest neighbor interchange

Figure 7: Nearest neighbor interchange

Pick a interior branch and then flip the attached nodes:

- the two branches attached to the nodes on each side of the branch can be flipped.
- the two branches attached to the nodes on each side of the branch, can be joined

5.4 Subtree pruning and regrafting

Algorithm 7 Subtree pruning and regrafting

break T at arbitrary branch into T_1 and T_2
for all branches in tree T_1 **do**
  $T_{1,i} \leftarrow$ add "root" of T_2
  $\omega_i \leftarrow$ calculate optimality criterion for $T_{1,i}$
  if $\omega_i > \omega_{i-1}$ **then**
    $B \leftarrow T_{1,i}$
  else
    $\omega_j = \omega_{i-1}$
  end if
end for
return B

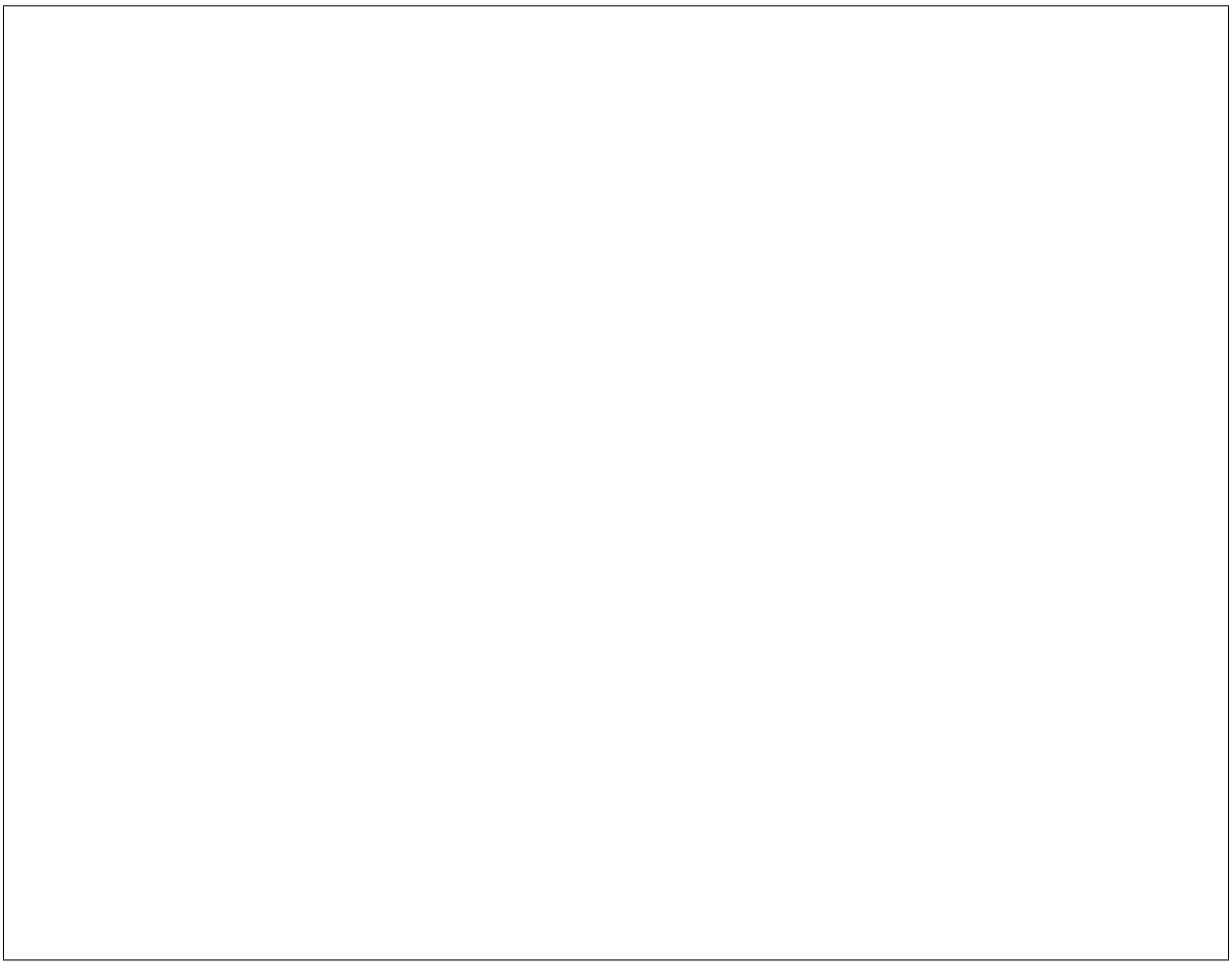

Figure 8: Tree bisection and reconnection

5.5 Tree bisection and reconnection (TBR)

5.6 Disk methods

Arrangements are done on a prescribed local area of the tree. [more to come]

5.7 Other tree rearrangement methods

JF:44

There are several other tree rearrangement possibilities that we have not discussed. some of them will come in later lectures, for example a variant of the SPR in a population genetic context, or many different tree rearrangement in Bayesian phylogenetic inference. There are also other methods, such as the tree fusing method by Goloboff (1999), where one picks two complete trees and exchanges subtrees. Methods like this might work well in a parallel-computer environment where many computer-nodes can work on the same tree-problem and exchange subtrees, of course

one would need to have a methods that makes sure that the there is no duplication or loss of taxa (tips) on any given tree.

6 Study questions

1. Describe the differences in counting between rooted and unrooted trees?
2. Unlabeled trees produce much fewer shapes than labeled trees, describe the rate of increase between the two types.
3. Describe the difference between labeled histories and labeled trees.
4. Why would you want to do an exhaustive search?
5. Write down the algorithm for the *Star decomposition* method.
6. Write down the algorithm for the *TBR* method.
7. Why is it important to try several heuristics (for example, not simply using stepwise addition) to find the best tree. Your answer need to cover more than simply mentioning greedy methods.
8. Can you come up with your own (insert your name here) tree-rearrangment? Algorithm and pictures?