Maximum likelihood II

Peter Beerli

October 3, 2005

1 Conditional likelihoods revisited

Some of this is already covered in the algorithm in the last chapter, but more elaboration on the practical procedures and why are using these might give an even better understanding. This section follows the Felsenstein book pages 251-255. We express the likelihood of the tree in Figure 1 as

$$\text{Prob}(D^{(i)}|T) = \sum_z \sum_y \sum_w \sum_x \text{Prob}(A, A, C, G, G, w, y, x, z|T)|T) \tag{1}$$

where $T = (t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8)$. Each summation is over all 4 nucleotides. The above probability can be separated into

$$\begin{aligned}\text{Prob}(A, A, C, G, G, w, y, x, z|T)|T) =& \text{Prob}(z) \\ & \times \text{Prob}(w|y, t_3)\text{Prob}(A|w, t_1)\text{Prob}(A|w, t_2) \\ & \times \text{Prob}(y|t_5, z)\text{Prob}(C|y, t_4) \\ & \times \text{Prob}(x|z, t_6)\text{Prob}(G|x, t_7)\text{Prob}(G|x, t_8)\end{aligned}$$

Prob(z) at the root is often assumed to depend on the stationary base frequencies. All parts are easy to calculate and if we order the terms of the sum in formula 1 and move the summations as far right as possible we get a summation pattern that uses the same structure as our tree ((C, (A, A)),(G, G))

$$\begin{aligned}\text{Prob}(D^{(i)}|T) = & \sum_z \text{Prob}(z) \left(\sum_y \text{Prob}(y|t_5, z)\text{Prob}(C|y, t_4) \right. \\ & \times \left. \left[\sum_w \text{Prob}(w|y, t_3)\text{Prob}(A|w, t_1)\text{Prob}(A|w, t_2) \right] \right) \\ & \times \left(\sum_x \text{Prob}(x|z, t_6)\text{Prob}(G|x, t_7)\text{Prob}(G|x, t_8) \right)\end{aligned}$$

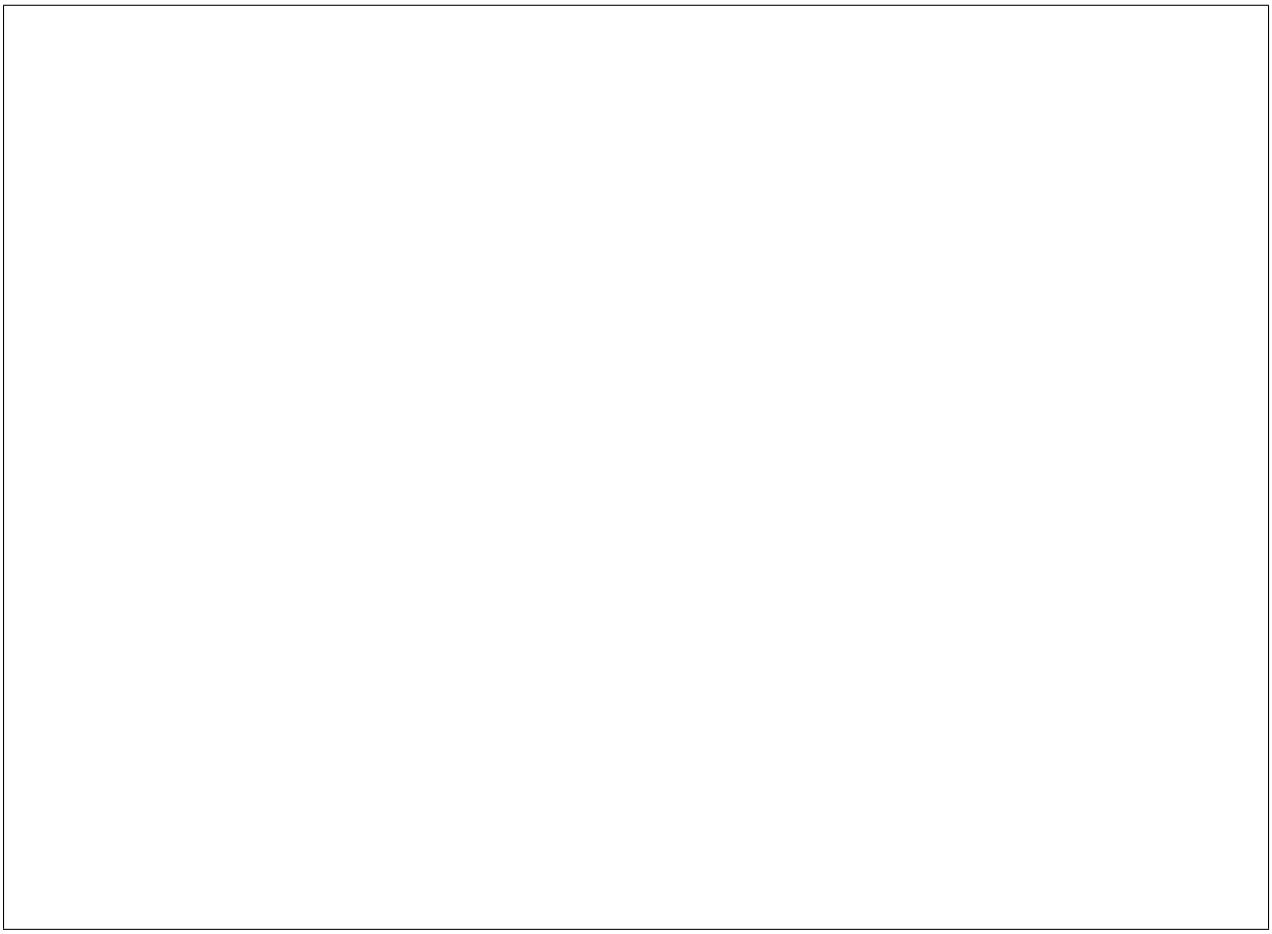


Figure 1: A tree with branch length and data

This method was introduced by Felsenstein in 1973 as *pruning* but was known before that in pedigree analyses and even long before that as a summation reduction device called Horner's rule. Even though it is named after William George Horner, who described the algorithm in 1819, it was already known to Isaac Newton in 1669 and even to the Chinese mathematician Ch'in Chiu-Shao around 1200s (Horner's rule Wikipedia.com 2005). Using the tree structure and calculating quantities (conditional likelihoods) on nodes we arrive at the solution outlined in the algorithm in the last lecture.

2 Scaling of likelihoods

With large trees the conditional likelihoods will be very small and on finite machines we need to scale the conditional likelihoods so that they do not underflow. This would have dire consequences

because when all conditional likelihoods in a node approach zero, all following node in the downpass will be zero, independent of the values of their sister nodes. The general scheme to solve underflow problems is to use a scaling quantity and also when possible logarithms. Typically the scaler is the largest value found in a loop before the the underflowing multiplication. Instead of simply adding the results of the conditionals from above with the transition probability matrix, we find the largest summand and scale the calculation with that number. Each site has its own scaler. The scaler is passed down from the tips towards the root. It is often economical not to scale the conditional likelihoods in every node but only every 5-10 nodes. To achieve this, the scalers from the daughter nodes are multiplied together (this is done typically using additions of log values), if the node is earmarked for a new scaler operation the resulting conditional likelihood is scaled by the largest value, and that log(value) is added the log-scaler.

3 Finding the maximum likelihood tree

So far we covered how to calculate likelihoods on a specific tree and earlier we talked about tree rearrangement. Of course, we can find optimal trees using likelihood as an optimality criterion. This is somewhat more difficult than evaluating tree scores using parsimony. We need to optimize continuous quantities such as the branch length and (perhaps) also the mutation model parameters AND search through all possible topologies.

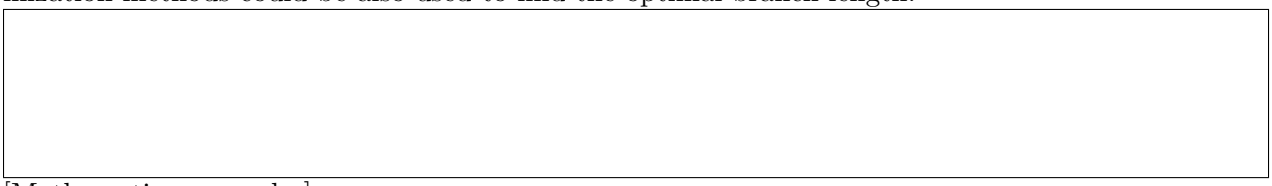
There is no simple procedure to find this best tree and only heuristic approaches are known. An approach most often used is using a heuristic search strategy on topology and then optimize all branch lengths of a promising topology. It has been shown by Steel (1994) that there can be multiple maxima, it is also true that the tree surface is not unimodal and little is known about the number of tree islands and whether they have similar or identical likelihoods. Several researchers work on such problems: can we guarantee a global maximum, how many tree islands are there.

3.1 Optimizing branch lengths

Reversible mutation models allow to treat branch lengths the same way whether we look down or up on a tree. This allows for considerable efficiency to calculate optimal branch length, so that one can optimize a single branch, quickly recalculate the likelihood on the tree and so evaluate the optimal branch length. typically this is done in one branch at a time for all branches in a tree. The procedure is strictly hill climbing because non-optimal branch-lengths are not accepted. Adding a root to a node bordering the branch length of interest changes the likelihood calculation and we

need only little recalculation. The conditional likelihood of the subtrees towards the tips did not change by this action.

For the optimization, it seems, often Newton-Raphson minimization is used. This needs analytical derivates (first and second) and might be unstable for some mutation models. Several other minimization methods could be also used to find the optimal branch length.



[Mathematica examples]

4 Optimizing mutation models

Optimizing parameters in the mutation models are rather cumbersome. Every change in the model is changing the likelihood on the tree. Branch length need to be re-optimized. Often in programs like PAUP* one is not co-optimizing the tree and the model at the same time. but running preliminary runs on fixed trees to optimize the mutation model then optimizing the tree, and perhaps use a couple of cycles to hopefully converge to the best mutation model and best tree. It might be possible to use minimization methods that use derivatives, such as Newton-Raphson minimization, but the calculation of the derivatives might be rather difficult for complex models and only numerical approximations might be available.

5 Ancestral state reconstruction

We can use the same technique as for finding the best branch length to find the site that contributes most to the likelihood. We could in turn set the root close to all the nodes of interest and recalculate the likelihood, but we have seen already that this not necessary as most of the tree does not change on such an operation and only the "views" to the branch that contains the root needs to change. These saving are similar to the ones one can do in parsimony reconstruction.

scaler.nb 1

Scaling on trees (an example using Jukes-Cantor and the tree (((A,A),C),(G,G))

■ Transition probability matrix

```
pii[t_] := 3 / 4 + 1 / 4 Exp[-4 t / 3]
pij[t_] := 1 / 4 - 1 / 4 Exp[-4 t / 3]

p[t_] := {{pii[t], pij[t], pij[t], pij[t]}, {pij[t], pii[t], pij[t], pij[t]},
  {pij[t], pij[t], pii[t], pij[t]}, {pij[t], pij[t], pij[t], pii[t]}}
```

■ Conditional likelihoods

```
condL[gi_, gj_, ti_, tj_] := Module[{},
  pi = p[ti];
  pj = p[tj];
  hi = Plus @@ (pi gi);
  hj = Plus @@ (pj gj);
  hi hj
 ]
```

■ Likelihood of a single site

```
In[437]:=
treelike[gi_, basefreq_] := Log[ Plus @@ (gi basefreq)]
```

■ Examples

```
condL[{1, 0, 0, 0}, {0, 0, 0, 1}, 0.01, 0.01]
```

```
{0.00330025, 0.0000109641, 0.0000109641, 0.00330025}
```

```
condL[%, {0, 0, 0, 1}, 0.01, 0.01]
```

```
{0.000010928, 1.08673 10^-7, 1.08673 10^-7, 0.00328939}
```

```
condL[%, {1, 0, 0, 0}, 0.01, 0.01]
```

```
{0.0000217123, 3.65449 10^-8, 3.65449 10^-8, 0.0000108559}
```

scaler.nb 2

■ Example tree (((A,A),C),(G,G))

```
w = condL[{1, 0, 0, 0}, {1, 0, 0, 0}, 0.01, 0.01]
```

```
y = condL[w, {0, 1, 0, 0}, 0.05, 0.06]
```

```
x = condL[{0, 0, 1, 0}, {0, 0, 1, 0}, 0.02, 0.02]
```

```
z = condL[y, x, 0.04, 0.08]
```

```
treelike[z, {1 / 4, 1 / 4, 1 / 4, 1 / 4}, 0]
```

```
{0.993389, 0.0000109641, 0.0000109641, 0.0000109641}
```

```
{0.018786, 0.0157197, 0.000308069, 0.000308069}
```

```
{0.0000432775, 0.0000432775, 0.986886, 0.0000432775}
```

```
{0.000468974, 0.000394289, 0.000727305, 0.0000189071}
```

```
-7.81814
```

■ Truncation problem

```
condL[{.00001, 0, 0, 0}, {0, 0, 0, .00001}, 0.1, 0.1]
```

```
{3.02328 10^-12, 9.73856 10^-14, 9.73856 10^-14, 3.02328 10^-12}
```

```
Chop[condL[{.00001, 0, 0, 0}, {0, 0, 0, .00001}, 0.1, 0.1]]
```

```
{0, 0, 0, 0}
```

```
condL[%, {1, 0, 0, 0}, 0.1, 0.1]
```

```
{0, 0, 0, 0}
```

■ Using a scaling factor

```
condLs[gi_, gj_, ti_, tj_, si_, sj_, scale_] := Module[{},
  pi = p[ti];
  pj = p[tj];
  hi = Plus @@ (pi gi);
  hj = Plus @@ (pj gj);
  gp = hi hj;
  If[scale == 1, {gp / Max[gp], Log[Max[gp]] + si + sj}, {gp, si + sj}]
 ]
```

```
treelikes[gi_, basefreq_, scaler_] := Log[(Plus @@ (gi basefreq))] + scaler
```

```
condLs[{.00001, 0, 0, 0}, {0, 0, 0, .00001}, 0.1, 0.1, 0, 1]
```

```
{{1., 0.0322119, 0.0322119, 1.}, -26.5247}
```

scaler.nb 3

■ Example tree (((A,A),C),(G,G)) using scaler

```
w = condLs[{1, 0, 0, 0}, {1, 0, 0, 0}, 0.01, 0.01, 0, 0, 1]
```

```
y = condLs[w[[1]], {0, 1, 0, 0}, 0.05, 0.06, 0, w[[2]], 0]
```

```
x = condLs[{0, 0, 1, 0}, {0, 0, 1, 0}, 0.02, 0.02, 0, 0, 0]
```

```
z = condLs[y[[1]], x[[1]], 0.04, 0.08, y[[2]], x[[2]], 1]
```

```
treelikes[z[[1]], {1 / 4, 1 / 4, 1 / 4, 1 / 4}, z[[2]]]
```

```
{{1., 0.0000110371, 0.0000110371, 0.0000110371}, -0.00663341}
```

```
{{0.018911, 0.0158243, 0.000310119, 0.000310119}, -0.00663341}
```

```
{{0.0000432775, 0.0000432775, 0.986886, 0.0000432775}, 0}
```

```
{{0.64481, 0.542123, 1., 0.0259961}, -7.22616}
```

```
-7.81814
```

Figure 2: Scaling of conditional likelihoods [Example with mathematica]

Figure 3: Different peaks in the surface with 3 tips

Figure 4: Minimizing recalculation of branch length