Implementation of *BEAST with MUL-tree

Graham Jones

2012-01-09

1 Introduction

Note on implementation of of model ('AlloppMUL') which is basically *BEAST applied to a multiply labelled tree. This does not represent an evolutionary process, but it is a much simpler model than the network approach. Node times and populations are allowed to vary with no constraints coming from the fact that diploid genomes in the allotetraploids belonging to the same population (and species). The model therefore allows parameter values (topology, node times, and populations within allotetraploid subtrees) which are not actually possible in reality.

2 The formula

1. M is the multiply labelled tree.
2. θ is the population parameters.
3. λ is the parameter(s) for M, that is, the topology and node times. λ could consist of a speciation rate for the Yule model, or birth-death parameters.
4. η is the population mean, appearing in a hyperprior for θ
5. n is the number of gene trees.
6. τ_i is the i'th gene tree topology and node times.
7. α_i is all the other parameters belonging to the i'th gene tree: parameters for site rate heterogeneity, substitution model, branch rate model, root model.
8. g_i is (τ_i, α_i), that is, all the parameters for the i'th gene tree.
9. γ_i is the permuations of sequences within individuals for the i'th gene.
10. d_i is the sequence data for the i'th gene.

$\tau = (\tau_1, ...\tau_n)$, and similarly for α, g, γ, d.

$$\begin{aligned}
\Pr(M, \theta, g, \gamma | d) \quad \propto \quad & \Pr(M|\lambda) \Pr(\lambda) \times & (1) \\
& \Pr(\theta|\eta) \Pr(\eta) \times & (2) \\
& \Pr(\gamma) \times & (3) \\
& \prod_i \Pr(\tau_i | M, \theta, \gamma_i) \times & (4) \\
& \prod_i \Pr(d_i | g_i) & (5)
\end{aligned}$$

1. $\Pr(M|\lambda) \Pr(\lambda)$ is the network prior: the probability of M before seeing any molecular data.
2. $\Pr(\theta|\eta) \Pr(\eta)$ is the population prior.

3. $\Pr(\gamma)$ is the permutation prior. Quite likely uniform, so can be ommitted.
4. $\Pr(\tau_i | M, \theta, \gamma_i)$ is the probability of τ_i, when permuted by γ_i, fitting into the multiply labelled tree M with populations determined by θ. Note that this probability does not depend on α_i.
5. $\Pr(d_i | g_i) = \Pr(d_i | \tau_i, \alpha_i)$ is the 'Felsenstein likelihood' of the data for the i'th gene given the i'th gene tree.

This is essentially the same as for *BEAST except for the addition of the permutations, and therefore the need for an operator that re-assigns sequences.

3 Adding to *BEAST vs modifying allopolyploid network

It would be possible to implement AlloppMUL either by adding to *BEAST or by modifying the code for an allopolyploid network. I have chosen the former, which seems likely to be the quickest to implement, though maybe not the best way.

4 List of classes

4.1 Parsers in `dr.evomodelxml.operators`

- `MulTreeNodeSlideParser`
- `MulTreeSequenceReassignmentParser`

4.2 Parsers in `dr.evomodelxml.speciation`

- `MulMSCoalescentParser`
- `MulSpeciesBindingsParser`
- `AlloppNetworkPriorParser`
- `MulSpeciesTreeModelParser`

4.3 Classes in `dr.evomodel.operators`

- `MulTreeNodeSlide`. An operator which changes node heights and tree topology within a homoploid tree and changes hybridization times and split height in no-diploid case.
- `MulTreeSequenceReassignment` An operator which changes assignments of sequences within an individual

4.4 Classes in `dr.evomodel.speciation`

- `MulMSCoalescent` computes coalescent log-likelihood of a set of gene trees embedded inside a multiply labelled species tree. It is an `instanceof Likelihood`.
- `MulSpeciesBindings` knows how species are made of individuals and individuals are made of taxa (= diploid genomes within individuals). It is an `instanceof Model`.
- `MulSpeciesTreeModel` implements the multiply labelled species tree. It is an `instanceof Model`.

5 Details of the major classes

The remaining sections describe the major classes.

6 Operator `MulTreeNodeSlide`

Very similar to `TreeNodeSlide` in *BEAST. I think I should be able to re-use the code, but for now it is copy-and-paste, with different types (`MulSpeciesTreeModel`, `MulSpeciesBindings` not `SpeciesTreeModel`, `SpeciesBindings`). Maybe I should have a '`TreeNodeSlidable`' interface.

7 Operator `MulTreeSequenceReassignment`

This is very similar to `AlloppSequenceReassignment`. It calls either `permuteOneSpeciesOneIndivForOneGene()` or `permuteSetOfIndivsForOneGene()` in `MulSpeciesBindings`.

8 Likelihood `MulMSCoalescent`

Similar to `MultiSpeciesCoalescent` in *BEAST. In `calculateLogLikelihood()` I have done the compatibility checks always. In *BEAST, `checkCompatibility` and `compatibleCheckRequired[]` keep track of what needs to be updated. Re-assigning sequences invalidates these, and I haven't figured out how to implement an efficient check. Maybe in `modelChangedEvent()` ?

9 Model `MulSpeciesBindings`

This is a sort of combination of `SpeciesBindings` from *BEAST and `AlloppSpeciesBindings` from AlloppNET model.

*BEAST uses a different way of collecting and representing the information about coalescences (times and species info) and the numbers of lineages in different branches) to my implemtation of AlloppNET. This is the information used for determining compatibilty, and calculating likelihoods. I have used the *BEAST code but don't have a good understanding of how it works.

*BEAST also has a complex model for population sizes which I don't understand. How do `popTimesSingle` and `popTimesPair` work? I think a lot of the complexity is not needed for AllopMUL, but it is not yet clear how to remove it.

The code from `AlloppSpeciesBindings` deals with the species-individual-sequence structure of the data, and implements inner class `GeneTreeInfo` which contains the sequence assignments.

9.1 Methods

Only ones (co-)written by me listed here.

`nSpSeqs()` returns the number of species-sequence pairs in the alignment (one per diploid, two per tetraploid). From outside this class, this plays a similar role to that of the number of species in *BEAST. It is the number of tips in the MUL tree.

`apspeciesName()`, `spandseq2spseqindex()`, `spseqindex2sp()`, `spseqindex2seq()`, `apspeciesId2speciesindiv()` are utilliy functions that navigate the species-individual-sequence structure.

`permuteOneSpeciesOneIndivForOneGene()`, `permuteSetOfIndivsForOneGene()` implement the sequence re-assignments.

`collectCoalInfo()`. It is here that the sequence assignments are used in order to provide information that is used to test compatibility with species tree, and calculate $\Pr(\tau_i|M, \theta, \gamma_i)$, the likelihood of the i'th gene tree fitting into the network.

`GeneTreeInfo()` constuctor, like for `AlloppSpeciesBindings`.

`getSeqassigns()`. Does what it says.

`storeSequenceAssignments()`, `restoreSequenceAssignments()`. Do what they say, for MCMC accept/reject.

`permuteOneSpeciesOneIndiv()`, `permuteSetOfIndivs()` operate on one gene chosen by `permuteOneSpeciesOneIndivForOneGene()`, `permuteSetOfIndivsForOneGene()`.

private to `GeneTreeInfo`:

`collectIndivsOfNode()`, `permuteOneAssignment()`.

`handleModelChangedEvent()` Not yet clear what this should do.

`storeState()` stores sequence assignments, deals with other parameters like *BEAST.

`restoreState()` retores sequence assignments, deals with other parameters like *BEAST.

`getColumns()` logs sequence assignments.

private to `MulSpeciesBindings`:

`spseqindex2spandseq()`.

10 Model `MulSpeciesTreeModel`

Very similar to `SpeciesTreeModel` in *BEAST. Not clear how to deal with this, but for now, it is mostly copy-and-paste. It contains a `MulSpeciesBindings` which is seriously different from a `SpeciesBindings`.