# Contents

1 Introduction .......................................................... 3
   1.1 Jargon .............................................................. 3

2 AlloppNET overview .................................................. 4
   2.1 Arbitrary numbers of diploids and hybridizations ............... 4
   2.2 AlloppNET formula ................................................ 4

3 Overall code structure of AlloppNET ................................ 7
   3.1 List of classes .................................................... 7
      3.1.1 Parsers in `dr.evomodelxml.operators` ....................... 7
      3.1.2 Parsers in `dr.evomodelxml.speciation` ....................... 7
      3.1.3 Classes in `dr.evomodel.operators` ......................... 8
      3.1.4 Classes in `dr.evomodel.speciation` ......................... 8
      3.1.5 Classes in `dr.evolution.tree` .............................. 9
      3.1.6 Classes in `test.dr.evomodel.speciation` .................... 9
      3.1.7 Classes in `dr.util` ......................................... 9
   3.2 Tests ................................................................ 9
      3.2.1 Test 1: network to MUL-tree .................................. 9
      3.2.2 Test 2: \( \Pr(\tau_i|W, \theta, \gamma_i) \) ........................ 9
      3.2.3 Test 3: network to diploid history .......................... 9

4 Interfaces and classes in AlloppNET ................................ 10
   4.1 Interface `AlloppNode` ............................................. 10
   4.2 Interface `SlidableTree` .......................................... 11
      4.2.1 Methods ......................................................... 11
   4.3 Operator `AlloppNetworkNodeSlide` ................................ 11
      4.3.1 Methods ........................................................ 11
   4.4 Operator `AlloppSequenceReassignment` .......................... 12
      4.4.1 Methods ........................................................ 12
   4.5 Operator `AlloppMoveLegs` ........................................ 12
   4.6 Diploid history `AlloppDiploidHistory` .......................... 12
      4.6.1 Inner classes .................................................. 12
      4.6.2 Methods ........................................................ 13
   4.7 Homoploid tree model `AlloppLeggedTree` ....................... 13
      4.7.1 Methods ........................................................ 14
   4.8 Likelihood calculator `AlloppMSCoalescent` ...................... 15
Chapter 1

Introduction

This describes the implementation of two models called AlloppMUL and AlloppNET. AlloppMUL is much simpler and is described in Chapter 5.

It is a work in progress.

1.1 Jargon

I have started using the words leg, foot, and union to mean special things. Also, I use MNL move and hyb-tip.

The species network is composed of trees, which are joined together. A higher ploidy tree is joined to a lower ploidy tree with one or two legs. The point where a leg meets the branch of the lower ploidy tree is a foot.

A union is a set of indices where each index represents a species, or more often, a species and a sequence. For example, if a, b, c, ... are species, 1, 2, 3, ... are individuals and A, B, C, ... label sequences, so that a single sequence (taxon) in an alignment might be c2B, then a union can represent a set of elements like \{aA, bA, bC\}. The identity of individuals 1,2,3, ... is ignored. At a tip in a gene tree, or a tip in the multiply labelled tree, there will be a single species and a single sequence, like aA or bC. At internal nodes, there are sets of them, each one being a union of its children’s sets. The union at a node is unique to that node in the multiply labelled tree (with one exception near the root in the no-diploids case). Unions can therefore be used to identify nodes. In the species network, the identity of the sequence is lost, but a similar mechanism is used.

A MNL move is a MCMC move for trees based on the ideas of Mau, Newton and Larget (1999). Similar moves are used in *BEAST. The tree is randomly oriented (so all nodes ordered left to right), a random internal node is chosen and its height altered, then the tree is reconstructed.

A hyb-tip, short for ‘hybridization tip’ is a node at some time before present in the diploid part of the network, which is the point at which a diploid hybridized with another. If the tetraploid subtrees are removed from the network, these nodes become tips in what is left.
Chapter 2

AlloppNET overview

2.1 Arbitrary numbers of diploids and hybridizations

I am working on a version for arbitrary numbers of diploids and one hybridization, to be followed by a version for arbitrary numbers of hybridizations. I have re-organised the code to facilitate this (mostly during April 2012). New classes and interfaces:

- **AlloppDiploidHistory** is main new code.
- **AlloppFootLinks** was FootLinks in AlloppMulLabTree.
- **AlloppLegLink** was LegLink in AlloppMulLabTree.
- **AlloppNode** is an interface.
- **AlloppTreeLeg** was Leg in AlloppLeggedTree.
- **SlidableTree** is an interface.

**AlloppNetworkNodeSlide** has become more complex. I think that it makes **AlloppMoveLegs** and **AlloppMoveLegsParser** redundant and it would need too much work to make them deal with more than two diploids, but I haven’t removed them yet.

2.2 AlloppNET formula

1. $W$ is the network topology and node times. It can also be seen as a multiply labelled tree.
2. $\theta$ is the population parameters. See 2012-03-30-popvalues-to-nodes.pdf for details.
3. $\lambda$ is the parameters for $W$, that is, the topology and node times. $\lambda$ could consist of a speciation rate, an extinction rate, and a hybridization rate.
4. $\eta$ is the population mean, appearing in a hyperprior for $\theta$
5. $n$ is the number of gene trees.
6. \( \tau_i \) is the i’th gene tree topology and node times.
7. \( \alpha_i \) 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 \((\tau_i, \alpha_i)\), that is, all the parameters for the i’th gene tree.
9. \( \gamma_i \) is the permutations of sequences within individuals for the i’th gene.
10. \( d_i \) is the sequence data for the i’th gene.

\[ \tau = (\tau_1, \ldots, \tau_n), \text{ and similarly for } \alpha, g, \gamma, d. \]

\[
\begin{align*}
\Pr(W, \theta, g, \gamma | d) & \propto \Pr(W|\lambda) \Pr(\lambda) \times \\
& \Pr(\theta | \eta) \Pr(\eta) \times \\
& \Pr(\gamma) \times \\
& \prod_i \Pr(\tau_i | W, \theta, \gamma_i) \times \\
& \prod_i \Pr(d_i | g_i)
\end{align*}
\]

1. \( \Pr(W|\lambda) \Pr(\lambda) \) is the network prior: the probability of \( W \) 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 omitted.
4. \( \Pr(\tau_i | W, \theta, \gamma_i) \) is the probability of \( \tau_i \), when permuted by \( \gamma_i \), fitting into the network \( W \) with populations determined by \( \theta \). Note that this probability does not depend on \( \alpha_i \). See below for what permuting \( \tau_i \) by \( \gamma_i \) means.
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.

I previously thought of

\[
\Pr(d_i | g_i, \gamma_i) \Pr(g_i | W, \theta)
\]

as

\[
\Pr(\gamma_i (d_i)|g_i) \Pr(g_i | W)
\]

so that the \( \gamma_i \) are thought of as permuting the sequence data \( d_i \). This doesn’t work well in implementation in BEAST: lots of existing code does not expect sequences to be swapped around. It would be possible to regard the \( \gamma_i \) as specifying a topological change in a gene tree (eg two terminal branches would be swapped). However, it seems simplest to regard \( \gamma_i \) as permuting the tip labels of the gene tree. The sequences attached to a tip don’t change, nor does the gene tree topology. Instead, the way in which the sequences are assigned tips in the multiply labelled tree \( W \) is changed. Thus one can write:
\[ \Pr(\tau_i | W, \theta, \gamma_i) = \Pr(\gamma_i(\tau_i) | W, \theta) \]
Chapter 3

Overall code structure of AlloppNET

List of classes, and tests.

3.1 List of classes

3.1.1 Parsers in dr.evomodelxml.operators

- AlloppNetworkNodeSlideParser
- AlloppSequenceReassignmentParser
- AlloppMoveLegsParser

3.1.2 Parsers in dr.evomodelxml.speciation

- AlloppMSCoalescentParser
- AlloppNetworkPriorModelParser
- AlloppNetworkPriorParser
- AlloppSpeciesBindingsApSpInfoParser
- AlloppSpeciesBindingsIndividualParser
- AlloppSpeciesBindingsParser
- AlloppSpeciesNetworkModelParser
### 3.1.3 Classes in `dr.evomodel/operators`

- **AlloppNetworkNodeSlide**: An operator which changes node heights and tree topology within a homoploid tree and changes hybridization times and split height in no-diploid case.
- **AlloppSequenceReassignment**: An operator which changes assignments of sequences within an individual.
- **AlloppMoveLegs**: An operator which changes the way a tetraploid subtree joins the diploid tree.

### 3.1.4 Classes in `dr.evomodel/speciation`

- **AlloppDiploidHistory**: Represents the part of the network before hybridizations. It is basically a tree with some tips representing diploid species at present time and others representing the points at which hybridization occurs.
- **AlloppFootLinks**: For gathering and organising the links between trees of different ploidy, so that the rootward-pointing legs can become tipward-pointing branches. Used to construct `AlloppDiploidHistory`, `AlloppMulLabTree`.
- **AlloppLeggedTree**: A homoploid ‘tree with legs’.
- **AlloppLegLink** as `AlloppFootLinks`.
- **AlloppNode**: An interface implemented by `DipHistNode` in `AlloppDiploidHistory`, and `MulLabNode` in `AlloppMulLabTree`. The ABC `AlloppNode.Abstract` contains some common functionality, especially for constructing `AlloppDiploidHistory`, `AlloppMulLabTree`.
- **AlloppMSCoalescent**: Computes coalescent log-likelihood of a set of gene trees embedded inside a allopolyploid species network. It is an `instanceof Likelihood`.
- **AlloppMulLabTree**: Implements a multiply labelled binary tree. Its nodes can store populations and hybridization times.
- **AlloppNetworkPrior**: Computes log-likelihood of prior for the network. It is an `instanceof Likelihood`.
- **AlloppNetworkPriorModel**: Stores parameters for network prior (eg rates). It is an `instanceof Model`.
- **AlloppSpeciesBindings**: Knows how species are made of individuals and individuals are made of taxa (= diploid genomes within individuals). It is an `instanceof Model`.
- **AlloppSpeciesNetworkModel**: Implements the species network as a collection of ‘tree with legs’, that is, `AlloppLeggedTree`'s, and converts this representation into a multiply labelled binary tree, that is, an `AlloppMulLabTree`. It is an `instanceof Model`.
- **AlloppTreeLeg**: Is part of a `AlloppLeggedTree`, and used by `AlloppDiploidHistory`. 

8
3.1.5 Classes in \texttt{dr.evolution.tree}

- \texttt{SlidableTree} An interface implemented by \texttt{AlloppDiploidHistory}, \texttt{AlloppLeggedTree}. It is a minimal tree interface that supports MNL moves.

3.1.6 Classes in \texttt{test.dr.evomodel.speciation}

- \texttt{AlloppSpeciesNetworkModelTEST} is for testing.

3.1.7 Classes in \texttt{dr.util}

- \texttt{AlloppMisc} is odds and ends, mainly for testing.

3.2 Tests

I have written three JUnit tests in \texttt{AlloppSpeciesNetworkModelTEST}. I have decided to create a new (inner) class for each test, which is passed to constructors and other functions. This distinguishes test code from normal code, and sometimes the class is used to supply extra information needed for the test. In other cases it looks messy.

3.2.1 Test 1: network to MUL-tree

I have code for translating a network representation into a mullab representation, which is tested by some small cases in a test unit \texttt{AlloppSpeciesNetworkModelTest}. That does 3 tetra species, and 2 diploid species, in various arrangements. This isn’t very relevant to the tetra only case, but it will be needed.

3.2.2 Test 2: $\Pr(\tau_i|W, \theta, \gamma_i)$

The second test is a likelihood calculation for $\Pr(\tau_i|W, \theta, \gamma_i)$ for a case with two tetraploids and five individuals. It is compared to the result calculated in R.

3.2.3 Test 3: network to diploid history

The third test is for the construction of a diploid history from a network, and the reverse: ie the replacement of the diploid tree and tetraploid tree legs by a possibly altered diploid history.
Chapter 4

Interfaces and classes in AlloppNET

The next two sections describe two interfaces. The remaining sections describe the major classes. They are in order of packages, and alphabetically within packages. Minor classes AlloppFootLinks, AlloppLegLink, AlloppTreeLeg are ignored.

4.1 Interface AlloppNode

In evomodel.speciation.

Implemented by MulLabNode and DipHistNode which both extend AlloppNode.Abstract.

nofChildren(), getChild(int ch), getAnc() getTaxon() getHeight(), getUnion(). These are all get methods.

setChild() setAnc(), setTaxon(), setHeight(), setUnion(), addChildren(). These are all set or simple edit methods.

fillinUnionsInSubtree(AlloppSpeciesBindings apsp). Recursive. If node has children, it calls itself on both children, then makes this node have clade equal to the union of the clades of its two children. If tips are filled, calling this on root does whole tree.

nodeOfUnionInSubtree(FixedBitSet x). Searches subtree rooted at node for most tipward node whose union contains x. If x is known to be a union of one of the nodes, it finds that node, so acts as a map union to node.

AlloppNode.Abstract implements the last two. It also has static methods as follows.

convertLegLinks(). For constructors of AlloppDiploidHistory and AlloppMulLabTree. This converts rootward pointing legs into tipward pointing branches.

simpletree2allopptree(). For constructors of AlloppDiploidHistory and AlloppMulLabTree. Recursively copies the topology of the SimpleTree into tree implemented as array. Fills in the unions at the tips, using apsp which converts species name and sequence index into union.
4.2 Interface SlidableTree

In evolution.tree.
Implementation of MNL move. The moves are applied to tetratrees as AlloppLeggedTrees and the DiploidHistory.

4.2.1 Methods

mauCanonical(). Returns a left-right ordering of nodes with randomly flipped children.

mauReconstruct(). Makes a new tree after a node height has changed.

Private:

mauCanonicalSub(). Recursive, for mauCanonical().

mauReconstructSub(). Recursive, for mauReconstruct().

highestNode() For mauReconstruct().

4.3 Operator AlloppNetworkNodeSlide

In evomodel.operators.

4.3.1 Methods

Implements standard simple methods for operator. The key one is doOperation() which calls operateOneNodeInNet().

Private:

randomnode(). Chooses a random node in network, which can mean a node in a homoploid tree or a hybridization event.

operateOneNodeInNet(). Calls randomnode() then one of operateOneNodeInLeggedTree() or operateHybridHeightInLeggedTree() or operateOneNodeInDiploidHistory().

operateHybridHeightInLeggedTree(). Moves the hybridization height to somewhere between root of tree and split height or most recent foot height.

operateOneNodeInLeggedTree(). Does a MNL move. It avoids moving the root before the hybridization height as well as limits from gene trees. It calls SlidableTree.Utils.mauCanonical() and SlidableTree.Utils.mauReconstruct().
operateOneNodeInDiploidHistory(). Does a MNL move. It obeys three restrictions when choosing a new node height. It avoids making nodes so ancient that they become incompatible with any of the gene trees. It avoids making parent nodes of hyb-tips more recent than the hyb-tips. It keeps the root a diploid by preventing the root becoming too recent, or another node becoming too ancient. It calls SlidableTree.Utils.mauCanonical() and SlidableTree.Utils.mauReconstruct().

4.4 Operator AlloppSequenceReassignment

In evomodel.operators.
Operator which changes the assignment of sequences belonging to a randomly chosen individual in a randomly chosen species.

4.4.1 Methods

Implements standard simple methods for operator. The key one is doOperation() which calls permuteOneGeneOneSpeciesOneIndiv() on the AlloppSpeciesBindings.

4.5 Operator AlloppMoveLegs

In evomodel.operators.
Operator which moves the legs of a tetraploid subtree, ie, changes the way it joins the diploid tree. Calls moveLegs() in AlloppSpeciesNetworkModel. OOD?

4.6 Diploid history AlloppDiploidHistory

In evomodel.speciation.
This a tree with some tips representing diploid species at present time and others representing the points at which hybridization (to form a tetraploid) occurs. The latter are in pairs and have times before present, and I call them ‘hybridization tips’ or ‘hyb-tips’.

4.6.1 Inner classes

HybHistory stores the legs of a tetraploid subtree; in other words, the part of an AlloppLeggedTree apart from the subtree itself. This means one or two AlloppTreeLeg’s, a hybridization height, and in case of one leg only, a split height.

DipHistNode is a private class used to represent the diploid history as a tree, in a way that supports MNL moves, and ‘knows’ how it fits into the network.
DipHistTESTINGNode is for testing. It is filled from a SimpleTree, not directly from DipHistNode’s.

HybTipPair is a small private class used for extracting the HybHistory’s.

### 4.6.2 Methods

The purpose of this class is to extract the part of the network before hybridizations (= ‘diploid history’) in a form that can be subjected to MNL moves, and (after such a move), provide methods for re-making the network. The constructor makes the diploid history, and the methods extractHybHistory() and ditreeFromDipHist() are used by AlloppSpeciesNetworkModel to re-make the network.

AlloppDiploidHistory(). Main constructor. Passed a AlloppSpeciesNetworkModel from which the AlloppDiploidHistory is made. This is an array of DipHistNode’s. The constructor also makes a SimpleTree representing the diploid history, for testing only.

There is another constructor which uses a collection of trees and a AlloppSpeciesBindings (which is needed for making unions). This constructor is used for testing, and by the main constructor.

diphistTreeAsText(), diphistTreeAsUniqueNewick(), diphistTreeAsNodeList(), nofTettrees(). For testing.

extractHybHistory(). This supplies the HybHistory for a given tetraploid subtree.

ditreeFromDipHist(). Supplies the diploid tree on extant tips only, as a SimpleTree.

getRoot(), setRoot(), getNodeCount(), getNodeHeight(), setNodeHeight(), isExternal(), getChild(), replaceChildren(), getNodeTaxon(). These are overrides for SlidableTree interface.

Private:

addhybtiptodiphisttree(). For constructor.

makesimpletree(). For testing, used by constructor.

makesimplesubtree(). For makesimpletree().

getchybTipPair(). For extractHybHistory().

findKeeperNode(). For ditreeSubtree().

ditreeSubtree(). For ditreeFromDipHist().

### 4.7 Homoploid tree model AlloppLeggedTree

In evomodel.speciation.

This is a ‘tree with legs’, which is used for a homoploid species tree which is attached to a tree of lower ploidy via its legs. It is also used for the diploid tree; in this case there are no legs.
The tree is a SimpleTree. Its nodes contain taxa at tips, and heights. There are several arrangements of legs.

NONE: for the diploid tree.

TWO_BRANCH: Two legs attached to different branches in a lower ploidy tree.

ONE_BRANCH: Two legs attached to the same branch in a lower ploidy tree at different times.

JOINED: One leg, representing two species which (going back in time) join before their ancestor is attached to a branch in a lower ploidy tree.

NODIPLOIDS: similar to JOINED, but a single diploid 'trunk' is assumed which has no tips.

The tree has a hybridization height before the root, and in cases JOINED and NODIPLOIDS, a split height before hybridization.

4.7.1 Methods

AlloppLeggedTree(). Constructor. Passed an array of Taxons, plus a leg type. Makes a random Yule-type tree and fills in random times for hybrid height and legs(s) and split time.

AlloppLeggedTree(AlloppLeggedTree). Clone constructor for store and restore methods.

There are also two constructors for testing, which make small nonrandom trees.

scaleAllHeights(). Passed a scaling factor this multiplies all heights in the tree and legs: node heights, hybrid heights, leg heights, split height.

getInternalHeights(). Returns array of speciation heights, for use by AlloppNetworkPrior.

getRootHeight(). Returns what it says.

getSplitHeight(). Returns what it says.

getHybridHeight(). Returns what it says.

getMaxFootHeight(). Returns maximum height of all (0,1 or 2) legs.

getMaxHeight(). Returns maximum of all heights. Can be the root height, a foot height or the split time, depending on leg type.

setFootUnion(). Sets the ‘foot union’ of a specified leg. The foot union defines a node within a species tree by specifying the (species index, sequence index) to which the leg is attached.

getFootUnion(). Passed a leg index, it returns what it says.

getNumberOfLegs(). Passed a leg index, it returns what it says.

getFootHeight(). Passed a leg index, it returns what it says.

setHybridHeight(). Modifies what it says. Called by AlloppNetworkNodeSlide.

setSplitHeight(). Modifies what it says. Called by AlloppNetworkNodeSlide.

moveSplitOrLeg(), moveMostRecentLegHeight(), moveMostAncientLegHeight(), moveLegTopology(). Moves. Called by moveLegs() in AlloppSpeciesNetworkModel.

logNow(int state). Not yet implemented.
Private:

randomnodeheight(). Used by constructor.
randomsplitheight(). Used by constructor.

Delegations for Tree:

getRoot(), getNodeCount(), getNode(int i), getInternalNode(int i),
getExternalNode(int i), getExternalNodeCount(), getInternalNodeCount(),
getNodeTaxon(NodeRef node), hasNodeHeights(), setNodeHeight(NodeRef node),
hasBranchLengths(), getBranchLength(NodeRef node), getNodeRate(NodeRef node),
getNodeAttribute(NodeRef node, String name), getNodeAttributeNames(NodeRef node),
isExternal(NodeRef node), isRoot(NodeRef node), getChildCount(NodeRef node),
getChildren(NodeRef node, int j), getParent(NodeRef node), getCopy(), getTaxonCount(),
getTaxon(int taxonIndex), getTaxonId(int taxonIndex), getTaxonIndex(String id),
getTaxonIndex(Taxon taxon), asList(),
getTaxonAttribute(int taxonIndex, String name), iterator(), getUnits(),
setUnits(Type units), setAttribute(String name, Object value),
getAttribute(String name), Iterator<String> getAttributeNames(),
addTaxon(Taxon taxon), removeTaxon(Taxon taxon),
setTaxonId(int taxonIndex, String id),
setTaxonAttribute(int taxonIndex, String name, Object value),
addMutableTaxonListListener(MutableTaxonListListener listener)

Delegations for MutableTree:

beginTreeEdit(), endTreeEdit(), addChild(NodeRef parent, NodeRef child),
removeChild(NodeRef parent, NodeRef child),
replaceChild(NodeRef node, NodeRef child, NodeRef newChild), setRoot(NodeRef root),
setNodeHeight(NodeRef node, double height), setNodeRate(NodeRef node, double rate),
setBranchLength(NodeRef node, double length),
setNodeAttribute(NodeRef node, String name, Object value),
addMutableTreeListener(MutableTreeListener listener)

4.8 Likelihood calculator AlloppMSCoalescent

In evomodel.speciation.

Computes coalescent log-likelihood of a set of gene trees embedded inside a allopolyploid species
network.
4.8.1 Methods

AlloppMSCoalescent(). Constructor. Stores AlloppSpeciesBindings and AlloppSpeciesNetworkModel and adds itself to their model-listeners.

There is also a constructor for testing.

Implements standard simple methods for likelihood. Key ones follow.

calculateLogLikelihood(). Calls geneTreeFitsInNetwork and geneTreeLogLikelihood in AlloppSpeciesBindings for each gene.

getLikelihoodKnown(). Returns false. Always recalculate from scratch.

4.9 Multiply labelled tree AlloppMulLabTree

In evomodel.speciation.

Represents the species network as single binary tree with tips that can be multiply labelled with species. It is reconstructed when the network changes. It is ‘refilled’ with information about coalescences in order to calculate the likelihood Pr(τ|W, θ, γ).

4.9.1 Inner classes

MulLabNode - Contains information about populations size, coalescent times, and a union field which is a set of (species index, sequence index) pairs which identifies the node.

SpSqUnion - low level class used for mapping population values to nodes in AlloppMulLabTree.

PopulationAndLineages - records the information (times, populations, number of lineages) needed to calculate the probability of coalescences in a single branch of the AlloppMulLabTree.

4.9.2 Inner class methods

MulLabNode.tippop(), MulLabNode.hybpop(), MulLabNode.rootpop(). Returns the population value at the tip end of the branch, just after hybridization, or root end of the branch, respectively.

MulLabNode.asText(). For testing

PopulationAndLineages.populationAt(). For calculating the probability of coalescences in a single branch of the multiply labelled tree.

4.9.3 AlloppMulLabTree Methods

AlloppMulLabTree(). Constructor. Passed a 2D array of AlloppLeggedTree’s, the AlloppSpeciesBindings and the population values. Makes a single multiply labelled binary tree from the set of homoploid SimpleTrees. It counts tips, makes array of MulLabNodes. Then it copies the homoploid trees into array using simpletree2mullabtree(), with eg two copies for
tetraploid subtrees, collecting leg info in LegLinks while copying. Then it re-organises the root-pointing LegLinks into tip-pointing FootLinks, and adds nodes to lower-level ploidy subtrees appropriately. Finally it calls fillInUnionsInSubtree().
mullabTreeAsNewick(). Converts the multiply labelled tree to a Newick string, currently just for testing.
asText(). For testing.
clearCoalescences(). Removes coalescent information from nodes. Calls
clearSubtreeCoalescences(). The method for recording coalescences ‘accumulates’ them as they are found in gene trees, so need to remove them all first.
recordLineageCounts(). Fills in counts of lineages at nodes. Calls
recordSubtreeLineageCounts().
coalescenceIsCompatible(). See method of same name in AlloppSpeciesNetworkModel.
recordCoalescence(). See method of same name in AlloppSpeciesNetworkModel.
sortCoalescences(). See method of same name in AlloppSpeciesNetworkModel.
geneTreeInMlTreeLogLikelihood(). Calculates the log-likelihood for a gene tree in the multiply labelled tree. Calls fillInPopVals() and geneTreeInMlSubtreeLogLikelihood().

Private:
makesimpletree(). Converts the multiply labelled tree into a SimpleTree so that it can be logged by standard tree logger.
makesimpleSubtree(). Recursive. For makesimpletree().
subtreeAsText(). Recursive. For asText().
simpleTree2MullabTree(). Recursive. Makes copy of a \verb SimpleTree+, as used by AlloppLeggedTree in array of MulLabNodes. It fills in union fields as it copies.
fillInUnionsInSubtree() fills in union fields after most of MulLabTree is constructed.
nodeOfUnion(). Passed FixedBitSet x, it returns the most tipward node whose union contains x. If x is known to be a union of one of the nodes, it finds that node, so acts as a map from union to node. Calls nodeOfUnionInSubtree().
nodeOfUnionInSubtree(). Recursive, for nodeOfUnion().
mullabSubtreeAsNewick(). Recursive, for mullabTreeAsNewick().
clearSubtreeCoalescences(). Recursive, for clearCoalescences().
recordSubtreeLineageCounts(). Recursive, for recordLineageCounts().
fillInPopVals(). This copies population values in the Parameter popvalues to nodes in the MulLabTree. The population values are per-species-clade (per-branch in network), but of course more than one node in MulLabTree may correspond to the same species. The other complications are that tips are different from internal nodes, and that nodes which roots of tetratrees or just
below, as well as the root are special cases. It collects unions (which represent sets whose elements identify a species and a sequence) from the nodes and then sorts them so that sets of nodes with same species clade are grouped together. This mainly does what is required, since nodes with the same species clade are treated the same. Calls `fillinpopvalsforspunionNoDiploids()` and `fillinpopvalsforspunionTwoDiploids()` to deal with a set of nodes with same species clade.

`fillinpopvalsforspunion()`. For `fillinpopvals()`. Deals with a one union of species indices, that is, for all nodes in the multiply labelled tree which contain a particular group of species in their clade.

`fillinpopvalsforspunionTwoDiploids()`. This copies 0 to 3 population values into a set of nodes with same species clade. Quite complex. In two diptoids case, get one of: (1) one diploid node (tip or root); (2) one foot node, in case where feet meet different diploid branches; (3) two nodes from different tettrees; (4) two nodes which are two feet of tettree meeting one diploid branch; (5) three nodes which are two tetroots and a leg-join. I have written the code (2011-08-12) in the hope it works for more general cases (bigger diploid tree, more than one tetraploid tree) but that is untested.

`fillinpopvalsforspunionNoDiploids()`. Similar to but simpler than previous method. In no diptoids case, either get two nodes from different tettrees or two tetroots and the root.

`geneTreeInMLSubtreeLogLikelihood()`. Recursive, for `geneTreeInMLTreeLogLikelihood()`. Calls `branchLLInMULtreeNoDiploids()` or `branchLLInMULtreeTwoDiploids()`.

`branchLLInMULtreeTwoDiploids()`. Likelihood calculation of gene tree in multiply labelled tree for one branch in the case of no diptoids. This

`branchLLInMULtreeNoDiploids()`. Likelihood calculation of gene tree in multiply labelled tree for one branch in the case of no diptoids. It collects information (coalescent heights, start and end populations, number of lineages) and calls `limbLogLike()`. For some nodes it involve before and after hybridization within branch.

`limbLogLike()`. A 'limb' is part or all of a branch in which the population varies linearly (no hybridization or other jumps). This is used by `geneTreeInMLSubtreeLogLikelihood()`.

`limbLinPopIntegral()`. For `limbLogLike()`.

`union2spseqindex(union)`. Passed union for a tip, hence only containing one (species index, sequence index) pair. It returns the index of that.

Comparators:

`FOOTHEIGHT\_ORDER.compare()`. For `MullabTree()`.

`SPUNION\_ORDER.compare()`. For `fillinpopvals()`. This is quite a complex sort, see `fillinpopvals()`. Its main task is to sort the nodes (defined by clades of (species, sequence) pairs) so that nodes with the same species clades are grouped together. But it also sorts the groups so that tips occur first, and it sorts nodes within groups in a well-defined way that `fillinpopvalsforspunionNoDiploids()` and `fillinpopvalsforspunionTwoDiploids()` rely on. More detail follows.
The comparator sorts the unions of (species, sequence) pairs (\texttt{SpSqUnions}) so that all unions containing the same set of species (ignoring sequence) are grouped together. Call the sets of \texttt{SpSqUnions} for the same species a group. There can be 1, 2 or 3 \texttt{SpSqUnions} in a group.

The groups are sorted in order of increasing number of species (clade size). All groups for a single species (a tip in the network) come first, then those groups for two species, and so on to the root for all species. For groups that have equal numbers of species, a lexicographical ordering using species indices is used.

Within each group, species and sequence information is used to sort the 1 to 3 \texttt{SpSqUnions}. The size of the ‘clade’ of (species, sequence) pairs is used first in the comparison, which ensures that the three nodes with the same species - corresponding to two roots of tetratrees in the multiply-labelled tree plus a leg-join - are ordered so that the two roots come first.

4.10 Prior likelihood calculator \texttt{AlloppNetworkPrior}

In \texttt{evomodel.speciation}.

Computes prior log-likelihood of allopolyploid species network.

4.10.1 Methods

Implements standard simple methods for likelihood. Key ones follow.

\texttt{calculateLogLikelihood()}. Uses \texttt{getSpeciationHeights()} and rate parameter from \texttt{AlloppNetworkPriorModel}. Simple implementation.

\texttt{getLikelihoodKnown()}. Returns false. Always recalculate from scratch.

4.11 Connecting network to gene trees: \texttt{AlloppSpeciesBindings}

In \texttt{evomodel.speciation}.

\texttt{AlloppSpeciesBindings} knows how species are made of individuals and individuals are made of taxa (= diploid genomes within individuals).

It also contains the list of gene trees - tree topologies and node times, plus popfactors. Given a \texttt{AlloppSpeciesNetworkModel} it can say if a gene tree is compatible, and calculate the loglikelihood of the gene tree ‘fitting’ into the network.

It is here that assignments of sequence copies within individuals get permuted. See \texttt{GeneTreeInfo.AlignmentRowInfo} below.
4.11.1 Inner classes

**Individual** - Simple helper class for one individual containing one or more sequences.

**SpeciesIndivPair** - Simple helper class used by `permuteSetOfIndivs()` which is part of sequence reassignment operator.

**ApSpInfo** - Simple helper class for one (allopolyploid) species, containing one or more individuals.

**GeneTreeInfo** - A gene tree as used by BEAST, plus popfactor, plus indices for each individual to map sequences to indices (0 or 1 for tetraploids) which identify the legs of the tetraploid subtree.

**GeneTreeInfo.SequenceAssignment** - where the indices just mentioned are stored.


**GeneTreeInfo.GeneUnionTree** - serves same function as JH’s `CoalInfo`, storing the set (union) of species-sequence pairs belonging to a node in the gene tree. I copy gene tree topology and times into a `GeneUnionTree` to do calculations.

4.11.2 Methods

There are a lot of mapping of one kind of index or indices to others, one to get list of species at a given ploidy level, etc. `fitsInNetwork()` and `geneTreeLogLikelihood()` are key methods.

4.11.3 Methods for small inner classes

**ApSpInfo.taxonFromIndSeq()** Passed indices for individual, sequence, returns taxon.

**GeneTreeInfo.SequenceAssignment.toString().** For header in log file.

**GeneTreeInfo.GeneUnionNode.asText().** For testing.

4.11.4 GeneTreeInfo.GeneUnionTree methods

**GeneUnionTree().** Constructor. Calls `genetree2geneuniontree()` to build the `GeneUnionTree`

private to `GeneUnionTree`:

**subtreeFitsInNetwork()** Recursive. Calls `coalescenceIsCompatible()` in network.

**subtreeRecordCoalescence()** Recursive. Calls `recordCoalescence()` in network.

**genetree2geneuniontree()** Recursive. Copies topology, fills in union fields.

**asText().** For testing. Makes textual rep of `GeneUnionTree`.

**subtreeAsText().** For `asText()`.
4.11.5 GeneTreeInfo methods

GeneTreeInfo(). Constructor. Fills array of AlignmentRowInfo s. Fills int array of lineage counts at tips.

seqassignsAsText(). For testing.

genetreeAsText(). For testing.

fitsInNetwork(). Calls subtreeFitsInNetwork() in GeneUnionTree.

GeneTreeInfo.treeLogLikelihood(). Calls clearCoalescences() in network, makes new GeneUnionTree, calls subtreeRecordCoalescence() in GeneUnionTree, then sortCoalescences() in network, then recordLineageCounts() in network, and finally geneTreeInNetworkLogLikelihood() in network.

storeGeneTreeState(). Stores sequence assignments.

restoreGeneTreeState(). Restores stored sequence assignments.

speciationUpperBound(). Passed two sets of species, it finds the most recent coalescence for this gene such that the children of this gene in the gene tree contain at least one species from each set. Used by AlloppNetworkNodeSlide.

diploidSplitUpperBound(). For one tetra tree case, this finds the most recent coalescence for this gene such that the children of this gene in the gene tree contain at least one from each sequence assignment. Used by AlloppNetworkNodeSlide.

permuteOneSpeciesOneIndiv(). Chooses a random species and a random individual, and calls permuteOneAssignment().

permuteSetOfIndivs(). Chooses a set of (species, individual) pairs, and calls permuteOneAssignment(). 2011-08-12: How it chooses the set is odd. Probably need revisiting when more testing of MCMC efficiency done.

getSeqassigns(). Passed taxon index, it returns (reference to) a SequenceAssignment. Used by logger.

wasChanged(). Does nothing. (Might set dirty flag one day.)

Private:

collectIndivsOfNode(). Used by permuteSetOfIndivs().

subtreeSpeciationUpperBound(). Called by speciationUpperBound() to do the real work.

subtreeDiploidSplitUpperBound(). Called by diploidSplitUpperBound() to do the real work.

permuteOneAssignment(). Passed indices for a species and an individual, to do one ‘flip’ of a sequence assignment.
4.11.6 AlloppSpeciesBindings methods

AlloppSpeciesBindings(). Constructor. Made from array of ApSpInfos, array of TreeModels, array of popFactors, and minheight. Makes 'flattened' arrays of species, individuals, taxa, sets up maps of indices. Makes array of GeneTreeInfos from TreeModels and popFactors, and then fixes the node heights to at least minheight.

There are also two constructors for testing.

initialMinGeneNodeHeight(). Returns what it says. Used for starting state of network.

spsqunion2spunion(). Converts a set (a union) containing (species index, sequence index) pairs into a set containing just species indices.

numberOfGeneTrees(). Returns what it says.

maxGeneTreeHeight(). Returns what it says. Used for Pr(gi|S) calculation in root.

geneTreeFitsInNetwork(). Passed index of a gene tree, it calls fitsInNetwork() in a GeneTreeInfo.

geneTreeLogLikelihood(). Passed index of a gene tree, it calls treeLogLikelihood() in a GeneTreeInfo.

numberOfSpecies(). Returns what it says.

apspeciesName(). Passed index of a species, returns what it says.

SpeciesWithinPloidyLevel(). Passed a ploid level, it returns an array of Taxons for species.

spandseq2spseqindex(). Converts a (species index, sequence index) pair into a single index.

spseqindex2sp(), spseqindex2seq(). Inverse of above, they convert a single index into a (species index, sequence index) pair. They call spseqindex2spandseq().

apspeciesId2index(). Species name to index.

apspeciesId2speciesindiv(). Converts a species id (name like 03\textunderscore d\textunderscore B) to a pair of indices (species,indiv). (Loosely, 03\textunderscore d\textunderscore B \rightarrow (d,03)). used by permuteSetOfIndivs().

numberOfSpSeqs(). Returns number of (species index, sequence index) pairs.

nLineages() Passed index of a species, returns lineage count at tip.

taxonFromSpIndSeq(). Passed three indices, for species, individual, sequence, returns a Taxon.

speciationUpperBound(). Calls method of same name (which see) in each GeneTreeInfo to find minumum.

diploidSplitUpperBound(). Calls method of same name (which see) in each GeneTreeInfo to find minumum.

permuteOneGeneOneSpeciesOneIndiv(). Chooses a random gene and calls permuteOneSpeciesOneIndiv() on it.

permuteSetOfIndivsForOneGene(). Chooses a random gene and calls permuteSetOfIndivs() on it.

seqassignsAsText(). Calls method of same name (which see) in one GeneTreeInfo.
genetreeAsText(). Calls method of same name (which see) in one GeneTreeInfo.

handleModelChangedEvent(Model model, Object object, int index). Calls wasChanged() on each gene tree.

handleVariableChangedEvent(Variable variable, int index, ChangeType type). Never called. (asserts false.)

storeState(). Calls storeGeneTreeState() on each GeneTreeInfo.

restoreState(). Calls restoreGeneTreeState() on each GeneTreeInfo.

acceptState(). Does nothing.

getColumns(). Returns array of LogColumn's for logger, for logging sequence assignments. One column for each (gene,taxon) pair.

Private:

spseqindex2spandseq(). Used by spseqindex2sp(), spseqindex2seq().

4.12 Central class AlloppSpeciesNetworkModel

In evomodel.speciation.

This is the 'main' class for the allopolyploid model. It contains two representations of the network. It implements the species network as a collection of 'trees with legs' and converts this representation into a multiply labelled binary tree. The general idea is that the network is easiest to change (eg detach and re-attach tetraploid subtrees) while likelihood calculations are easiest to do in the multiply labelled tree.

The individual 'trees with legs' are implemented by AlloppLeggedTree's. The representation as a multiply labelled binary tree is implemented by AlloppMulLabTree. It contains a reference to AlloppSpeciesBindings. There is much communication between these classes.

This class implements the Tree interface, although it is really an interface for the multiply labelled tree (AlloppMulLabTree). I think this is necessary because the tree logger (I presume) needs a constant reference whereas the multiply labelled tree comes and goes.

4.12.1 Methods

AlloppSpeciesNetworkModel(). Constructor. Made from an AlloppSpeciesBindings, a popvalue, and a a flag for single hybridization (2011-08-12 this must be true). 2011-08-12, it calls makeInitialOneTetraTreeNetwork() or makeInitialOneTetraTreeTwoDiploidsNetwork() to make a random Yule-type tree with a leg to diploid history, then scales it to be shorter than apsp.initialMinGeneNodeHeight(). It makes a population Parameter of right size with values all equal to popvalue. Then it converts this to a multiply labelled tree, by constructing an AlloppMulLabTree.
There are also two constructors for testing.

getcitations(). Displays citation info (not much yet).

tostring(). For debugging logger. Calls asText() in AlloppMulLabTree and genetreeAsText() and seqassignsAsText() in AlloppSpeciesBindings.

getcolumns(). For debugging logger.

scaleallheights(). Scales all heights by calling method of same name in each AlloppLeggedTree. Used by constructor and by operator.

coalescencisiscompatible(). Passed a gene coalescence height and union. Called from AlloppSpeciesBindings to check if a node in a gene tree is compatible with the network. Calls function of same name in AlloppMulLabTree to do the calculation.

clearcoalescences(). Called from AlloppSpeciesBindings to remove coalescent information from branches of mullabtree. Calls function of same name in AlloppMulLabTree to do the work.

recordcoalescence(). Called from AlloppSpeciesBindings to add a node from a gene tree to its branch in mullabtree. Calls function of same name in AlloppMulLabTree to do the work.

sortcoalescences(). Sorts coalescences within each branch of the multiply labelled tree. Calls function of same name in AlloppMulLabTree to do the work.

recordlineagecounts(). Records the number of gene lineages at nodes of the multiply labelled tree. Calls function of same name in AlloppMulLabTree to do the work.

geneTreeInNetworkLogLikelihood(). Calculates the log-likelihood for a single gene tree in the network. Requires that clearcoalescences(), recordcoalescence(), recordlineagecounts() called to fill mullabtree with information about a particular gene tree’s coalescences first. Calls function of same name in AlloppMulLabTree to do some of the work.

getTippcount(). Returns total number of tips of all AlloppLeggedTree’s. Used by AlloppNetworkPrior.

getEventHeights(). Returns array of ‘event’ heights in the one-tetra-tree case. Used by AlloppNetworkPrior. An ‘event’ is not yet clearly defined. It should include speciations, but not sure what else.

getNumberOfDiTrees(). Returns number of diploid trees.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

getNumberOfNodeHeightsInTree(). Passed ploidy and index, returns the number of node heights in a AlloppLeggedTree in the network.

getHomoploidTree(). Passed ploidy and index, returns a tree in the network.

mullabTreeAsText(). For testing. Calls asText() in AlloppMulLabTree.

beginNetworkEdit(). For operators.

endNetworkEdit(). For operators. Remakes multiply labelled tree and calls fireModelChanged().

getName(). Returns model name.

scale(double scaleFactor, int nDims). Operator. Scales all heights, or scales one node height.
moveLegs(double scaleFactor, int nDims). Operator. Moves the legs of a tetraploid subtree, ie, changes the way it joins the diploid tree.

handleModelChangedEvent(). Calls fireModelChanged().

handleVariableChangedEvent(). Does nothing.

protected void storeState(). Makes copies of all homoploid trees in network.

protected void restoreState() Restores copies of all homoploid trees in network, and remakes multiply labelled tree.

protected void acceptState(). Does nothing.

public Type getUnits()?

public void setUnits(Type units)?

Private:

makeInitialOneTetraTreeNetwork(). For simple case of one tetraploid tree and no diploids. Assumes a history before root of a diploid speciating, the two diploids (or two descendants) forming a hybrid, which speciates at the root of the tetraploid tree.

There is also a version of makeInitialOneTetraTreeNetwork() for testing.

makeInitialOneTetraTreeTwoDiploidsNetwork(). For case of one tetraploid tree and two diploids.

diploidtipbitset(). Passed index of external node in diploid tree, it returns a species-sequence union with one element representing the diploid species at tip. Called by makeInitialOneTetraTreeTwoDiploidsNetwork() and moveLegs().

numberOfPopParameters(). Calculates the number of pop parameters, 2011-08-12 only for no-diploid and two-diploid cases.

Delegations for Tree:

getRoot(), getNodeCount(), getNode(int i), getInternalNode(int i),
getExternalNode(int i), getExternalNodeCount(), getInternalNodeCount(),
geNodeTaxon(NodeRef node), hasNodeHeights(), setNodeHeight(NodeRef node),
hasBranchLengths(), getBranchLength(NodeRef node), getNodeRate(NodeRef node),
geNodeAttribute(NodeRef node, String name), getNodeAttributeNames(NodeRef node),
isExternal(NodeRef node), isRoot(NodeRef node), getChildCount(NodeRef node),
getChild(NodeRef node, int j), getParent(NodeRef node), getCopy(), getTaxonCount(),
getTaxon(int taxonIndex), getTaxonId(int taxonIndex), getTaxonIndex(String id),
geTaxonIndex(Taxon taxon), asList(),
geTaxonAttribute(int taxonIndex, String name), iterator(), getUnits(),
setUnits(Type units), setAttribute(String name, Object value),
getAttribute(String name), Iterator<String> getAttributeNames(),
addTaxon(Taxon taxon), removeTaxon(Taxon taxon),

25
setTaxonId(int taxonIndex, String id),
setTaxonAttribute(int taxonIndex, String name, Object value),
addMutableTaxonListListener(MutableTaxonListListener listener)

Testing:

testExampleNetworkToMulLabTree(). Builds arrangements of trees with legs to test conversion to the multiply labelled tree.
Chapter 5

Implementation of AlloppMUL

5.1 Introduction

Note on implementation 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.

5.2 AlloppMUL formula

1. \( M \) is the multiply labelled tree.
2. \( \theta \) is the population parameters.
3. \( \lambda \) is the parameter(s) for \( M \), that is, the topology and node times. \( \lambda \) could consist of a speciation rate for the Yule model, or birth-death parameters.
4. \( \eta \) is the population mean, appearing in a hyperprior for \( \theta \)
5. \( n \) is the number of gene trees.
6. \( \tau_i \) is the i’th gene tree topology and node times.
7. \( \alpha_i \) 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 \( (\tau_i, \alpha_i) \), that is, all the parameters for the i’th gene tree.
9. \( \gamma_i \) is the permutations 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 \( \alpha, g, \gamma, d \).

\[
\Pr(M, \theta, g, \gamma | d) \propto \Pr(M | \lambda) \Pr(\lambda) \times \\
\Pr(\theta | \eta) \Pr(\eta) \times \\
\Pr(\gamma) \times \\
\prod_i \Pr(\tau_i | M, \theta, \gamma_i) \times \\
\prod_i \Pr(d_i | g_i)
\]

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 omitted.
4. \( \Pr(\tau_i | M, \theta, \gamma_i) \) is the probability of \( \tau_i \), when permuted by \( \gamma_i \), fitting into the multiply labelled tree \( M \) with populations determined by \( \theta \). Note that this probability does not depend on \( \alpha_i \).
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.

### 5.3 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.

### 5.4 List of classes

#### 5.4.1 Parsers in dr.evomodelxml.operators

- MulTreeNodeSlideParser
- MulTreeSequenceReassignmentParser

#### 5.4.2 Parsers in dr.evomodelxml.speciation

- MulMSCoalescentParser
- MulSpeciesBindingsParser
- AlloppNetworkPriorParser
• MulSpeciesTreeModelParser

5.4.3 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

5.4.4 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.

5.5 Details of the major classes

The remaining sections describe the major classes.

5.6 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.

5.7 Operator MulTreeSequenceReassignment

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

5.8 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()`?

5.9 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 implementation of AlloppNET. This is the information used for determining compatibility, 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.

5.9.1 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 utility 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()` constructor, 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()`.

30
handleModelChangedEvent() Not yet clear what this should do.
storeState() stores sequence assignments, deals with other parameters like *BEAST.
restoreState() restores sequence assignments, deals with other parameters like *BEAST.
getColumns() logs sequence assignments.
private to MulSpeciesBindings:
spseqindex2spandseq().

5.10 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.