## Contents

1 Introduction .................................................. 3  
   1.1 Jargon .................................................. 3  
   1.2 Network structure ........................................ 4  
   1.3 AlloppNET formula ........................................ 5  

2 Overall code structure of AlloppNET ....................... 7  
   2.1 List of classes ............................................ 7  
      2.1.1 Parsers in `dr.evomodelxml.operators` ............... 7  
      2.1.2 Parsers in `dr.evomodelxml.speciation` ............... 7  
      2.1.3 Classes in `dr.evomodel.operators` .................... 8  
      2.1.4 Classes in `dr.evomodel.speciation` .................... 8  
      2.1.5 Classes in `dr.evolution.tree` ....................... 8  
      2.1.6 Classes in `test.dr.evomodel.speciation` ............ 9  
      2.1.7 Classes in `dr.util` .................................. 9  
   2.2 Tests ..................................................... 9  
      2.2.1 Test 1: network to MUL-tree ......................... 9  
      2.2.2 Test 2: Likelihood of MUL-tree ...................... 9  

3 Interfaces and classes in AlloppNET .................... 10  
   3.1 Interface `AlloppNode` ................................ 10  
   3.2 Interface `SlidableTree` ................................ 11  
      3.2.1 Interface Methods .................................... 11  
      3.2.2 Methods (static utils) ............................... 11  
   3.3 Operator `AlloppChangeNumHybridizations` .............. 11  
      3.3.1 Methods ............................................. 11  
   3.4 Operator `AlloppHybPopSizesScale` ....................... 12  
   3.5 Operator `AlloppNetworkNodeSlide` ....................... 12  
      3.5.1 Methods ............................................. 12  
   3.6 Operator `AlloppSequenceReassignment` ................. 13  
      3.6.1 Methods ............................................. 13  
   3.7 Diploid history `AlloppDiploidHistory` .................. 13  
      3.7.1 Inner classes ......................................... 14  
      3.7.2 Methods ............................................. 14  
   3.8 Homoploid tree model `AlloppLeggedTree` ............... 15  
      3.8.1 Inner classes ......................................... 15  
      3.8.2 Methods ............................................. 15
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.9</td>
<td>Likelihood calculator AlloppMSCoalescent</td>
<td>16</td>
</tr>
<tr>
<td>3.9.1</td>
<td>Methods</td>
<td>17</td>
</tr>
<tr>
<td>3.10</td>
<td>Multi-labelled tree AlloppMulLabelTree</td>
<td>17</td>
</tr>
<tr>
<td>3.10.1</td>
<td>Inner classes</td>
<td>17</td>
</tr>
<tr>
<td>3.10.2</td>
<td>Inner class methods</td>
<td>17</td>
</tr>
<tr>
<td>3.10.3</td>
<td>AlloppMulLabelTree Methods</td>
<td>18</td>
</tr>
<tr>
<td>3.11</td>
<td>Prior likelihood calculator AlloppNetworkPrior</td>
<td>20</td>
</tr>
<tr>
<td>3.11.1</td>
<td>Methods</td>
<td>20</td>
</tr>
<tr>
<td>3.12</td>
<td>Prior model AlloppNetworkPriorModel</td>
<td>20</td>
</tr>
<tr>
<td>3.13</td>
<td>Connecting network to gene trees: AlloppSpeciesBindings</td>
<td>21</td>
</tr>
<tr>
<td>3.13.1</td>
<td>Inner classes</td>
<td>21</td>
</tr>
<tr>
<td>3.13.2</td>
<td>Methods</td>
<td>21</td>
</tr>
<tr>
<td>3.13.3</td>
<td>Methods for small inner classes</td>
<td>21</td>
</tr>
<tr>
<td>3.13.4</td>
<td>GeneTreeInfo.GeneUnionTree methods</td>
<td>22</td>
</tr>
<tr>
<td>3.13.5</td>
<td>GeneTreeInfo methods</td>
<td>22</td>
</tr>
<tr>
<td>3.13.6</td>
<td>AlloppSpeciesBindings methods</td>
<td>23</td>
</tr>
<tr>
<td>3.14</td>
<td>Central class AlloppSpeciesNetworkModel</td>
<td>24</td>
</tr>
<tr>
<td>3.14.1</td>
<td>Methods</td>
<td>25</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This describes the implementation the AlloppNET model for arbitrary numbers of diploids, tetraploids, and hybridizations.

The code has been changed quite a bit 2012-05-04 to cater for the extensions. Mainly

- The network is divided into diploid and tetraploid parts differently. See section 1.2.
- The ‘move legs’ MCMC move no longer used, but is replaced by changes to diploid history.
- New move to change the number of hybridizations (ie number of tetraploid trees).
- I no longer use SimpleTree to implement the diploid or tetraploid trees.
- I have removed the ‘no diploids’ case.
- In some places I have specialised more to tetraploids (no hexaploids etc).
- In terms of classes, since April/May 2012 AlloppFootLinks, AlloppLegLink, AlloppTreeLeg have disappeared, but AlloppDiploidHistory has become more complicated. AlloppNetworkNodeSlide has become larger and swallowed AlloppMoveLegs. Operators AlloppChangeNumHybridizations and AlloppHybPopSizesScale added.
- Some unit tests no longer work (or aren’t relevant).

1.1 Jargon

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

The species network is composed of homoploid trees, which are joined together. A higher ploidy tree is joined to a lower ploidy tree with one or more legs. The point where a leg meets the branch of the lower ploidy tree is a foot. In the new (Aug 2012) setup, the feet are represented as hyb-tips in the diploid history. See section 1.2.

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 multi-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 multi-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.

1.2 Network structure

![Network diagram](image)

Figure 1.1: Network structure: 4 diploids and 6 tetraploids in three subtrees

[History: For the single hybridization case, I had a simple tree for the diploids with fairly complicated legs in the (single) tetraploid tree specifying which diploid branches it joined to. I now divide the network up with more information in the diploid part, as described next.]

The species network is composed of one or more trees representing tetraploid (sub)trees, and a diploid history. The latter is 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’. From the point of view of a tetraploid tree these hyb-tips are feet at the ends of legs.

See Fig 1.1. Symbols $s_1, s_8$ denote speciation nodes in the diploid history, $r_1, ..., r_3, s_8$ denote speciation nodes in tetraploid subtrees, and $b_1, ..., b_3$ indicate pairs of hybridization tips. $D$ is the diploid history with 3 tips at present and 3 pairs of hyb-tips. There are three tetraploid subtrees $T_1, T_2, T_3$ with 1,2,3 tips respectively. $s_5, s_6, s_7, s_8$ are feet.
1.3 AlloppNET formula

1. \( W \) is the network topology and node times. The multi-labelled tree derived from it is \( M_W \).
2. \( \theta \) is the population parameters. See 2012-08-06-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 scaling factor, 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.

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

\[
\begin{align*}
\Pr(W, \theta, g, \gamma | d) & \propto \Pr(W|\lambda) \Pr(\lambda) \times (1.1) \\
& \times \Pr(\theta|\eta) \Pr(\eta) \times (1.2) \\
& \times \Pr(\gamma) \times (1.3) \\
& \times \prod_i \Pr(\tau_i|M_W, \theta, \gamma_i) \times (1.4) \\
& \times \prod_i \Pr(d_i|g_i) (1.5)
\end{align*}
\]

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. \( \Pr(\tau_i|M_W, \theta, \gamma_i) \) is the probability of \( \tau_i \), when permuted by \( \gamma_i \), fitting into the multi-labelled tree \( M_W \) 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
\text{Pr}(\gamma_i(d_i)|g_i) \text{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 (e.g., 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 multi-labelled tree \( W \) is changed. Thus one can write:

\text{Pr}(\tau_i|W, \theta, \gamma_i) = \text{Pr}(\gamma_i(\tau_i)|W, \theta)

Chapter 2

Overall code structure of AlloppNET

List of classes, and tests.

2.1 List of classes

2.1.1 Parsers in dr.evomodelxml.operators

- AlloppChangeNumHybridizationsParser
- AlloppHybPopSizesScaleParser
- AlloppNetworkNodeSlideParser
- AlloppSequenceReassignmentParser

2.1.2 Parsers in dr.evomodelxml.speciation

- AlloppMSCoalescentParser
- AlloppNetworkPriorModelParser
- AlloppNetworkPriorParser
- AlloppSpeciesBindingsApSpInfoParser
- AlloppSpeciesBindingsIndividualParser
- AlloppSpeciesBindingsParser
- AlloppSpeciesNetworkModelParser
2.1.3 Classes in dr.evomodel.operators

- **AlloppChangeNumHybridizations** An operator which changes the number of tetraploid subtrees by split and merge operations.
- **AlloppHybPopSizesScale** An operator which changes the sizes of populations just after hybridization.
- **AlloppNetworkNodeSlide** An operator which changes node heights and possibly tree topology within a tetraploid tree or diploid history or changes hybridization times.
- **AlloppSequenceReassignment** An operator which changes assignments of sequences within an individual.

2.1.4 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.
- **AlloppLeggedTree** is a homoploid ‘tree with legs’. Used only for tetraploid trees (July 2012).
- **AlloppNode** An interface implemented by DipHistNode in AlloppDiploidHistory, TetraTreeNode in AlloppLeggedTree, 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 instance of Likelihood.
- **AlloppMulLabTree** implements a multi-labelled binary tree. Its nodes can store populations and hybridization times. It made from the tetraploid trees and diploid history.
- **AlloppNetworkPrior** computes log-likelihood of prior for the network. It is an instance of Likelihood.
- **AlloppNetworkPriorModel** stores parameters for network prior (eg rates). It is an instance of Model.
- **AlloppSpeciesBindings** knows how species are made of individuals and individuals are made of taxa (= diploid genomes within individuals). It is an instance of Model.
- **AlloppSpeciesNetworkModel** implements the species network as a collection of ‘tree with legs’, that is, AlloppLeggedTree’s, and converts this representation into a multi-labelled binary tree, that is, an AlloppMulLabTree. It is an instance of Model.

2.1.5 Classes in dr.evolution.tree

- **SlidableTree** An interface implemented by AlloppDiploidHistory, AlloppLeggedTree. It is a minimal tree interface that supports MNL moves.
2.1.6 Classes in test.dr.evomodel.speciation

- AlloppSpeciesNetworkModelTEST is for testing.

2.1.7 Classes in dr.util

- AlloppMisc is odds and ends, mainly for testing.

2.2 Tests

JUnit tests are in 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.

2.2.1 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 AlloppSpeciesNetworkModelTest. That does 3 tetra species, and 2 diploid species, in various arrangements.

2.2.2 Test 2: Likelihood of MUL-tree

Builds a small AlloppMullabTree, with every height, coalescence, nlineages, flags, unions filled in, mostly ‘manually’. Filling population indices and calculation of likelihood is then done as in real usage. The result is compared to that calculated by R code.
Chapter 3

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.

3.1 Interface AlloppNode

In evomodel.speciation.

Implemented by MulLabNode, TetraTreeNode, 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.

String asText(). For debugging output.

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 a static method:

subtreeAsText(). For debugging output.
3.2 Interface SlidableTree

In evolution.tree.

Provides implementation of MNL move. The moves are applied to tetratrees as AlloppLeggedTree s and the DiploidHistory.

3.2.1 Interface Methods

getSlidableRoot(), getSlidableNodeCount(), getSlidableNodeTaxon(),
getSlidableNodeHeight(), setSlidableNodeHeight(), isExternalSlidable(),
getSlidableChild(), replaceSlidableChildren(), replaceSlidableRoot().

3.2.2 Methods (static utils)

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().

3.3 Operator AlloppChangeNumHybridizations

In evomodel.operators.

Operator which changes the number of tetraploid trees. See 2012-09-24-MCMCmoves.pdf for details of the algorithm.

It contains two simple private classes SplitCandidate and MergeCandidate for listing possible moves and choosing one.

3.3.1 Methods

Implements standard simple methods for operator. The key one is doOperation() which calls either doMergeMove() or doSplitMove().

doMergeMove() uses findCandidateMerges() to list possible merges, chooses a random one and calls mergeTettreePair() to carry it out. Calls removeHybPopParam() in AlloppSpeciesNetworkModel. Calculates and returns Hastings ratio.
doSplitMove(). uses findCandidateSplits() to list possible splits, chooses a random one and calls splitTettree() to carry it out. Calls addHybPopParam() in AlloppSpeciesNetworkModel.
Calculates and returns Hastings ratio.

findCandidateMerges(), countCandidateMerges() for doMergeMove(), latter for Hastings ratio.
findCandidateSplits(), countCandidateSplits() for doSplitMove(), latter for Hastings ratio.
pairAreMergeable() checks topology of legs in diploid history agree and meet in the way that is produced by a split move, Done using tettreesShareLegs() in AlloppDiploidHistory.
mergeTettreePair() merges two tetraploid trees by calling a constructor in AlloppLeggedTree, fixes up the links to diploid history, then calls removeFeet() in AlloppDiploidHistory to make a smaller diploid history. See PDF mentioned above for more details.

splitTettree() splits a tetraploid tree. It is passed two nodes which were children of root of old tree and which become the roots of the two new ones. It makes two calls to a constructor in AlloppLeggedTree to make the new trees. It calls addTwoDipTips() in AlloppDiploidHistory to make a larger diploid history. See PDF mentioned above for more details.

### 3.4 Operator AlloppHybPopSizesScale

This is a simple scaling operator which scales a single randomly chosen population size just after hybridization. Can’t use the standard scaleOperator because the number of such population sizes changes.

### 3.5 Operator AlloppNetworkNodeSlide

In evomodel.operators.

Extension of ideas of Mau et al (1999) to deal with allopolyploid networks. It is three operators in one:

- Change a hybridization height
- Slide a node in a tetraploid tree
- Slide a node in a diploid tree

It contains a simple private class NodeHeightInNetIndex for recording a chosen node/move type.

#### 3.5.1 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 the diploid history, a node in a tetraploid subtree or a hybridization event.

operateOneNodeInNet(). Calls randomnode() then one of operateOneNodeInTetraTree() or operateHybridHeight() or operateOneNodeInDiploidHistory().

operateHybridHeight(). Moves the hybridization height to somewhere between root of tree and split height or most recent foot height. This means changing the heights of a pair of hyb-tips in parallel.

operateOneNodeInTetraTree(). Does a MNL move. It avoids moving the root before the hybridization height as well as respecting 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 to the left or right of all diploids, which can mean preventing the root becoming too recent, or another node becoming too ancient. It calls SlidableTree.Utils.mauCanonical() and SlidableTree.Utils.mauReconstruct().

3.6 Operator AlloppSequenceReassignment

In evomodel.operators.

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

3.6.1 Methods

Implements standard simple methods for operator. The key one is doOperation() which does one of three types of move. The first two, permuteOneSpeciesOneIndivForOneGene(), and permuteSetOfIndivsForOneGene() are in AlloppSpeciesBindings and do ‘small’ and ‘big’ changes to sequence assignments within a gene tree. The third swaps a whole tetraploid subtree. It makes multiple calls to flipAssignmentsForAllGenesOneSpecies() in AlloppSpeciesBindings and flipLegsOffTetraTree() in AlloppSpeciesNetworkModel.

3.7 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’. From the point of view of a tetraploid tree these hyb-tips are feet at the ends of legs.
The purpose of this class is to represent the part of the network before hybridizations (=‘diploid history’) in a form (the array of DipHistNode’s) that can be subjected to Mau-type moves.

The diploid history is constructed from a diploid tree and a single tetraploid tree initially. MCMC moves can change the number of tetraploid trees.

### 3.7.1 Inner classes

DipHistNode is a private class implementing AlloppNode. An array of these is used to represent the diploid history as a tree. It has fields for unions, and for linking hyb-tips to an AlloppLeggedTree.

### 3.7.2 Methods

AlloppDiploidHistory(). Main constructor for initial random state. Passed an array of taxa for the diploids, and an AlloppLeggedTree for the tetraploids, 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 a copy constructor.

There is another constructor for testing, which is built form an array of SimpleNodes, a set of tetraploid trees, and an AlloppSpeciesBindings (which is needed for making unions).

getSlidableRoot(), replaceSlidableRoot(), getSlidableNodeCount(), getSlidableNodeHeight(), getSlidableNodeTaxon()+, setSlidableNodeHeight()+, isExternalSlidable(), getSlidableChild(), replaceSlidableChildren(). These are overides for SlidableTree interface.

asText() for debugging.

getInternalNodeCount() for node slide operator.

getDiploidTipCount() used to construct AlloppMulLabTree.

collectFeet() used by move hyb height operator.

tipIsDiploidTip() used by node slide operator for keeping a diploid root.

tettreesShareLegs() for move that merges two tettrees.

intervalOfFoot() is for move that merges two tettrees. This is for Hastings ratio calculation.

removeFeet() is for move that merges two tettrees. This is for after merge.

addTwoDipTips() is for move that splits a tettree. This is for after split.

getAncHeight() is for move that splits a tettree.

getRootIndex(), getHeightFromIndex(), getLftFromIndex(), getRgtFromIndex(), getTaxonFromIndex() are for AlloppMulLabTree constructor.

getRootHeight()

getHybHeight()
collectInternalAndHybHeights() for prior lhood.
moveHybridHeight() for move.
scaleAllHeights() Passed a scaling factor this multiplies all heights (including those of hyb-tips) in the tree. For move and network construction.
clearAllNodeTettree() for tests during merging tetrees.
getNodeTettree() for moving hyb time, and tests during merging tetrees.
getNodeLeg() for node slide in diploid history, for unions, for gene-tree checking.
setNodeTettree() is for move that merges two tettrees.
diphistTreeAsText(), diphistTreeAsUniqueNewick(), diphistTreeAsNodeList(). Output for testing.
diphistOK() for internal consistency checks.

Private:
buildSubtreeFromNodes() for merging and splitting moves, when diploid history size changes.
removeTip() for removeFeet().
makesimpletree(). For testing, used by constructor.
makesimplesubtree(). For makesimpletree().
simplesimpletree2dhtesttree() for unit test construction.

3.8 Homoploid tree model AlloppLeggedTree

In evomodel.speciation.
This is a ‘tree with legs’, which is used for a tetraploid species tree which is attached to a diploid history via its legs.

3.8.1 Inner classes

ALTNode is a private class implementing AlloppNode used to represent a tree, in a way that supports MNL moves. Used for the tetraploid trees, and maybe one day for hexaploids. It has a union field, used for calculations to restrict MCMC moves to be compatible.

3.8.2 Methods

AlloppLeggedTree(). Main constructor for initial random state. Passed an array of Taxons. Makes a random Yule-type tree for them, as an array of ALTNodes. The legs are unset at this point.
There is a copy constructor for store and restore methods.
There is a constructor which makes a merged tree from two trees, for merge move.
There is a constructor which makes a tree from a subtree of a given tree, for split move.
There is also a constructor for testing, which make small nonrandom trees.

getSlidableRoot(), replaceSlidableRoot(), getSlidableNodeCount(),
getSlidableNodeHeight(), getSlidableNodeTaxon()+, setSlidableNodeHeight()+,
isExternalSlidable(), getSlidableChild(), replaceSlidableChildren(). These are
overrides for SlidableTree interface.

asText() for debugging.
leggedtreeOK() for internal consistitency checks.

()  
scaleAllHeights(). Passed a scaling factor this multiplies all heights in the tree. For moves and
 network construction.
fillinTipUnions() for unionOfWholeTetTree() in AlloppSpeciesNetworkModel.

getRootHeight() returns what it says.
getExternalNodeCount(), getInternalNodeCount() return what they say.
collectInternalHeights() returns what it says, for prior likelihood calculation.

setDiphistLftLeg(), setDiphistRgtLeg() used by diploid history constructor and moves to
change legs.

getDiphistLftLeg(), getDiphistRgtLeg() return what they say.

Private:
copySubtree() for ‘merge’ constructor.
noftipsSubtree() for ‘subtree’ constructor.
randomnodeheight() for initial constructor.

3.9 Likelihood calculator AlloppMSCoalescent

In evomodel.speciation.
Computes coalescent log-likelihood of a set of gene trees embedded inside a allopolyploid species
network.
3.9.1 Methods

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

There is also a constructor for testing.

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

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

3.10 Multi-labelled tree AlloppMulLabTree

In evomodel.speciation.

Represents the species network as a 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 $Pr(\tau_i | W, \theta, \gamma_i)$.

It contains the population size parameters. These are three arrays, for tip populations, rootward populations, and hybrid populations.

3.10.1 Inner classes

MulLabNode implements AlloppNode. It contains information (‘node rôle flags’) about the node’s rôle in the network (in a tetraploid tree, the root of one, ancestral to one), populations sizes (via indices), and a union field which is a set of (species index, sequence index) pairs which identifies the node. It has fields for number of lineages, and for coalescent times, which are filled in for one gene at a time during likelihood calculation.

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.

3.10.2 Inner class methods

Apart from AlloppNode implementation, there is:

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.

PopulationAndLineages.populationAt(). For calculating the probability of coalescences in a single branch of the multi-labelled tree.
3.10.3 AlloppMulLabTree Methods

AlloppMulLabTree(). Constructor. Passed a AlloppDiploidHistory and an array of AlloppLeggedTree’s, the AlloppSpeciesBindings and the population values. Makes a single multi-labelled binary tree. It counts tips, makes array of MulLabNodes. It calls 
subtree2MulLabNodes() to construct the topology with taxons and unions filled at tips, and set some node role flags. Then it calls fillinUnionsInSubtree(). Then fillinTetraFlagsInSubtree() completes the job of setting node role flags. Finally it calls makesimpletree() to make tree for output (so that AlloppSpeciesNetworkModel can implement Tree interface).

There is a constructor for testing conversion of diploid history plus tetraploid trees to MUL-tree. 
There is a constructor for testing likelihood calculations. 
testGeneTreeInMULTreeLogLikelihood() for testing.
fillmlnodesforlhoodtest1() for testing.
mullabtreeOK() for internal consistisitency checks.
mullabTreeAsNewick(). Converts the multi-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 multi-labelled tree. Calls fillinpopvals() and geneTreeInMLSubtreeLogLikelihood().

Private:
fillinTetraFlagsInSubtree() for constructor.

subtree2MulLabNodes() for constructor. Visits nodes in diploid history recursively, and when it gets to a hyb-tip, it calls alloppptree2MulLabNodes() to recurse into a tetraploid tree. Each tetraploid tree is visited twice, once for each hyb-tip. It thus builds the MUL-tree topology. Taxons and unions are set at tips. This also sets some tetraancestor flags and all tetraroot flags.
alloppptree2MulLabNodes(). Visits nodes in a tetraploid tree recursively. Taxons and unions are set at tips. This also sets some intetratree flags.
makesimpletree(). Converts the multi-labelled tree into a SimpleTree so that it can be logged by standard tree logger.

makesimplesubtree(). Recursive. For makesimpletree().

subtreeAsText(). Recursive. For asText().

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().

clearSubtreeCoalescences(). Recursive, for clearCoalescences().

recordSubtreeLineageCounts(). Recursive, for recordLineageCounts().

fillinpopvals(). This fills in indices in nodes to index values in the three population arrays. 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. Other complications are that tips are different from internal nodes, and that nodes which roots of tetratrees, ancestors of such roots, as well as the root are treated differently. 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 does some of what is required, since nodes with the same species clade are treated the same. Calls fillinpopvalsforspunion() to deal with a set of nodes with same species clade. See 2012-08-06-popvalues-to-nodes.pdf for more details.

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

siblingOfNode() is simple help function.

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

branchLLInMULtree(). Likelihood calculation of gene tree in multi-labelled tree for one branch in the case of no diploids. This

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.

Comparator:

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 (SpSqUnions) so that all unions containing the same set of species (ignoring sequence) are grouped together. Call the sets of SpSqUnions for the same species a group. There can be 1, 2 or 3 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 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 multi-labelled tree pus a leg-join - are ordered so that the two roots come first.

3.11 Prior likelihood calculator AlloppNetworkPrior

In evomodel.speciation.

Computes prior log-likelihood of allopolyploid species network. This has two main components: for the topology and node times; and for the populations. (Unlike earlier versions of AlloppNET, the prior model on populations are specified as part of the network prior model, since the network needs to get hold of the distribution to generate new values when the number of hybridizations increases. Aug 2012.)

3.11.1 Methods

Implements standard simple methods for likelihood. Key ones follow.

Constructor gets the distribution model for hybrid populations from AlloppNetworkPriorModel and sets this in AlloppSpeciesNetworkModel.

calculateLogLikelihood(). Calls loglikelihoodEvents(), loglikeNumHybridizations() to calculate the prior for the topology and node times. Adds the priors for the population values via the distribution models in AlloppNetworkPriorModel.

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

loglikelihoodEvents() is Yule-like likelihood using times of events.

loglikeNumHybridizations() is experimental adjustment.

3.12 Prior model AlloppNetworkPriorModel

In evomodel.speciation.

Constructed from two one-dimensional parameters, event rate and population scaling factor, and three distribution models, one for each set of population values (tip, rootward, hybrid). It just stores this stuff for AlloppNetworkPrior and AlloppSpeciesNetworkModel.
3.13 Connecting network to gene trees:

AlloppSpeciesBindings

In evomodel.speciation.

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 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 GeneTreeInfo.AlignmentRowInfo below.

3.13.1 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.GeneUnionNode - for GeneUnionTree.

GeneTreeInfo.GeneUnionTree - This is used during calculations fitsInNetwork(), treeLogLikelihood() for one gene tree at a time, then discarded. It serves a similar function to 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.

3.13.2 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.

3.13.3 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.
3.13.4 GeneTreeInfo.GeneUnionTree methods

GeneUnionTree(). Constructor. Calls genetree2geneuniontree() to build the GeneUnionTree.
getRoot(). Does what it says.

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().

3.13.5 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.
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.

storeSequenceAssignments(). Stores sequence assignments.
restoreSequenceAssignments(). Restores stored sequence assignments.

spseqUpperBound(). Passed two sets of species-sequences, 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-sequence from each set. 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().

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

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

flipAssignmentsForSpecies(). Passed index for a species and calls flipOneAssignment() for all individuals.

### 3.13.6 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.

speciesseqEmptyUnion() makes a union of the right size.

taxonseqToTipUnion() makes a union for a tip node, from the taxon.

spsequnion2spunion(). 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.d.B) to a pair of indices (species, indiv). (Loosely, 03.d.B → (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.

spseqUpperBound(). Calls method of same name (which see) in each GeneTreeInfo to find minimum.

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

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

flipAssignmentsForAllGenesOneSpecies(). For all gene trees, this calls flipAssignmentsForSpecies() 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.

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

Private:

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

3.14 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 multi-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 multi-labelled tree.

The individual ‘trees with legs’ are implemented by a AlloppDiploidHistory and one or more AlloppLeggedTree’s. The representation as a multi-labelled binary tree is implemented by AlloppMulLabTree. It contains a reference to AlloppSpeciesBindings. There is much communication between these classes.
This class extends AbstractModel and implements Scalable, Units, Citable, Tree, 
TreeLogger.LogUpon. The Tree interface is really an interface for the multi-labelled tree 
(AlloppMulLabTree). I think this is necessary because the tree logger (I presume) needs a 
constant reference whereas the multi-labelled tree comes and goes.

3.14.1 Methods

AlloppSpeciesNetworkModel(). Constructor. Made from an AlloppSpeciesBindings, and three 
starting values for populations. It gets the taxa, diploids and tetraploids, from the 
AlloppSpeciesBindings and calls makeInitialNDipsNTetsNetwork() to make a random 
network. It then scales it to be shorter than apsp.initialMinGeneNodeHeight() to fit the gene 
trees. It makes three arrays for populations (tips, rootward, hybrid population values). The first 
two are Parameters of the right size, but the hybrid population values are stored in an array of 
doubles because the dimensionality changes. A Parameter is constructed from this as required for 
output. Then it converts the network representation into a multi-labelled tree, by constructing an 
AlloppMulLabTree.

There is also a constructor for testing the conversion of network to multtree.

setHybPopModel(). This is called from AlloppNetworkPrior (which is created after network) to 
supply a ParametricDistributionModel for the prior on the hybrid population values. It 
completes the construction of the network.

getCitations(). Returns citation info for display.

alloppspeciesnetworkOK() for internal consistency checks.

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

The next bunch are AbstractModel overrides.

handleModelChangedEvent(). Calls fireModelChanged().

handleVariableChangedEvent(). Does nothing.

protected void storeState(). Makes copies of all homoploid trees in network. Makes copy of 
hybrid population values.

protected void restoreState() Restores copies of all homoploid trees in network. Restores 
copy of hybrid population values. Remakes multi-labelled tree. Remakes Parameter for hybrid 
population values.

protected void acceptState(). Does nothing.

toString(). For debugging logger. Calls asText() in AlloppDiploidHistory, 
AlloppLeggedTree, AlloppMulLabTree and genetreeAsText() and seqassignsAsText() in 
AlloppSpeciesBindings.

columns(). For debugging logger.

The next bunch are used by MCMC operators (scale() is a MCMC operator).

beginNetworkEdit(). Called before MCMC operation. Currently just debug checks.
endNetworkEdit(). Called after MCMC operation. Remakes Parameter for hybrid population values, remakes multi-labelled tree and calls fireModelChanged().

getName(). Returns model name. (For Scalable implementation.)

scale(). For Scalable implementation, this stretches/squeezes whole network. Calls scaleAllHeights() and scaleAllPopValues().

calculateDipHistTipUnion() finds the union of a tip (diploid or hyb-tip) in the diploid history. Used by move that slides node in diploid history to check gene-tree compatibility.

addHybPopParam(). Adds a new hybrid population value, sampled from the prior distribution set by setHybPopModel(). Returns the logarithm of the prior density, evaluated at the newly created value, for Hastings ratio calculations.

removeHybPopParam(). Removes a hybrid population value. Returns the logarithm of the prior density, evaluated at the value that has disappeared, for Hastings ratio calculations.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

calculateDipHistTipUnion() finds the union of a tip (diploid or hyb-tip) in the diploid history. Used by move that slides node in diploid history to check gene-tree compatibility.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

calculateDipHistTipUnion() finds the union of a tip (diploid or hyb-tip) in the diploid history. Used by move that slides node in diploid history to check gene-tree compatibility.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

calculateDipHistTipUnion() finds the union of a tip (diploid or hyb-tip) in the diploid history. Used by move that slides node in diploid history to check gene-tree compatibility.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

calculateDipHistTipUnion() finds the union of a tip (diploid or hyb-tip) in the diploid history. Used by move that slides node in diploid history to check gene-tree compatibility.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

calculateDipHistTipUnion() finds the union of a tip (diploid or hyb-tip) in the diploid history. Used by move that slides node in diploid history to check gene-tree compatibility.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

calculateDipHistTipUnion() finds the union of a tip (diploid or hyb-tip) in the diploid history. Used by move that slides node in diploid history to check gene-tree compatibility.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

calculateDipHistTipUnion() finds the union of a tip (diploid or hyb-tip) in the diploid history. Used by move that slides node in diploid history to check gene-tree compatibility.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

calculateDipHistTipUnion() finds the union of a tip (diploid or hyb-tip) in the diploid history. Used by move that slides node in diploid history to check gene-tree compatibility.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

getNumberOfTetraTrees(). Returns number of tetraploid trees.

calculateDipHistTipUnion() finds the union of a tip (diploid or hyb-tip) in the diploid history. Used by move that slides node in diploid history to check gene-tree compatibility.

getNumberOfTetraTrees(). Returns number of tetraploid trees.
sortCoalescences(). Sorts coalescences within each branch of the multi-labelled tree. Calls function of same name in AlloppMulLabTree to do the work.

recordLineageCounts(). Records the number of gene lineages at nodes of the multi-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.

Private:

unionOfWholeTetTree() for calculateDipHistTipUnion().

makeInitialNDipsNTetsNetwork(). For constructor of initial random state.

makeLoggingHybPopParam().

scaleAllPopValues(). Stretches or squashes all population values.

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

numberOfTipPopParameters(). Calculates the number of tip pop parameters.

numberOfRootPopParameters(). Calculates the number of rootward pop parameters.

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),
getChild(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)
Testing:

`testExampleNetworkToMulLabTree()` builds arrangements of trees with legs to test conversion to the multi-labelled tree.

`addSimpleNodeChildren()` for test cases.