Graham Jones

2011-06-21

1 Classes

I have added these classes:

`AlloppSpeciesBindingsApSpInfoParser`

`AlloppSpeciesBindingsIndividualParser`

`AlloppSpeciesBindingsParser`

`AlloppSpeciesNetworkModelParser`

`AlloppSpeciesBindings`

`AlloppSpeciesNetworkModel`

`AlloppSpeciesTreeModel (=AlloppLeggedTree)`

`AlloppMSCoalescent`

`AlloppSpeciesNetworkModelTEST`

2 Jargon

I have started using the words **leg**, **foot**, and **union** to mean special things.

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, ... are sequences, then a union can represent a set of elements like {aA, bA, bC}. 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 childrens' 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.

3 XML Parsing

I have edited the TrialAlloppApr18.xml file so it contains bits for the parsers to read.

I have added my parsers to development parsers file.
`beast16/bin/dr/app/beast/development_parsers.properties`

Can parse some XML, reading in info about species, individuals and sequences (=taxa) to make a `AlloppSpeciesBindings`. Can read in XML for `AlloppSpeciesNetworkModel` which just contains a parameter for population sizes.

Parsers are: `AlloppSpeciesBindingsApSpInfoParser`, `AlloppSpeciesBindingsIndividualParser`, `AlloppSpeciesBindingsParser`, `AlloppSpeciesNetworkModelParser`

`AlloppMSCoalescentParser` is needed soon.

I have copied and adapted code from other parsers. I don't have a good understanding of how they work. I have added `getSyntaxRules()` implementations, but it is hard to tell if I have done as much as I should or could.

4 Tests

I have written two JUnit tests. I have decided to create a new (sub)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.

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. This isn't very relevant to the tetra only case, but it will be needed.

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

5 Main things done since 20 May

I have implemented code to test compatibility of gene trees and the species network.

I have implemented code to calculate $\Pr(g_i|S)$, and written R code for the same calculation in a particular case.

I have written R code to generate simulated gene trees from 2 and 3 tetraploid species, and write them to file in a suitable form for Seq-gen to use to simulate alignments. I compiled Seq-gen for Windows and fixed a bug.

6 Main things NOT done yet

There is no interaction with the BEAST MCMC code.

I will need implementations of `operate()`, `accept()`, `reject()` for my `MCMCOperators`, and `getModel()`, `getLogLikelihood()`, `makeDirty()` for `AlloppMSCoalescent`, and `getLogPrior()`.

I need to hook up with the gene tree part of MCMC. Any change in any gene tree requires a recalculation of the $\Pr(g_i|S)$.

I need to implement an equivalent of NodeReheight for the species network. For tetra-only case, it should be very similar to *BEAST, except I will need extra moves for the times in diploid history.

Populations need moves.

A lot of this should be similar to *BEAST, and mainly I need to understand how that works. One thing that is new is an MCMC operator for sequence assignments.

7 `AlloppSpeciesBindings`

`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.

It is here that assignments of sequence copies within individuals get permuted. See `GeneTreeInfo.AlignmentRowInfo` below.

7.1 Inner classes

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

`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.

[Aside: in cases where both parental diploid species are extinct, there is only one leg, so what exactly is identified here? It matters whether two genes are assigned the same index, or a different one but not what the indices are.]

`GeneTreeInfo.AlignmentRowInfo` - where the indices just mentioned are stored.

`GeneTreeInfo.GeneUnionNode` - for `GeneUnionTree`.

`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.

7.2 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.

7.3 `AlloppSpeciesBindings.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.

7.4 `AlloppSpeciesBindings.GeneTreeInfo` methods

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

`fitsInNetwork()`. Calls subtreeFitsInNetwork() in GeneUnionTree.

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

7.5 `AlloppSpeciesBindings` methods

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

`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 $\Pr(g_i|S)$ 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.

`NamesWithinPloidyLevel()`. Passed a ploid level, it returns an array of names,

`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.

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

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

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

TODO:

`handleModelChangedEvent(Model model, Object object, int index)`

`handleVariableChangedEvent(Variable variable, int index, ChangeType type)`

`storeState()`

`restoreState()`

`acceptState()`

private:

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

8 AlloppSpeciesNetworkModel

This contains the `AlloppSpeciesBindings`, and two representations of the network.

Implements the species network as a collection of 'trees with legs' and converts this representation into a multiply labelled binary tree.

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 `AlloppSpeciesTreeModel (=AlloppLeggedTree)`'s.

8.1 Inner classes

`MulLabNode` - for MulLabTree. 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.

`MulLabTree` - represents the species network as single binary tree with tips that can be multiply labelled with species.

`MulLabTree.LegLink, MulLabTree.FootLinks` - for gathering and organising the links between trees of different ploidy, so that the rootward-pointing legs can become tipward-pointing branches.

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

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

8.2 Methods

Lots of key methods here.

8.3 `MulLabTree` methods

`MulLabTree()`. Constructor. Makes a single multiply labelled tree from the set of homoploid `SimpleTree`s. It counts tips, makes array of `MulLabNode`s. Then it copies the homoploid trees into array using `simpletree2mullabtree()`, with eg two copies for tetraploid subtrees, filling the union fields, and collecting leg info in `LegLinks` while copying. Then it re-organise the root-pointing `LegLinks` into tip-pointing `FootLinks`, and adds nodes to lower-level ploidy subtrees appropriately.

`mullabTreeAsNewick()`. Converts `MulLabTree` to a Newick string, currently just for testing.

`nofTips()`. Returns what it says.

`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 coounts of lineages at nodes. Calls `recordSubtreeLineageCounts()`.

`geneTreeInMLTreeLogLikelihood()`. Calculates the log-likelihood for a gene tree in `MulLabTree`. Calls `fillinpopvals()` and `geneTreeInMLSubtreeLogLikelihood()`.

private:

`simpletree2mullabtree()`. Recursive. Makes copy of a `SimpleTree`, as used by `AlloppSpeciesTreeModel (=AlloppLeggedTree)` in array of `MulLabNodes`. It fills in union fields as it copies.

`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 union -¿ node. Calls `nodeOfUnionInSubtree()`.

`nodeOfUnionInSubtree()`. Recursive, for `nodeOfUnion()`.

`mullabSubtreeAsNewick()`. Recursive, for `mullabTreeAsNewick()`.

`clearSubtreeCoalescences()`. Recursive, for `clearCoalescences()`.

`recordSubtreeLineageCounts()`. Recursive, for `recordLineageCounts()`.

`fillinpopvals()`. Copies population values in the `Parameter` popvalues to nodes in the `MulLabTree`. The population values are per-species (per-branch in network), but more than one node in `MulLabTree` may correspond to the same species. The other complications are that tips are different from internal nodes, and the root is a special case.

`fillinpopvalsforspunion()`. For `fillinpopvals()`. Deals with a one union of species indices, that is, for all nodes in `MulLabTree` which contain a particular group of species in thier clade.

`geneTreeInMLSubtreeLogLikelihood()`. Recursive, for `geneTreeInMLTreeLogLikelihood()`. Quite complex: for example it deals with coalecences before and after hybridization events.

`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()`.

Comparators:

`FOOTHEIGHT_ORDER.compare()`. For `MulLabTree()`.

`SPUNION_ORDER.compare()`. For `fillinpopvals()`.

8.4 `AlloppSpeciesNetworkModel` methods

`AlloppSpeciesNetworkModel()`. Constructor. Made from an `AlloppSpeciesBindings` and a popvalue. Currently, it calls `makeInitialOneTetraTreeNetwork()` 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 `MulLabTree`.

There are also two constructors for testing.

`coalescenceIsCompatible()`. Passed a gene coalesence height and union. Called from `AlloppSpeciesBindings` to check if a node in a gene tree is compatible with the network.

`clearCoalescences()`. Called from `AlloppSpeciesBindings` to remove coalescent information from branches of mullabtree.

`recordCoalescence()`. Called from `AlloppSpeciesBindings` to add a node from a gene tree to its branch in mullabtree.

`recordLineageCounts()`. Records the number of gene lineages at nodes of mullabtree.

`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.

TODO:

```
public String getName()
```

```
public int scale(double scaleFactor, int nDims)
```

```
protected void handleModelChangedEvent(Model model, Object object, int index)
```

`protected final void handleVariableChangedEvent(Variable variable, int index, Parameter.ChangeType type)`

`protected void storeState()`

`protected void restoreState()`

`protected void acceptState()`

`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.

`numberOfPopParameters()`. Calculates the number of pop parameters, currently only for tetraploid-only case.

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

Testing:

`testExampleNetworkToMulLabTree()`. Builds arrangements of trees with legs to test conversion to `MulLabTree`.

9 AlloppSpeciesTreeModel (=AlloppLeggedTree)

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 csse there are no legs.

The tree is a SimpleTree. Its nodes contain taxa at tips, and heights. There are several arangements of legs

`NONE`: for the diploid tree.

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

`ONEBRANCH`: Two legs attached to the same branch in a lower ploidy tree at diffeent times.

`JOINED`: One leg, representing two species whcih (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.

9.1 Inner classes

`Leg` - defines attachment to a lower ploidy tree. Uses a union to specify the branch.

9.2 Methods

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

There are also two constructor 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.

`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.

`setLegHalfPloidyLevel()`. Sets the 'half ploidy level' of a specified leg. The half ploidy level is for identifying the tree to which the leg attaches.

`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.

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

`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.

private:

`randomnodeheight()`. Used by constructor.

`randomsplitheight()`. Used by constructor.

9.2.1 Lots of delegations

`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)
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)

9.2.2 TreeTraitProvider

TreeTrait[] getTreeTraits()

getTreeTrait(String key)

9.2.3 MutableTree which extends Tree, MutableTaxonList

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)

handleModelChangedEvent(Model model, Object object, int index)

handleVariableChangedEvent(Variable variable, int index, ChangeType type)

void storeState()

restoreState()

void acceptState()

9.2.4 Scalable

scale(double factor, int nDims)

getName()

9.2.5 TreeLogger.LogUpon

logNow(int state)