Manual for using AlloppNET, AlloppMUL on real data

Graham Jones

2012-02-14, updated 2012-07-13, 2012-10-23

1 The models

The two models are aimed at phylogenetic analysis of diploids and genomic allotetraploids.

- AlloppMUL. Here the multiply labelled tree is inferred directly. The topology, the node times, and the population sizes along the branches are allowed to vary freely, as if the diploid genomes within the allotetraploid(s) were different species. This approach therefore throws away some information implicit in the assumptions. The main advantage of this approach is the simplicity of implementation, since it is a relatively straightforward generalization of the *BEAST model. It can be used when the number of hybridization events is not known.
- AlloppNET. The second model (AlloppNET) is more faithful to evolutionary events, and is more restricted in the scenarios it can deal with. A single hybridization is modelled explicitly, and from that time, the diploid genomes within the allotetraploid(s) must share population sizes and speciation events. At 2012-02-14 it is also restricted to dealing with two diploids. When the assumptions it uses are appropriate it is expected to be more accurate (confirmed by simulations).

2 The models in BEAST

The models are available in version 1.7 of BEAST. If you want to use the models as described in 'Statistical Inference of Allopolyploid Networks in the Presence of Incomplete Lineage Sorting', the rest of this section can be ignored.

For future versions of the models, you may need to download and compile the development version of BEAST. You will need:

- Java and a Java development kit (JDK) installed.
- Subversion installed. `http://subversion.apache.org/`
- Ant installed. `http://ant.apache.org/`

2.1 Linux instructions

2.1.1 Check out the code

Type

```
svn checkout http://beast-mcmc.googlecode.com/svn/trunk/ beast16
```

`beast16` names a directory; you can use another name. You should see a long listing of files ending something like

```
A    beast16/examples/incorrect/testOTFPCLikelihood.xml
 U   beast16
Checked out revision 4366.
```

2.1.2 Build BEAST

`cd` to `beast16` and type `ant`. You should see something like this:

```
[gjones@pc158250 beast16]$ ant
Buildfile: build.xml

clean:

init:
     [echo] BEAST: /home/gjones/beast16/build.xml

compile-all:
    [mkdir] Created dir: /home/gjones/beast16/build
    [javac] Compiling 1836 source files to /home/gjones/beast16/build
    [javac] Note: Some input files use or override a deprecated API.
    [javac] Note: Recompile with -Xlint:deprecation for details.
    [javac] Note: Some input files use unchecked or unsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.
     [echo] Successfully complied.

dist-all:
    [mkdir] Created dir: /home/gjones/beast16/build/dist
      [jar] Building jar: /home/gjones/beast16/build/dist/beast.jar
      [jar] Building jar: /home/gjones/beast16/build/dist/beauti.jar

build:

BUILD SUCCESSFUL
Total time: 37 seconds
[gjones@pc158250 beast16]$
```

2.1.3 Copy some 'properties' files from source directory to build directory

Type

```
cp beast16/src/dr/app/beast/*.properties  beast16/build/dr/app/beast/
```

2.2 Windows instructions

You have to do the same steps: download the code, compile it, and run it via a java command. I use TortoiseSVN which is a GUI for Subversion and Eclipse which is an IDE for Java. This is overkill if you just want to run a development version of BEAST, but I have not tried any other way.

2.3 Test BEAST is working

In order to run BEAST you need to execute a java command and supply a 'classpath' which is in this case a list of three places (`.` meaning 'here', `/home/gjones/beast16/build/`, and `/home/gjones/beast16/lib/*`) to tell java where to look for code. Then there is the java class `BeastMain` to run and the XML file `YuleSingleLocus.xml` which is input to BEAST. Under Linux, all as one line:

```
java -classpath '.:/home/gjones/beast16/build/:/home/gjones/beast16/lib/*'
   dr.app.beast.BeastMain beast16/examples/release/starBEAST/YuleSingleLocus.xml
```

Under Windows, the classpath is separated by `;` not `:` and the code is a different place (`bin` not `build`) and file paths use `\` not `/` and the single quotes don't seem to be needed. It might look like this:

```
java -classpath .;C:\workspace\beast16\bin;C:\workspace\beast16\lib/*
   dr.app.beast.BeastMain beast16\examples\release\starBEAST\YuleSingleLocus.xml
```

You should see a normal BEAST run.

3 Preparing data and making BEAST XML

There is no support in BEAUTi for AlloppNET or AlloppMUL. I have written some R code to help. You can get it from `http://www.indriid.com` (see working notes 2012, look for '2012-05-05. AlloppSim (5th version, as used for article)'. You only need the code in directory `AlloppRcode`.

This R code will generate a 'vanilla' BEAST XML file for these models from .nex files and a text file I call a 'taxa table'.

3.1 Taxa table

```
ID                species    individual    genome
Suw_u6364_A       Suw        u6364         A
Suw_u6432_A       Suw        u6432         A
Suw_u7023_A       Suw        u7023         A
Suw_u7587_A       Suw        u7587         A
Suw_u7614_A       Suw        u7614         A
Suw_u7681_A       Suw        u7681         A
Suw_u14594_A      Suw        u14594        A
Suw_w12596_A      Suw        w12596        A
Salsv_a6362_A     Salsv      a6362         A
Salsv_a7049_A     Salsv      a7049         A
Salsv_a7441_A     Salsv      a7441         A
Salsv_a12534_A    Salsv      a12534        A
Salsv_l7485_A     Salsv      l7485         A
Salsv_l12405_A    Salsv      l12405        A
Salsv_s12338_A    Salsv      s12338        A
Salsv_s12339_A    Salsv      s12339        A
Salsv_s12396_A    Salsv      s12396        A
Salsv_v7725_A     Salsv      v7725         A
Salsv_v12211_A    Salsv      v12211        A
Salsv_v12392_A    Salsv      v12392        A
Si_2492_A          Si        2492          A
Si_2492_B          Si        2492          B
Si_7363_A          Si        7363          A
Si_7363_B          Si        7363          B
Si_7608_A          Si        7608          A
Si_7608_B          Si        7608          B
Si_7701_A          Si        7701          A
Si_7701_B          Si        7701          B
```

Items are separated by spaces and must not contain spaces.

There are three putative species here: diploids Suw and Salsv, and allotetraploid Si.

The ID column, together with a locus, gives a unique label for a single sequence. It will become a taxon id in BEAST XML. The labels don't have to have the above format, but they must be unique and different from any of the labels given to species or individuals.

The species column defines which species the ID belongs to, and the individual column defines the individual. The labels for individuals must be unique within species.

All ID's are given a genome label, which must be A or B and should be A for diploids.

3.2 NEX files

Each nex file contains sequences for some or all of the ID's. Labels must match exactly. Example nexus file for one gene, with sequences truncated:

```
#NEXUS
[Data written by write.nexus.data.R, Tue Feb 14 10:56:43 2012]
BEGIN TAXA;
  DIMENSIONS NTAX=18;
  TAXLABELS
    Suw_u6364_A Suw_u6432_A Suw_u7023_A Suw_u7587_A Suw_u7681_A
    Suw_u14594_A Si_7701_A Si_7608_B Si_7608_A Si_7701_B
    Salsv_a7441_A Salsv_a7049_A Salsv_a6362_A Salsv_s12339_A Salsv_s12396_A
    Salsv_s12338_A Salsv_l12405_A Salsv_v12392_A
  ;
END;

BEGIN CHARACTERS;
  DIMENSIONS NCHAR=616;
  FORMAT MISSING=? GAP=- DATATYPE=DNA INTERLEAVE=NO;
  MATRIX
    Suw_u6364_A         -tcttttggtttttgtctctg
    Suw_u6432_A         ---ttttggtttttgtctctg
    Suw_u7023_A         --cttttggtttttgtctctg
    Suw_u7587_A         --cttttggtttttgtctctg
    Suw_u7681_A         ----------------------
    Suw_u14594_A        --------gtttttgtctctg
    Si_7701_A           -tcttttggtttttgtctctg
    Si_7608_B           -tcttttggtttttgtctctg
    Si_7608_A           -tcttttggtttttgtctctg
    Si_7701_B           ttcttttggtttttgtctctg
    Salsv_a7441_A       -tcttttggtttttgtctctg
    Salsv_a7049_A       -tcttttggtttttgtctctg
    Salsv_a6362_A       -tcttttggtttttgtctctg
    Salsv_s12339_A      --ctttggtttttgtctctg
    Salsv_s12396_A      --------gtttttgtctctg
    Salsv_s12338_A      --------gtttttgtctctg
    Salsv_l12405_A      -tcttttggtttttgtctctg
    Salsv_v12392_A      -tcttttggtttttgtctctg
  ;
END;
```

3.3 The R code

The following R code was used to make the BEAST XML file.

The first section is standard. It loads other R code. You will need to change the first line after you have downloaded the code to point to where you have put the `AlloppRcode` directory.

```
setwd("C:/AAA/Programming/Goteborg/AlloppRcode")

library(ape)

source("AlloppSim_5beastxml_toplevel.r")
source("AlloppSim_6beastxml_bits.r")
```

The second section is particular to the data set and analysis to be done.

```
# directory for input and output
data.dpath <- "C:/AAA/Programming/GoteborgReal/Silene/"

# Input data
fpath.taxatable <- "SileneTaxaTable.txt"
alignmentnames <- c("RPA2", "RPB2", "RPD2a", "RPD2b")

# BEAST information
beastXMLfilename <- "Silene.XML"

beast.chain.length <- "1000000"
beast.screen.logevery <- "1000"
beast.params.logevery <- "1000"
beast.gtrees.logevery <- "1000"
beast.multree.logevery <- "1000"
beast.dbugtune.logevery <- "10000"

# BEAST output file names
sampledgtrees.fnamebase <- "sampledgtrees"
sampledmultrees.fname <- "sampledmultrees.txt"
sampledparams.fname <- "sampledparams.txt"
DBUGTUNE.fname <- "DBUGTUNE.txt"
```

The third section is standard, and and makes two BEAST XML files, for model AlloppNET, and AlloppMUL.

```
beastNETorMULxmlinfo <- list(
    data.dpath=data.dpath,
    fpath.taxatable=fpath.taxatable,
    alignmentnames=alignmentnames,
    beastXMLfilename=beastXMLfilename,
    beast.chain.length=beast.chain.length,
    beast.screen.logevery=beast.screen.logevery,
    beast.params.logevery=beast.params.logevery,
    beast.gtrees.logevery=beast.gtrees.logevery,
    beast.multree.logevery=beast.multree.logevery,
    beast.dbugtune.logevery=beast.dbugtune.logevery,
    sampledgtrees.fnamebase=sampledgtrees.fnamebase,
    sampledmultrees.fname=sampledmultrees.fname,
    sampledparams.fname=sampledparams.fname,
    DBUGTUNE.fname=DBUGTUNE.fname)

make.beastxml.MULandNET.forRealData(beastNETorMULxmlinfo)
```

Some more details on the second section. The names like 'RPA2' in character vector

`alignmentnames` refer to files like 'RPA2.nex' The name 'Silene.XML' assigned to `beastXMLfilename` will result in XML files called 'NETSilene.XML' and 'MULSilene.XML' for the two models. The name 'sampledgtrees' to `sampledgtrees.fnamebase` will result in file names like 'MULsampledgtrees1.txt' and 'MULsampledgtrees2.txt', for genes 1 and 2 in MULSilene.XML, and similarly for NETSilene.XML. Likewise for the other output file names, except there is only one file, not one per gene, for each analysis.

Here is part of the XML generated, showing the list of taxa, and (truncated) sequences. The last sequence shown is not present in the relevant nex file, so is filled with ???.

```
<!--  Taxa. Each taxon is a sequence (A or B) from an individual  -->
<!--  in a species.  -->
  <taxa id="taxa">
    <taxon id="Suw_u6364_A" />
    <taxon id="Suw_u6432_A" />
    <taxon id="Suw_u7023_A" />
    <taxon id="Suw_u7587_A" />
    <taxon id="Suw_u7614_A" />
    <taxon id="Suw_u7681_A" />
    <taxon id="Suw_u14594_A" />
    <taxon id="Suw_w12596_A" />
    <taxon id="Salsv_a6362_A" />
    <taxon id="Salsv_a7049_A" />
    <taxon id="Salsv_a7441_A" />
    <taxon id="Salsv_a12534_A" />
    <taxon id="Salsv_l7485_A" />
    <taxon id="Salsv_l12405_A" />
    <taxon id="Salsv_s12338_A" />
    <taxon id="Salsv_s12339_A" />
    <taxon id="Salsv_s12396_A" />
    <taxon id="Salsv_v7725_A" />
    <taxon id="Salsv_v12211_A" />
    <taxon id="Salsv_v12392_A" />
    <taxon id="Si_2492_A" />
    <taxon id="Si_2492_B" />
    <taxon id="Si_7363_A" />
    <taxon id="Si_7363_B" />
    <taxon id="Si_7608_A" />
    <taxon id="Si_7608_B" />
    <taxon id="Si_7701_A" />
    <taxon id="Si_7701_B" />
  </taxa>
```

```
<!--  Alignment for gene 1 from nex file RPA2  -->
  <alignment id="alignment1" dataType="nucleotide">
    <sequence>
      <taxon idref="Suw_u6364_A" />
      ----------------------------------
    </sequence>
    <sequence>
      <taxon idref="Suw_u6432_A" />
      CCTGATCTTATTATAAACCCACATGCATTTCCTTC
    </sequence>
    <sequence>
      <taxon idref="Suw_u7023_A" />
      CCTGATCTTATTATAAACCCACATGCATTTCCTTC
    </sequence>
    <sequence>
      <taxon idref="Suw_u7587_A" />
      ????????????????????????????????????
    </sequence>
    ...
```

The following section of XML shows the assignment of sequences to individuals and individuals to species, and connects the gene trees to the multiply-labelled species tree. The AlloppNET model uses a similar XML element called `alloppSpecies`.

```
<!--  Multiply-labelled species tree model.  -->
<!--  Assignments of sequences to individuals and individuals to species.  -->
<mulSpecies id="mulSpecies" >
  <apsp id="Suw" ploidylevel="2" >
    <individual id="Suwu6364" >
      <taxon idref="Suw_u6364_A" />
    </individual>
    ...
    <individual id="Suww12596" >
      <taxon idref="Suw_w12596_A" />
    </individual>
  </apsp>
  <apsp id="Salsv" ploidylevel="2" >
    <individual id="Salsva6362" >
      <taxon idref="Salsv_a6362_A" />
    </individual>
    ...
    <individual id="Salsvv12392" >
      <taxon idref="Salsv_v12392_A" />
    </individual>
  </apsp>
  <apsp id="Si" ploidylevel="4" >
    <individual id="Si2492" >
      <taxon idref="Si_2492_A" />
      <taxon idref="Si_2492_B" />
    </individual>
    <individual id="Si7363" >
      <taxon idref="Si_7363_A" />
      <taxon idref="Si_7363_B" />
    </individual>
    <individual id="Si7608" >
      <taxon idref="Si_7608_A" />
      <taxon idref="Si_7608_B" />
    </individual>
    <individual id="Si7701" >
      <taxon idref="Si_7701_A" />
      <taxon idref="Si_7701_B" />
    </individual>
  </apsp>
    <geneTrees id="geneTrees" >
      <treeModel idref="1.treeModel" />
      <treeModel idref="2.treeModel" />
      <treeModel idref="3.treeModel" />
      <treeModel idref="4.treeModel" />
    </geneTrees>
</mulSpecies>
```

3.4 Alleles and homeologs

The XML format was designed with a particular 'shape' of data in mind. It was assumed that there would be one sequence from each diploid individual, and two homeologous sequences from

each tetraploid individual, possibly with some missing data. However you may have more alleles than this for some individuals, or it may be unclear if two sequences from a single individual are different because they are different homeologs, or different due to heterozygosity. You should not take the meaning of the word 'individual' in the XML too literally: the purpose of the XML element is to identify pairs of homeologs in tetraploid individuals. As long as the 'individual's in the XML identify such pairs, it does not matter if they do not correspond to the normal meaning of an individual.

Example 1. If you have two sequences from a diploid individual, which you believe are due to heterozygosity, you can put both sequences in as two individuals. Likewise, if you have four sequences from a tetraploid individual, and you can identify two pairs of homeologs, you can put both pairs of homeologous sequences in as two 'individual's: each of these 'individuals' will have one 'A' and one 'B' in the taxa table.

Example 2. If you don't know if two alleles are homeologs, you could put them in as separate individuals, and add missing data to represent the other homeolog for each of them. This will give the program less information to use than the case when homeologs can be identified.