Manual for using AlloppNET for diploids and allotetraploids

Graham Jones

2013-07-30

1 The model

The model, AlloppNET, is aimed at phylogenetic analysis of diploids and genomic allotetraploids. It is derived from the model in [2] but the new version is able to deal with arbitrary numbers of diploid species, allotetraploid species and hybridizations. It allows you to infer a multi-labeled species tree in an analogous way to *BEAST [1].

The model is available in the development version of (1.8.0pre) of BEAST (not BEAST 2). Until it is incorporated into the release version of BEAST (v1.8 probably) you need to download and compile the development version of BEAST. (Section 3.)

2 Preparing data and making BEAST XML

There is no support in BEAUTi for AlloppNET. I have written some R code to help. You can get it from `http://www.indriid.com` (see working notes 2013). This R code will generate a 'vanilla' BEAST XML file for the model from .nex files and a text file I call a 'taxa table'.

The examples here use data from the genus *Silene* (Caryophyllaceae). The RPD2 data has been published in [3]. The RPA2 data is about to be published in [4]. The RPB2 data was supplied by Anna Petri and has not yet been published.

The file `Silene2-makebeastxml.r` takes as input the four nex files `RPA2.nex`, `RPB2.nex`, `RPD2a.nex`, `RPD2b.nex`, and the taxa table `Silene2TaxaTable.txt` and produces a BEAST XML file

2.1 Taxa table

ID	species	individual	genome
Suw_u6364_A	Suw	u6364	A
Suw_u6432_A	Suw	u6432	A
Suw_u7587_A	Suw	u7587	A
Suw_u7614_A	Suw	u7614	A
Suw_u7681_A	Suw	u7681	A
Suw_u7023_A	Suw	u7023	A
Suw_u14348_A	Suw	u14348	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_a704902134_A	Salsv	a704902134	A
Salsv_a1253402121_A	Salsv	a1253402121	A
Salsv_a1253402142_A	Salsv	a1253402142	A
Salsv_a7441_A	Salsv	a7441	A
Salsv_a74412_A	Salsv	a74412	A
Salsv_a74413_A	Salsv	a74413	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
Ss_6678_A	Ss	6678	A
Ss_6678_B	Ss	6678	B
Ss_6820_A	Ss	6820	A
Ss_6820_B	Ss	6820	B
St_6832_A	St	6832	A
St_6832_B	St	6832	B
St_6834_A	St	6834	A
St_6834_B	St	6834	B

Items are separated by spaces and must not contain spaces.

There are five putative species here: diploids Suw and Salsv, and allotetraploids Si, Ss, St.

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. These A's and B's identify homeologous sequences within individuals. The labels themselves don't have biological meaning.

2.2 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 Oct 09 15:39:02 2012]
BEGIN TAXA;
  DIMENSIONS NTAX=28;
  TAXLABELS
    Si_7701_B Si_7608_B Suw_u6364_A Suw_u6432_A Suw_u7681_A
    Suw_u7023_A Si_7701_A Si_7608_A Salsv_a74412_A Salsv_l7485_A
    Salsv_a74413_A Salsv_s12339_A Ss_6820_A Salsv_a1253402121_A Salsv_v7725_A
    St_6834_A Salsv_s12338_A Salsv_a7049_A Salsv_a1253402142_A St_6832_A
    St_6832_B Salsv_l12405_A Salsv_v12211_A Ss_6820_B St_6834_B
    Ss_6678_A Suw_w12596_A Suw_u14348_A
  ;
END;

BEGIN CHARACTERS;
  DIMENSIONS NCHAR=829;
  FORMAT MISSING=? GAP=- DATATYPE=DNA INTERLEAVE=NO;
  MATRIX
    Si_7701_B               cctgatcttattataaacccacatgcattt
    Si_7608_B               cctgatcttattataaacccacatgcattt
    Suw_u6364_A             -----------------------------
    Suw_u6432_A             cctgatcttattataaacccacatgcattt
    Suw_u7681_A             cctgatcttattataaacccacatgcattt
    Suw_u7023_A             cctgatcttattataaacccacatgcattt
    Si_7701_A               cctgatcttattataaacccacatgcattt
    Si_7608_A               cctgatcttattataaacccacatgcattt
    Salsv_a74412_A          cctgatcttattataaacccgcatgcattt
    Salsv_l7485_A           cctgatcttattataaacccgcatgcattt
    Salsv_a74413_A          cctgatcttattataaacccgcatgcattt
    Salsv_s12339_A          --------------------gcatgcattt
    Ss_6820_A               -----------------------------
    Salsv_a1253402121_A     -----------------------------
    Salsv_v7725_A           -----------------------------
    St_6834_A               cctgatcttattataaacccacatgcattt
    Salsv_s12338_A          cctgatcttattataaacccgcatgcattt
    Salsv_a7049_A           -----------------------------
    Salsv_a1253402142_A     cctgatcttattataaacccgcatgcattt
    St_6832_A               cctgatcttattataaacccgcatgcattt
    St_6832_B               -----------------------------
    Salsv_l12405_A          -----------------------------
    Salsv_v12211_A          cctgatcttattataaacccgcatgcattt
    Ss_6820_B               -----------------------------
    St_6834_B               -----------------------------
    Ss_6678_A               cctgatcttattataaacccgcatgcattt
    Suw_w12596_A            -----------------------------
    Suw_u14348_A            ------------------------------
      ;
END;
```

2.3 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 code.

```
setwd("C:/Users/Graham/AAA/Programming/Goteborg/AlloppRcode-nhybs")

library(ape)

source("AlloppDT_5beastxml_toplevel.r")
source("AlloppDT_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:/Users/Graham/AAA/Programming/GoteborgReal/Silene2/test/"

# Input data
fpath.taxatable <- "Silene2TaxaTable.txt"
alignmentnames <- c("RPA2", "RPB2", "RPD2a", "RPD2b")

# BEAST information
beastXMLfilename <- "Silene2.XML"

beast.chain.length <- "1000000"
beast.screen.logevery <- "10000"
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 the BEAST XML file.

```
beastAlloppDTxmlinfo <- 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.fpathbase=paste(data.dpath, sampledgtrees.fnamebase, sep=""),
    sampledmultrees.fpath=paste(data.dpath, sampledmultrees.fname, sep=""),
    sampledparams.fpath=paste(data.dpath, sampledparams.fname, sep=""),
    DBUGTUNE.fpath=paste(data.dpath, DBUGTUNE.fname, sep=""))


make.beastxml.AlloppDT.forRealData(beastAlloppDTxmlinfo)
```

Some more details on the second section. The names like 'RPA2' in character vector `alignmentnames` refer to files like 'RPA2.nex'. The name 'Silene2.XML' assigned to `beastXMLfilename` will result an XML file called 'Silene2.XML' which you can run in BEAST. The name 'sampledgtrees' to `sampledgtrees.fnamebase` will result in file names like 'sampledgtrees1.txt' and 'sampledgtrees2.txt', for genes 1 and 2 in Silene2.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_u7587_A" />
        ...
```

```
<!--  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" />
CCTGATCTTATTATAAACCCACATGCATTTCC
      </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.

```
<!--  Species network model.  -->
<!--  Assignments of sequences to individuals and individuals to species.  -->
 <alloppSpecies id="alloppSpecies" minGeneNodeHeight="0.2" >
 <apsp id="Suw" ploidylevel="2" >
 <individual id="Suwu6364" >
 <taxon idref="Suw_u6364_A" />
 </individual>
 <individual id="Suwu6432" >
 <taxon idref="Suw_u6432_A" />
 </individual>
 <individual id="Suwu7587" >
 <taxon idref="Suw_u7587_A" />
 </individual>
 <individual id="Suwu7614" >
 <taxon idref="Suw_u7614_A" />
 </individual>
 <individual id="Suwu7681" >
 <taxon idref="Suw_u7681_A" />
 </individual>
 <individual id="Suwu7023" >
 <taxon idref="Suw_u7023_A" />
 </individual>
 <individual id="Suwu14348" >
 <taxon idref="Suw_u14348_A" />
 </individual>
 <individual id="Suwu14594" >
 <taxon idref="Suw_u14594_A" />
 </individual>
 <individual id="Suww12596" >
 <taxon idref="Suw_w12596_A" />
 </individual>
 </apsp>
 <apsp id="Salsv" ploidylevel="2" >
           ...
 </apsp>
 <apsp id="Si" ploidylevel="4" >
           ...
 </apsp>
 <apsp id="Ss" ploidylevel="4" >
           ...
 </apsp>
 <apsp id="St" ploidylevel="4" >
 <individual id="St6832" >
 <taxon idref="St_6832_A" />
 <taxon idref="St_6832_B" />
 </individual>
 <individual id="St6834" >
 <taxon idref="St_6834_A" />
 <taxon idref="St_6834_B" />
 </individual>
 </apsp>
 <geneTrees id="geneTrees" >
 <treeModel idref="1.treeModel" />
 <treeModel idref="2.treeModel" />
 <treeModel idref="3.treeModel" />
 <treeModel idref="4.treeModel" />
 </geneTrees>
 </alloppSpecies>
```

2.4 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 specimens. You may prefer to think of 'individual' as meaning 'homeolog-pair'. 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.

Suppose that some sequences have been obtained for a single gene from a single tetraploid plant, and that no complications such as gene duplication are present. If there were four sequences, you would know that two of the sequences come from one parental diploid species, and two from another. To explain the options, suppose one parental diploid species is labelled 1, the other 2.

If you have one such sequence (call it 'a'), you can make an individual with missing data ('?') for the other homeolog. The program will consider assigning 'a' to 1 and '?' to 2, or vice-versa.

If you have two such sequences ('a', 'b'), you can make one individual if you are prepared to assume that they are homeologs. The program will consider assigning 'a' to 1 and 'b' to 2, or vice-versa. Otherwise, you can make two individuals with two sequences of missing data which will allow 'a', 'b' to be assigned to 1 or 2 independently.

For three or four such sequences, the current implementation does not do as well as it might. Suppose there are four such sequences ('a', 'b', 'c', 'd'). We could write the six possible assignments as 1122, 1212, 1221, 2112, 2121, 2211. Ideally the implementation would use MCMC moves to explore these six possibilities, but that has not yet been implemented. Instead, it allows two options. You may be prepared to divide the four sequences into two homeolog-pairs, say 'a' plus 'b' and 'c' plus 'd'. Then the program will consider 1212, 1221, 2112, 2121 but not 1122 or 2211. If you cannot divide the four sequences into two homeolog-pairs, you can make 4 individuals each with missing data. Then the program will consider all six possibilities, but will also consider ten impossible assignments (1111,1112,...2222). This amounts to throwing away some information (the fact that all four sequences came from one plant).

If you have lots of genes, it is likely that you will have some specimens where you have one sequence for one gene, two sequences which you are sure are homeologs in the case of another gene, two sequences which you are *not* sure are homeologs for a third gene, and so on. You may need to invent lots of individuals to deal with all the possibilities! For example 'specimen1' might become three individuals: 'specimen1knownpair', 'specimen1unknown1', 'specimen1unknown2'.

2.5 Interpreting the results

The MUL-trees can be summarized in the usual way with TreeAnnotator. There are some extra statistics.

There is a statistic "NumHybs" in the "sampledparams" file. This records the number of hybridizations.

The sampled MUL-trees have a label at each node which indicates whether the node is a root of a tetraploid subtree, and if so, which one. The labels are "X" which means "not a root of a tetraploid subtree", and "T0", "T1", "T2", ... for root of different tetraploid subtrees. The labels "T0", "T1", "T2", ... are arbitrary. These may be difficult to interpret - I am not sure how TreeAnnotator summarizes them. It is possible (eg with an R script) to extract the sampled MUL-trees which correspond to a particular partition of the tetraploid species into groups. For example, with tetraploid species "x", "y", "z", you could make a sublist of MUL-trees in which "x" and "z" belong to one tetraploid tree and "y" to another.

3 Downloading and compiling the development version of BEAST

Until it is incorporated into the release version of BEAST you need to download and compile the development version of BEAST. This section may help you, but if it doesn't please ask someone else, not me!

You will need:

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

3.1 Linux instructions

3.1.1 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.
```

3.1.2 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]$
```

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

Type

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

3.2 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.

3.3 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.

References

[1] J Heled and A Drummond. Bayesian inference of species trees from multilocus data. *Mol. Biol. Evol.*, 27:570–580, 2010.

[2] G Jones, S Sagitov, and B Oxelman. Statistical inference of allopolyploid species networks in the presence of incomplete lineage sorting. *Syst. Biol.*, 2013.

[3] A Petri and B Oxelman. Phylogenetic relationships within Silene (Caryophyllaceae) section Physolychnis. *Taxon*, 60: 953–968, 2011.

[4] A Petri, B Pfeil and B Oxelman. Introgressive hybridization between anciently diverged lineages of Silene (Caryophyllaceae). *Submitted to PLOS one*, under revision.