Build B cell lineage trees

Kenneth B. Hoehn

2024-10-21

Contents

Dowser offers multiple ways to build B cell phylogenetic trees. These differ by the method used to estimate tree topology and branch lengths (e.g. maximum parsimony and maximum likelihood) and implementation (IgPhyML, PHYLIP, or R packages ape and phangorn).

Before trees can be built, B cell sequences must be separated into clonal clusters, and had their clonal germline sequences reconstructed. Default settings assume input data is in AIRR TSV format, though column names may be specified using function arguments.

Format clones

Before trees can be built, data must be formatted into a data table of AIRR clone objects. This is accomplished using the formatClones function. This function will:

1. Change non-nucleotide characters to `N` characters.
2. By default, collapse sequences that are either identical or differ only by ambiguous characters.
3. Sequences will not be collapsed if they differ by columns specified in the `traits` option, or if the `collapse` option is set to `FALSE`.
4. Include data columns specified by `num_fields` or `text_fields`.
5. Remove uninformative sequence sites in which all sequences have `N` characters.

The output of this function is a tibble in which each row is a clone, ordered by the number of sequences. The column `data` contains airrClone objects with the clonal sequence alignments. By default, identical sequences will be collapsed into one sequence. This feature can be turned off by setting `collapse` to `FALSE`. Additionally, the `traits` argument can be used to specify other columns that should be considered when identifying identical sequences to collapse. Other columns contain information about the clone, and can be specified using the `columns` argument.

```
library(dowser)

# load example AIRR tsv data
```

```
data(ExampleAirr)

# Subset data for this example
ExampleAirr = ExampleAirr[ExampleAirr$clone_id %in% c("3170", "3184"),]
ExampleAirr$subject_id = "Subject_1"

# Process example data using default settings
clones = formatClones(ExampleAirr)

print(clones)
```

```
## # A tibble: 2 x 4
##   clone_id data       locus  seqs
##      <dbl> <list>     <chr> <int>
## 1     3170 <airrClon> N        13
## 2     3184 <airrClon> N        12
```

```
# Process example data keeping samples from different times
# distinct, adding duplicate_count among collapsed sequences,
# and show the sample_id within each clone in the tibble.
clones = formatClones(ExampleAirr, traits=c("sample_id","c_call"),
    num_fields=c("duplicate_count"), columns=c("subject_id"))

print(clones)
```

```
## # A tibble: 2 x 5
##   clone_id data       locus  seqs subject_id
##      <dbl> <list>     <chr> <int> <chr>
## 1     3170 <airrClon> N        13 Subject_1
## 2     3184 <airrClon> N        12 Subject_1
```

Build maximum parsimony trees

A common way to build B cell lineage trees is the find the tree topology that minimizes the number of mutations needed along the tree (i.e. is the most parsimonious). Branch lengths can then be estimated as the number of mutations per site between each node in the tree.

Maximum parsimony trees can be built with the `getTrees` function, which by default uses the `pratchet` maximum parsimony function in the `phangorn` phylogenetics package.

The output is the same tibble as the input, but with a `trees` column containing an R `ape::phylo` object for each clone.

Maximum parsimony trees using phangorn.

```
clones = getTrees(clones, nproc=1)

print(clones)
## A tibble: 2 x 6
#  clone_id data       locus  seqs subject_id trees
```

```
#     <dbl> <list>     <chr> <int> <chr>      <list>
#1     3170 <airrClon> N        13 Subject_1  <phylo>
#2     3184 <airrClon> N        12 Subject_1  <phylo>
```

Maximum parsimony trees can also be built using the PHYLIP function `dnapars`. To do this, the `build` option needs to be set as `dnapars` and the path to the `dnapars` executable needs to be specified in the `exec` option.

Maximum parsimony trees using dnapars.

```
# exec here is set to dnapars position in the Docker image.
clones = getTrees(clones, build="dnapars", exec="/usr/local/bin/dnapars", nproc=1)

clones
## A tibble: 2 x 6
#  clone_id data       locus  seqs subject_id trees
#     <dbl> <list>     <chr> <int> <chr>      <list>
#1     3170 <airrClon> N        13 Subject_1  <phylo>
#2     3184 <airrClon> N        12 Subject_1  <phylo>
```

Build maximum likelihood trees

A common way to build B cell lineage trees is the find the tree topology and branch lengths that maximize the likelihood of the sequence data given a substitution model.

Standard maximum likelihood trees can also be built with the `getTrees` function, by specifying the build argument as `pml`, which runs the `optim.pml` function in the `phangorn` phylogenetics package.

Maximum likelihood trees can also be built using the PHYLIP function `dnaml`. To do this, the `build` option needs to be set as `dnaml` and the path to the `dnaml` executable needs to be specified in the `exec` option.

Another maximum likelihood option that dowser supports is `RAxML` which utilizes RAxML Next Generation. This option also needs a path to the `RAxML` executable specified in the `exec` option.

Maximum likelihood trees using phangorn.

```
clones = getTrees(clones, build="pml")

print(clones)
## A tibble: 2 x 6
#  clone_id data       locus  seqs subject_id trees
#     <dbl> <list>     <chr> <int> <chr>      <list>
#1     3170 <airrClon> N        13 Subject_1  <phylo>
#2     3184 <airrClon> N        12 Subject_1  <phylo>
```

Maximum likelihood trees using dnaml.

```
# exec here is set to dnaml position in the Docker image.
clones = getTrees(clones, build="dnaml", exec="/usr/local/bin/dnaml")

clones
```

```
# A tibble: 2 x 6
#  clone_id data       locus  seqs subject_id trees
#     <dbl> <list>     <chr> <int> <chr>      <list>
#1     3170 <airrClon> N        13 Subject_1  <phylo>
#2     3184 <airrClon> N        12 Subject_1  <phylo>
```

Maximum likelihood trees using RAxML.

```
# exec here is set to raxml position in the Docker image.
clones = getTrees(clones, build="raxml", exec="/usr/local/bin/raxml-ng")

clones
# A tibble: 2 x 6
#  clone_id data       locus  seqs subject_id trees
#     <dbl> <list>     <chr> <int> <chr>      <list>
#1     3170 <airrClon> N        13 Subject_1  <phylo>
#2     3184 <airrClon> N        12 Subject_1  <phylo>
```

Build IgPhyML B cell trees

B cell somatic hypermutation violates important assumptions in most phylogenetic models. IgPhyML implements models that incorporate SHM hotspot and coldspot motifs. To build trees using IgPhyML, specify the build option appropriately and pass the location of the IgPhyML executable. The returns object will also include a `parameters` column, which will contain the HLP19 model parameters estimated from IgPhyML. Outside of Docker, IgPhyML is only supported for Linux and Mac OS.

Note: This function is slower than other maximum likelihood and parsimony approaches.

```
# exec here is set to IgPhyML position in the Docker image.
clones = getTrees(clones, build="igphyml",
    exec="/usr/local/share/igphyml/src/igphyml", nproc=1)

print(clones)
## A tibble: 2 x 7
#  clone_id data       locus  seqs subject_id trees        parameters
#     <dbl> <list>     <chr> <int> <chr>      <named list> <named list>
#1     3170 <airrClon> N        13 Subject_1  <phylo>      <named list [13]>
#2     3184 <airrClon> N        12 Subject_1  <phylo>      <named list [13]>


clones$parameters[[1]]$omega_mle
#[1] 0.5286
```

Building maximum likelihood trees with multiple partitions

Two of the tree building methods that dowser supports, IgPhyML v2.0.0 and RAxML, support models with multiple partitions. This allows different parameters to be estimated for different regions (partitions) of the sequence data. Both of these methods support "scaled" branch lengths

models. This approach allows different partitions to differ by a scalar factor estimated by maximum likelihood. This is recommended when analyzing single cell data paired heavy and light chains. IgPhyML also allows for separated values of omegas (dN/dS) to be estimated for different partitions. There are many possible models that can be specified, but the most common are detailed below. For full details on different possibilities, see the `buildIgphyml` documentation (all arguments can be passed through getTrees).

Three of the most useful IgPhyML partition models and defaults are listed below:

1. `single`
    - 1 partition for all sequence sites.
2. `cf`
    - 2 partitions: 1 for all CDRs and 1 for all FWRs. By default, a separate omega value is estimated for each. Both use the same branch lengths.
3. `hl`
    - 2 partitions: 1 for the heavy chains and 1 for the light chains. By default, a separate omega value is estimated for each, as well as a separate branch length scalar. This allows heavy and light chains to have proportionally longer or shorter branches. This is only possible if you have paired heavy and light chain sequences and run formatClones with chain="HL" (see Heavy+light chain tree building vignette.)

Building maximum likelihood trees with multiple partitions using the partition = `cf` arugment in IgPhyML.

```
# exec here is set to IgPhyML position in the Docker image.
# Only the newest version of IgPhyML (v2.0.0) supports multi-partition trees
clones = getTrees(clones, build="igphyml",
    exec="/usr/local/share/igphyml/src/igphyml", nproc=1, partition="cf")

print(clones)
## A tibble: 2 x 7
#  clone_id data       locus  seqs subject_id trees        parameters
#     <dbl> <list>     <chr> <int> <chr>      <named list> <named list>
#1     3170 <airrClon> N        13 Subject_1  <phylo>      <named list [13]>
#2     3184 <airrClon> N        12 Subject_1  <phylo>      <named list [13]>
```