A Quick Guide for the phyclust Package

Wei-Chen Chen
Iowa State University

Contents

Acknowledgement

The author thanks Drs. Karin S. Dorman and Ranjan Maitra in Iowa State University for helpful discussion and is a research assistant on grant NSF CAREER DMS-0437555.

1. Introduction

Warning. This document is written to explain the major functions of **phyclust**, version 0.1-4. Every effort will be made to insure future versions are consistent with these instructions, but new features in later versions may not be explained in this document.

This is a quick guide to the package **phyclust**. We will cover how to read and write sequence data, how to use the popular programs `ms` (Hudson 2002) and `seq-gen` (Rambaut and Grassly 1997) for generating coalescent trees and molecular sequences from within **phyclust**, the main function `phyclust()` for finding population structure, and Haplo-Clustering (Tzeng 2005). More information about the theory, other package functions, and any changes in future versions can be found on our website Phylogenetic Clustering at `http://thirteen-01.stat.iastate.edu/snoweye/phyclust/`.

Specifically, in Section 2, we introduce the basic data structures of **phyclust** and the I/O functions for reading and writing PHYLIP and FASTA files. In Section 3, we demonstrate how to simulate molecular data using the "ms+seqgen" approach from within R. In Section 4, we briefly describe the phylogenetic clustering method, its implementation in `phyclust()`, the visualization functions, the auxiliary function `.EMControl()` for choosing the model, initialization method, optimization method, and the EM algorithm variant, and propose a "ms+seqgen+phyclust" approach. In Section 5, we demonstrate the function `haplo.post.prob()` for Hap-Clustering.

1.1. Installation and quick start

You can install **phyclust** directly from CRAN at `http://cran.r-project.org` or by download from our website. On most systems, the easiest installation method is to type the following command into R's terminal:

```
> install.packages("phyclust")
```

When it finishes, you can use `library()` to load the package as

```
> library("phyclust")
```

Note that **phyclust** requires the **ape** package (Paradis *et al.* 2004), and **ape** also requires other packages depending on its version. All the required packages will be checked and automatically loaded when the **phyclust** is loading.

You can get started quickly with **phyclust** by using the `demo()` command in R.

```
> demo("toy", package = "phyclust")
```

This demo will produce the three plots shown in Figures 2, 3 and 4, and some of the results reported in the Section 4.3. The demo executes the commands duplicated below. You can find more information about each command in the sections that follow.

```
### Rename the data and obtain true cluster membership from sequence names.
  X <- seq.data.toy$org
  X.class <- as.numeric(gsub(".*-(.)", "\\1", seq.data.toy$seqname))
### A dot plot, Figure 2.
```

```
  windows()
  plotdots(X, X.class)
### A histogram plot, Figure 3.
  windows()
  plothist(X, X.class)
### A Neighbor-Joining plot, Figure 4.
  ret <- phyclust.edist(X, edist.model = .edist.model[3])
  ret.tree <- nj(ret)
  windows()
  plotnj(ret.tree, X.class = X.class)
### Fit a EE, JC69 model using emEM initialization, Section 4.3
  EMC.2 <- .EMControl(init.procedure = "emEM")
  set.seed(1234)
  (ret.2 <- phyclust(X, 4, EMC = EMC.2))
  RRand(ret.2$class.id, X.class)
```

1.2. Getting help

You can look for more examples on the help pages or our website: `http://thirteen-01.stat.iastate.edu/snoweye/phyclust/`. Also, you can email the author at `phyclust@gmail.com`. All comments are welcome. Bugs will be fixed and suggestions may be implemented in future versions of **phyclust**.

2. Sequence Data Input and Output

Two types of sequences are supported in **phyclust**, nucleotide and SNP. The supported types are stored in `.code.type`:

```
> .code.type
[1] "NUCLEOTIDE" "SNP"
```

Phyclust accepts three types of input:

1. Data read from a text file in PHYLIP format (Section 2.2).
2. Data read from a text file in FASTA format (Section 2.3).
3. Data simulated by the `ms+seqgen` approach (Section 3).

The data reading functions `read.*()` will return a list object of class `seq.data` (Section 2.2). Suppose we call the returned list object `ret`. Then, `ret$org.code` and `ret$org` are two matrices that store the data. Matrix `ret$org.code` contains the original data, e.g. A,G,C,T for nucleotide, and `ret$org` contains the data formatted for the computer, e.g. 0,1,2,3 for nucleotide. Matrix `ret$org` is translated from `ret$org.code` according to the standard encoding (Section 2.1) of the chosen data type and most calculations are done with `ret$org`.

Phyclust outputs sequence data in two formats: PHYLIP or FASTA (Section 2.4).

2.1. Standard coding

Genetic data are represented internally using an integer code, and only the integer values get passed to the C core. The two data frames, `.nucleotide` and `.snp`, are used to map between internal integer code (`nid` or `sid`) and the human interpretable code (`code` or `code.l`).

```
> .nucleotide
  nid code code.l
1   0    A      a
2   1    G      g
3   2    C      c
4   3    T      t
5   4    -      -
> .snp
  sid code
1   0    1
2   1    2
3   2    -
```

Note that we use "-" to indicate gaps and other non general syntax. The methods and functions to deal with gaps are still under development.

2.2. Input PHYLIP format

Some virus data collected from an EIAV-infected pony, #524 (Baccam *et al.* 2003), named "Great pony 524 EIAV rev dataset", are provided as an example of PHYLIP-formatted sequence data. You can view the file with commands

```
> data.path <- paste(.libPaths()[1], "/phyclust/data/pony524.phy", sep = "")
> edit(file = data.path)
```

Below, we show the first 5 sequences and first 50 sites. The first line indicates there are 146 sequences and 405 sites in this file. The sequences are visible starting from the second line, where the first 10 characters are reserved for the sequence name or id.

```
 146 405
AF314258     gatcctcagg gccctctgga aagtgaccag tggtgcaggg tcctccggca
AF314259     gatcctcagg gccctctgga aagtgaccag tggtgcaggg tcctccggca
AF314260     gatcctcagg gccctctgga aagtgaccag tggtgcaggg tcctccggca
AF314261     gatcctcagg gccctctgga aagtgaccag tggtgcaggg tcctccggca
AF314262     gatcctcagg gccctctgga aagtgaccag tggtgcaggg tcctccggca
```

By default, function `read.phylip()` will read in a PHYLIP file and assume the file contains nucleotide sequences. It will read in sequences, translate them using the encoding (Section 2.1), and store them in a list object of class `seq.data`. The following example reads the Pony 524 dataset.

```
> data.path <- paste(.libPaths()[1], "/phyclust/data/pony524.phy", sep = "")
> (my.pony.524 <- read.phylip(data.path))
```

```
code.type: NUCLEOTIDE, n.seq: 146, seq.len: 405.
> str(my.pony.524)
List of 7
 $ code.type: chr "NUCLEOTIDE"
 $ info     : chr " 146 405"
 $ nseq     : num 146
 $ seqlen   : num 405
 $ seqname  : Named chr [1:146] "AF314258" "AF314259" "AF314260" "AF314261" ...
  ..- attr(*, "names")= chr [1:146] "1" "2" "3" "4" ...
 $ org.code : chr [1:146, 1:405] "g" "g" "g" "g" ...
 $ org      : num [1:146, 1:405] 1 1 1 1 1 1 1 1 1 1 ...
 - attr(*, "class")= chr "seq.data"
```

The sample PHYLIP-formatted SNP dataset from a study of Crohn's disease (Hugot *et al.* 2001) can be loaded with the commands

```
> data.path <- paste(.libPaths()[1], "/phyclust/data/crohn.phy", sep = "")
> (my.snp <- read.phylip(data.path, code.type = .code.type[2]))
code.type: SNP, n.seq: 1102, seq.len: 8.
```

Notice, the `code.type` argument must specify the data is of type SNP.

2.3. FASTA format

The sequence data from another pony, #625 (Baccam *et al.* 2003), named "Great pony 625 EIAV rev dataset", is provided in FASTA format. Here is full-length first sequence in that file. It starts with ">" followed by a sequence id and description on the same line. Subsequent lines contain the actualy sequence until the next line starting with ">".

```
>AF512608 Equine infectious anemia virus isolate R93.3/E98.1 gp45 and rev
GATCCTCAGGGCCCTCTGGAAAGTGACCAGTGGTGCAGGGTCCTTCGGCAGTCACTACCT
GAAGAAAAAAATTCCATCGCAAACATGCATCGCGAGAAGACACCTGGGACCAGGCCCAACA
CAACATACACCTAGCAGGCGTGACCGGTGGATCAGGGAACAAATACTACAGGCAGAAGTA
CTCCAGGAACGACTGGAATGGAGAATCAGAGGAGTACAACAGGCGGCCAAAGAGCTGGAT
GAAGTCAATCGAGGCATTTGGAGAGAGCTACATTTCCGAGAAGACCAAAAGGGAGATTTC
TCAGCCTGGGGCGGTTATCAACGAGCACAAGAACGGCACTGGGGGGAACAATCCTCACCA
AGGGTCCTTAGACCTGGAGATTCGAAGCGAAGGAGGAAACATTTAT
>AF512609 Equine infectious anemia virus isolate R93.2/E105 ...
```

By default, function `read.fasta()` will read in a FASTA file and assume the file contains nucleotide sequences. It also returns a list object of class `seq.data`. The following code example reads the pony #625 dataset.

```
> data.path <- paste(.libPaths()[1], "/phyclust/data/pony625.fas", sep = "")
> (my.pony.625 <- read.fasta.nucleotide(data.path))
code.type: NUCLEOTIDE, n.seq: 62, seq.len: 406.
> str(my.pony.625)
List of 6
```

```
 $ code.type: chr "NUCLEOTIDE"
 $ nseq     : num 62
 $ seqlen   : int 406
 $ seqname  : chr [1:62] "AF512608" "AF512609" "AF512610" "AF512611" ...
 $ org.code : chr [1:62, 1:406] "G" "G" "G" "G" ...
 $ org      : num [1:62, 1:406] 1 1 1 1 1 1 1 1 1 1 ...
 - attr(*, "class")= chr "seq.data"
```

2.4. Saving sequences

To save sequences in a file, you can use the functions `write.*()`, which are analogous to the functions `read.*()` but take a data matrix `X` and a file name `filename`. With the following code, we save the two pony datasets in PHYLIP and FASTA formats to the working directory.

```
> # PHYLIp
> write.phylip(my.pony.625$org, "new.625.txt")
> edit(file = "new.625.txt")
> # FASTA
> write.fasta(my.pony.524$org, "new.524.txt")
> edit(file = "new.524.txt")
```

3. The `ms+seqgen` Approach

Phyclust incorporates two popular open source C programs: **ms** (Hudson 2002) and **seq-gen** (Rambaut and Grassly 1997). The original source code and documentation are available on the authors' websites. For **ms**, the pdf file (downloaded from the author's website) is in the install directory `phyclust/doc/Documents/msdoc.pdf` or in the source code directory `phyclust/inst/doc/Documents/msdoc.pdf`. For **seq-gen**, the html file (downloaded from the author's website) is in the install directory `phyclust/doc/Documents/Seq-Gen.v.1.3.2/Seq-Gen.Manual.html` or in the source code directory `phyclust/inst/doc/Documents/Seq-Gen.v.1.3.2/Seq-Gen.Manual.html`.

The **ms** documentation demonstrates how to use **ms** to generate coalescent trees, followed by sequence generation using **seq-gen**, the popular `ms+seqgen` approach for simulating molecular data. I have edited the source code slightly to make these commands available through new R functions `ms()` and `seqgen()`. This solution eases the burden on the user, bypassing the need to compile both programs. Moreover, combining these functions with the `phyclust()` function produces a `ms+seqgen+phyclust` approach for simulation and bootstrap studies (see Section 4.4).

3.1. Using the `ms()` function to generate trees

Almost all command line options of program **ms** are available through option `opts` in `ms()`. Call the function without arguments to see all the options.

```
> ms()
> ?ms
```

The following example generates a coalescent tree (**ms** option `-T`) with 3 leaves (`nsam = 3`) and population growth rate 0.1 (`-G 0.1`). Function `ms()` returns **ms** text output stored in an array, one line per element. The third line contains the tree in NEWICK format, which can be read by the `read.tree()` function in the **ape** package (Paradis *et al.* 2004). Function `read.tree()` returns an object of class `phylo`, which can be drawn by function `plot()` or `plot.phylo()` of the **ape** package.

```
> set.seed(1234)
> (ret.ms <- ms(nsam = 3, opts = "-T -G 0.1"))
ms 3 1 -T -G 0.1
//
(1: 0.568774938583,(2: 0.355949461460,3: 0.355949461460): 0.212825477123);
> (tree.anc <- read.tree(text = ret.ms[3]))

Phylogenetic tree with 3 tips and 2 internal nodes.

Tip labels:
[1] "1" "2" "3"

Rooted; includes branch lengths.
> tree.anc$tip.label <- paste("a", 1:3, sep = "")
> plot(tree.anc, type = "c")
> axisPhylo()
```

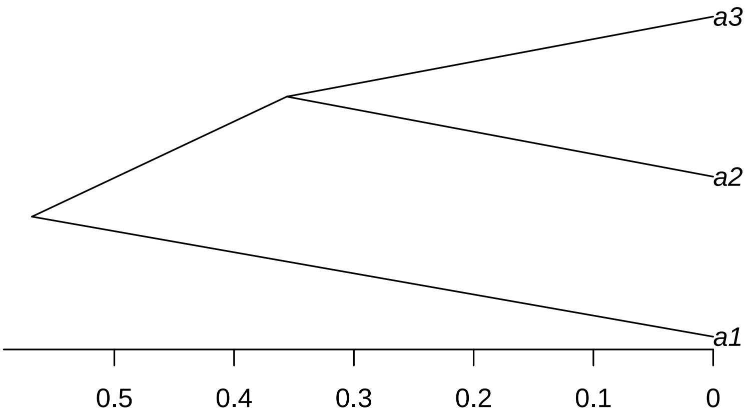

Figure 1: A diagram of a simple coalescent tree.

3.2. Using the seqgen() function to generate sequences

Almost all options of the command line program **seq-gen** are available from within R via the option `opts` in the `seqgen()` function. Call the function without arguments to see the options.

```
> seqgen()
> ?seqgen
```

The `seqgen()` function requires a rooted tree in NEWICK format or an object of class `phylo`. In the following, we demonstrate the `ms+seqgen` approach to generate sequences. The result is a character vector of class `seqgen`, which contains 5 sequences, each of 40 bases (**seq-gen** option `-l40`). The option `-mHKY` specifies the HKY85 model of evolution (Hasegawa *et al.* 1985). Without further options, HKY85 is equivalent to the JC69 model (Jukes and Cantor 1969).

```
> set.seed(123)
> ret.ms <- ms(nsam = 5, nreps = 1, opts = "-T")
> tree.anc <- read.tree(text = ret.ms[3])
> set.seed(123)
> seqgen(opts = "-mHKY -l40", newick.tree = ret.ms[3])
 5 40
1         CTCTCATTGGACGCACACTTTAGGGGGGGATTGCACTGCA
5         CTCTCTCTGGACGCACACTTTAAGGGGGGATTGAACTACA
2         CTCTTCGGGCTCGGATAAGTTTGGAGGGTTGTTCTCTACA
3         CTCTGAGTGCTCGGATTAGTTAGGGGGAATGACGTCTACA
4         CTCTTATCTCTCGGATAAGTTGGGGGTGATGGCTTTTACA
> set.seed(123)
> (ret.seq <- seqgen(opts = "-mHKY -l40", rooted.tree = tree.anc))
 5 40
1         CTCTCATTGGACGCACACTTTAGGGGGGGATTGCACTGCA
5         CTCTCTCTGGACGCACACTTTAAGGGGGGATTGAACTACA
2         CTCTTCGGGCTCGGATAAGTTTGGAGGGTTGTTCTCTACA
3         CTCTGAGTGCTCGGATTAGTTAGGGGGAATGACGTCTACA
4         CTCTTATCTCTCGGATAAGTTGGGGGTGATGGCTTTTACA
> str(ret.seq)
Class 'seqgen'  chr [1:6] " 5 40" "1         CTCTCATTGGACGCACACTTTAGGGGGG ...
```

The `seqgen()` function need not take in a tree from `ms()`, but `ms()` provides options to construct trees of different shapes using coalescent theory. Also, you can provide `seqgen()` with an ancestral sequence that is then evolved along the given tree (see Section 3.3).

3.3. Inputing an ancestral sequence to ms+seqgen

Phyclust provides two functions `gen.seq.HKY()` and `gen.seq.SNP()` to implement the `ms+seqgen` approach under wide-ranging parameter choices. A rooted tree is required and an ancestral sequence is an option.

The following example generates a tree and provides an ancestral sequence. Function `seqgen()` will use parameters κ (`kappa`) and $\pi_A, \pi_G, \pi_C, \pi_T$ (`pi.HKY`) to evolve the ancestral sequence (anc.HKY) down the tree. The function read.seqgen() reads the seqgen() object and returns a new dataset of class seq.data for use by the function phyclust() (see Section 4.2).

```
> # Generate a tree
> set.seed(1234)
> ret.ms <- ms(nsam = 5, nreps = 1, opts = "-T")
> tree.ms <- read.tree(text = ret.ms[3])
>
> # Generate nucleotide sequences
> (anc.HKY <- rep(0:3, 3))
 [1] 0 1 2 3 0 1 2 3 0 1 2 3
> paste(nid2code(anc.HKY, lower.case = FALSE), collapse = "")
[1] "AGCTAGCTAGCT"
> pi.HKY <- c(0.2, 0.2, 0.3, 0.3)
> kappa <- 1.1
> L <- length(anc.HKY)
> set.seed(1234)
> (HKY.1 <- gen.seq.HKY(tree.ms, pi.HKY, kappa, L, anc.seq = anc.HKY))
 5 12
1         AGCTTGACCGGC
3         AGCTTCACCGGT
2         ACCTCGCTAGCT
4         ACGACGCTCGCT
5         CCTACGCTAGCT
> (ret <- read.seqgen(HKY.1))
code.type: NUCLEOTIDE, n.seq: 5, seq.len: 12.
```

Function gen.seq.HKY() may be a good example for advanced users wanting to simulate more complex evolutionary processes, such as recombination, migration and island models. It passes an option input to seqgen(), which in this case is used to pass the ancestral sequence, but could be used to pass other options available in the **seq-gen** program. The option input takes in a character vector (including the tree) where each element contains one **seq-gen** option. Function seqgen() writes these options to a temporary file, which is later communicated to **seq-gen**.

```
### Partial source code of gen.seq.HKY().
        L <- length(anc.seq)
        mu <- paste(nid2code(anc.seq, lower.case = FALSE), collapse = "")
        seqname <- paste("Ancestor  ", collapse = "")
        input <- c(paste(" 1", length(anc.seq), sep = " "), paste(seqname,
            mu, sep = ""), 1, write.tree(rooted.tree, digits = 12))
        opts <- paste("-mHKY", " -t", ts.tv, " -f", paste(pi[c(1,
            3, 2, 4)], collapse = ","), " -l", L, " -s", rate.scale,
            " -u", ttips + 1, " -k1", " -q", sep = "")
        ret <- seqgen(opts, input = input)

### Partial source code of seqgen().
```

```
if (!is.null(newick.tree)) {
    write(newick.tree, file = temp.file.ms, sep = "")
}
else if (!is.null(input)) {
    write(input, file = temp.file.ms, sep = "\n")
}
else {
    stop("A newick or rooted/phylo tree is required.")
}
```

4. Phylogenetic Clustering (Phyloclustering)

Phylogenetic clustering (phyloclustering) is an evolutionary Continuous Time Markov Chain (CTMC) model-based approach to identify population structure from molecular data without assuming linkage equilibrium. Let $\boldsymbol{X} = (x_{nl})_{N\times L}$ be the data matrix containing N sequences observed at L sites. Denote the molecular sequence of individual n as $\boldsymbol{x}_n = (x_{n1}, \ldots, x_{nL}) \in \mathfrak{X}$ and $x_{nl} \in \mathcal{S}$ where $\mathfrak{X}$ contains all possible sequences of length L from alphabet $\mathcal{S}$, e.g. $\mathcal{S} = \{\mathrm{A}, \mathrm{G}, \mathrm{C}, \mathrm{T}\}$ for nucleotide sequences. A finite mixture model provides a statistical framework for clustering. In this setting, each individual sequence $\boldsymbol{x}_n$ is independent and identically drawn from $f(\boldsymbol{x}_n|\boldsymbol{\eta}, \boldsymbol{\Theta}) = \sum_{k=1}^{K} \eta_k f_k(\boldsymbol{x}_n|\Theta_k)$ where $f_k()$ is the density for the kth component, $\boldsymbol{\eta} = \{\eta_1, \ldots, \eta_K\}$ are the mixing proportions summing to one, and $\boldsymbol{\Theta} = \{\Theta_1, \ldots, \Theta_K\}$ contains parameters for the components (Fraley and Raftery 2002). Component $f_k()$ is modeled as a transition probability $p_{\boldsymbol{\mu}_k, \boldsymbol{x}_n}(t_k)$ from a CTMC mutation process (Felsenstein 2004), where a sequence $\boldsymbol{x}_n$ evolves from an ancestor $\boldsymbol{\mu}_k = (\mu_{k1}, \ldots, \mu_{kL}) \in \mathfrak{X}$ representing the kth cluster. The evolutionary process is modeled with instantaneous rate matrix $\boldsymbol{Q}_k$ and time t_k which are allowed to differ by cluster, so that $\Theta_k = \{\boldsymbol{\mu}_k, \boldsymbol{Q}_k, t_k\}$. The likelihood is maximized by an EM algorithm (Dempster *et al.* 1977), sequences are classified by the maximum posterior probabilities, and the number of clusters is assessed by bootstrap (Maitra and Melnykov 2010).

Available choices for the Q_k parameterization in **phyclust** include `JC69` (Jukes and Cantor 1969), `K80` (Kimura 1980), and `HKY85` (Hasegawa *et al.* 1985). These choices are listed in `.substitution`. In addition, Q_k and t_k can be constrained across clusters as shown in Table 1 (also see `.identifier`).

Table 1: Combinations of Models

Identifier	$\boldsymbol{Q}$	t
`EE`	$\boldsymbol{Q}_1 = \boldsymbol{Q}_2 = \cdots = \boldsymbol{Q}_K$	$t_1 = t_2 = \cdots = t_K$
`EV`	$\boldsymbol{Q}_1 = \boldsymbol{Q}_2 = \cdots = \boldsymbol{Q}_K$	$t_1 \neq t_2 \neq \cdots \neq t_K$
`VE`	$\boldsymbol{Q}_1 \neq \boldsymbol{Q}_2 \cdots \neq \boldsymbol{Q}_K$	$t_1 = t_2 = \cdots = t_K$
`VV`	$\boldsymbol{Q}_1 \neq \boldsymbol{Q}_2 \neq \cdots \neq \boldsymbol{Q}_K$	$t_1 \neq t_2 \neq \cdots \neq t_K$

The available initialization methods (`.init.method`) for the EM algorithm use pairwise distances, and the available models for computing the evolutionary distance are listed in `.edist.model`. The model used for computing distances need not match the model used to model evolution in phylogenetic clustering (in `.substitution`). There are additional pairwise distance models available in the **ape** package (Paradis *et al.* 2004).

The `.show.option()` function lists all options available in the **phyclust** package. These options can be used in the `.EMControl()` function to generate an options object (such as `.EMC`), which defines the selected phyloclust model, the initialization method, the EM algorithm and the data type. This argument is passed to function `phyclust()` using argument `EMC`. All options are explained in the help pages. The best choices for options may vary with application. In particular, initialization can be tricky, and you should try several initialization algorithms (see Section 4.3).

```
> .show.option()
boundary method: ADJUST, IGNORE
code type: NUCLEOTIDE, SNP
edist model: D_JC69, D_K80, D_HAMMING
em method: EM, ECM, AECM
identifier: EE, EV, VE, VV
init method: randomMu, NJ, randomNJ, PAM, K-Medoids, manualMu
init procedure: exhaustEM, emEM, RndEM, RndpEM
standard code:
     nid code code.l
[1,]   0    A      a
[2,]   1    G      g
[3,]   2    C      c
[4,]   3    T      t
[5,]   4    -      -
     sid code
[1,]   0    1
[2,]   1    2
[3,]   2    -
substitution model:
          model  code.type
 [1,]      JC69 NUCLEOTIDE
 [2,]       K80 NUCLEOTIDE
 [3,]       F81 NUCLEOTIDE
 [4,]     HKY85 NUCLEOTIDE
 [5,]  SNP_JC69        SNP
 [6,]   SNP_F81        SNP
 [7,]     E_F81 NUCLEOTIDE
 [8,]   E_HKY85 NUCLEOTIDE
 [9,] E_SNP_F81        SNP
```

4.1. Exploring data

Phyclust has functions to help visualize large datasets. We have prepared a simulated dataset (`seq.data.toy`) with 100 nucleotide sequences of length 200 sites from 4 clusters. The ancestral sequences were simulated using the HKY85 model (Hasegawa *et al.* 1985) along a tree of height 0.15 (expected number of mutations per site). The observed sequences were simulated along independent trees with height 0.09 descending from the ancestors.

We use `X` to indicate the data and `X.class` to indicate the classification which can be a result (`class.id`) of the `phyclust()` function or known, in the case of simulation (`ms+seqgen`). The **ms+seq-gen** simulation procedure within **phyclust** produces sequences with names "`sequence.id-class.id`," so the R function `gsub()` (for details, type `?gsub`) can be used to extract the sequence classification (`class id`). In this section, we demonstrate visualization of a simulated dataset, where the clusters are know. Similar figures can be produced for real datasets with clusters estimated by `phyclust()`.

The following code produces the plot of Figure 2. Each row represents a sequence and each column represents a site. By default, it will show all changes with respect to the consensus sequence. If the `X.class` is omitted, then it draws in the original order of data, otherwise in the cluster order.

```
> seq.data.toy
code.type: NUCLEOTIDE, n.seq: 100, seq.len: 200.
> X <- seq.data.toy$org
> X.class <- as.numeric(gsub(".*-(.)", "\\1", seq.data.toy$seqname))
> plotdots(X, X.class)
```

The chosen sequence is fully colored, with green, blue, purple and red representing nucleotides A, G, C, and T. All other sequences show only mutant sites compared to the consensus sequence. The dashed lines separate the clusters. The bottom row indicates the segregating sites, i.e. those sites containing at least one mutation. By default *only* segregating sites are shown. Type `?plotdots` for more information.

Next, we may wish to see how many mutations each sequence has relative to a reference sequence. The following code prepares the plot of Figure 3, showing the number of mutations of all sequences within each cluster relative to the chosen reference sequence. The top plot is for the whole dataset. The other plots are for the four clusters.

```
> plothist(X, X.class)
```

Last, we may like to visualize clusters on a more traditional diagram of evolutionary relationships, the phylogenetic tree. The following code produces Figure 4. The `phyclust.edist()` function takes in a data matrix `X`, computes and returns pairwise distances for all sequences using the Hamming distance as a distance measure (`.edist.model[3]` is `D_HAMMING`). The neighbor-joining method (Saitou and Nei 1987) is used to build a tree from the distance matrix. The function `plotnj()` is a function in **phyclust** for plotting the resulting tree with branches colored according to the clusters defined in argument `X.class`. These clusters may be provided by the user (as is the case here) or as a result of inferring the clusters using `phyclust()`.

```
> (ret <- phyclust.edist(X, edist.model = .edist.model[3]))
Class 'dist'  atomic [1:4950] 4 3 4 7 2 4 5 5 8 2 ...
  ..- attr(*, "Size")= int 100
  ..- attr(*, "Diag")= logi FALSE
  ..- attr(*, "Upper")= logi FALSE
```

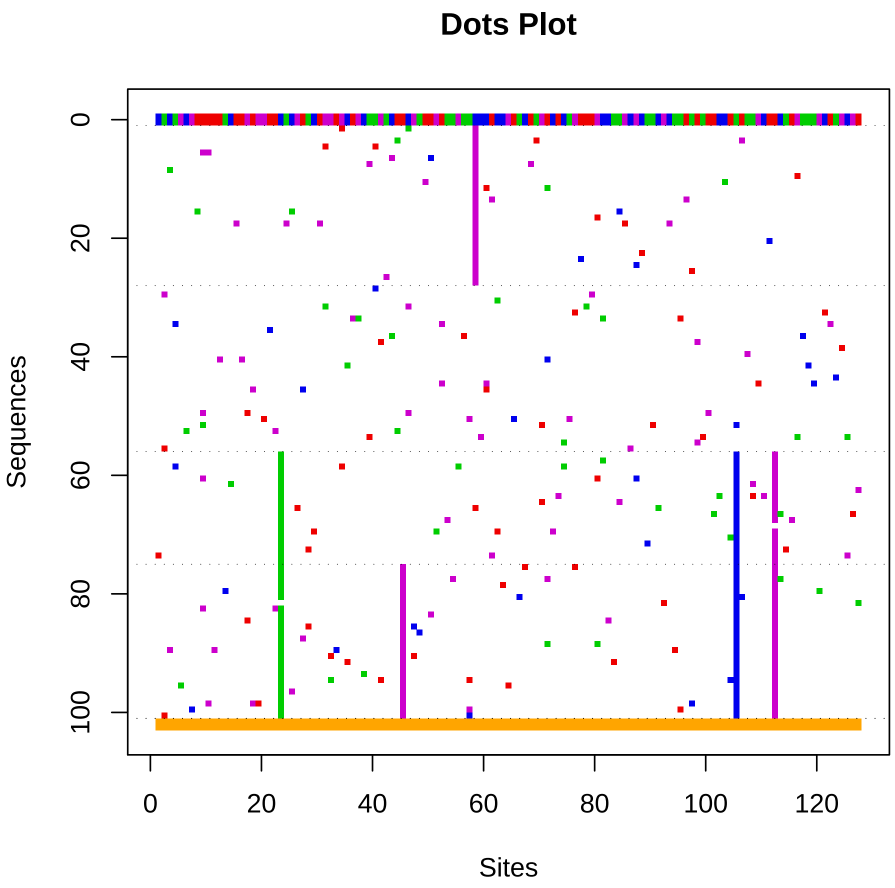

Figure 2: A dot plot for the toy dataset.

```
  ..- attr(*, "method")= chr "D_HAMMING"
> (ret.tree <- nj(ret))

Phylogenetic tree with 100 tips and 98 internal nodes.

Tip labels:
        1, 2, 3, 4, 5, 6, ...

Unrooted; includes branch lengths.
> plotnj(ret.tree, X.class = X.class)
```

4.2. Using the `phyclust()` function

We will use the toy dataset to demonstrate the `phyclust()` function, which requires two arguments: the data matrix `X` and the number of clusters `K`. The optional `EMC` argument of `phyclust()` is used to pass in model and optimization choices. By default, the object `.EMC`

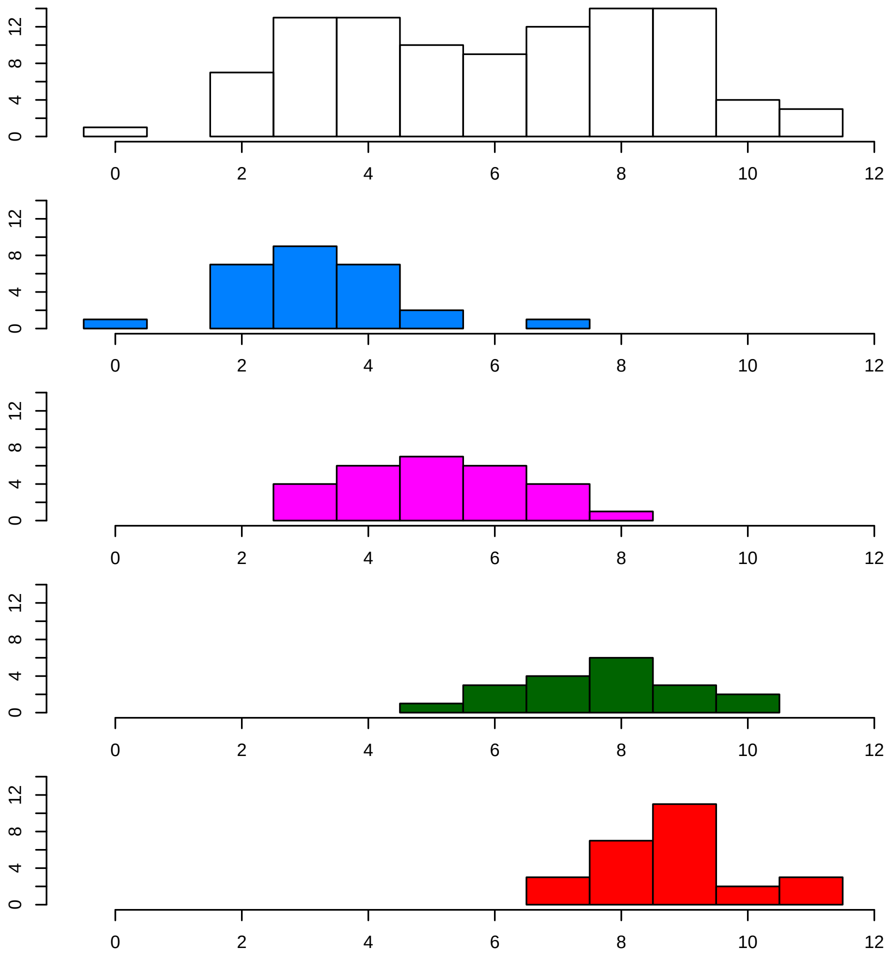

Figure 3: A histogram plot for the toy dataset.

is passed to `phyclust()`. See Section 4.3 for more information about changing the defaults. In the following example, we use the defaults to fit 4 clusters to the toy data.

```
> set.seed(1234)
> (ret.1 <- phyclust(X, 4))
Phyclust Results:
code type: NUCLEOTIDE, em method: EM, boundary method: ADJUST.
init procedure: exhaustEM, method: randomMu.
model substitution: JC69, distance: D_JC69.
iter: 37 3158 0, convergence: 0, check.param: 1.
eps: 4.851e-13, error: 0.
N.X.org: 100, N.X.unique: 87, L: 200, K: 4, p: 804, N.seg.site: 127.
logL: -1439, bic: 6581, aic: 4487, icl: 6588
identifier: EE
  Eta: 0.4360 0.01149 0.284 0.2700
  Tt: 0.003325
```

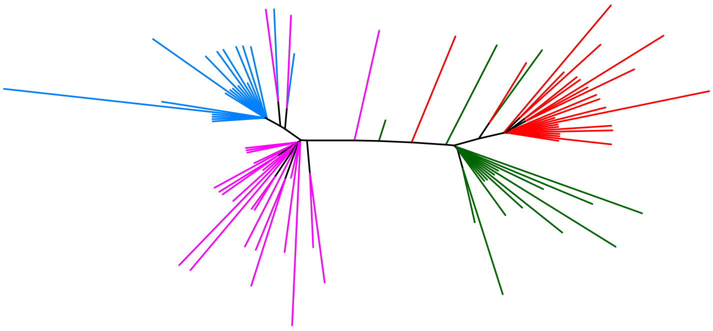

Figure 4: A NJ tree for the toy dataset.

```
  n.class: 44 1 28 27
> RRand(ret.1$class.id, X.class)
   Rand adjRand  Eindex
 0.9018  0.7653  0.1655
> class(ret.1)
[1] "phyclust"
```

The output of the call to `phyclust()` includes

`N.X.org`	the number of sequences
`N.X.unique`	the number of unique/distinct sequences
`L`	for the number of sites
`N.seq.site`	for the number of segregating sites
`K`	the number of clusters
`p`	the number of parameters
`logL`	the maximum likelihood value
`bic`, `aic` and `icl`	for BIC, AIC and ICL
`identifier`	the model choice for $\boldsymbol{Q}_k$ and t_k (see Table 1)
`Eta`	mixing proportions $\boldsymbol{\eta}$
`Tt`	t_k

A quick glance at the results shows that the default settings did not produce good results. There is a degenerate cluster (only one member), as indicated by the count of members in each of the four classes: `n.class`. The default initialization procedure is `exhaustEM` and the default initialization method is `randomMu`, which means it randomly picks 4 sequences to be the cluster centers and runs the EM algorithm to convergence. While the EM algorithm is guaranteed to converge, it may only find a local optimum. It is important to try multiple random initializations to improve your chances of finding the global maximum. The adjusted Rand index (Hubert and Arabie 1985), adjRand, which can be used to compare two clusterings, is about 0.7653 when comparing the phyclust() solution to the true clusters. It should be 1.000 for perfect agreement.

The phyclust() function returns a list object of class phyclust. You can type str(ret.1) to get the details of the phyclust class, or type ?phyclust to see descriptions of all elements. Section 4.3 will give more information for controlling the phyclust() function. This object can also be used as input to other functions, for example bootstrapping and simulations (see Section 4.4).

4.3. Using the .EMControl() function

The .EMControl() function provides a list object that can be used as argument EMC to phyclust(). With no arguments, it returns the default values. The internal object .EMC is a template object holding the defaults. Each element configures some aspect of the evolutionary model, phyloclust model, initialization, optimization, and EM algorithm. See the help page for details and visit our website for examples.

```
> ?.EMControl
> ?.EMC
```

You can either modify the template .EMC directly, or use the function .EMControl() to generate a new control object. The following example modifies an object copied from the template. It changes the initialization method to “emEM,” which yields a better solution than our previous attempt (ret.1). The adjusted Rand index is now 1.000, indicating a perfect match between the truth and inferred structures.

```
> EMC.2 <- .EMC
> EMC.2$init.procedure <- .init.procedure[2]
> ### The same as
> ### EMC.2 <- .EMControl(init.procedure = "emEM")
> set.seed(1234)
> (ret.2 <- phyclust(X, 4, EMC = EMC.2))
Phyclust Results:
code type: NUCLEOTIDE, em method: EM, boundary method: ADJUST.
init procedure: emEM, method: randomMu.
model substitution: JC69, distance: D_JC69.
iter: 103 8725 0, convergence: 0, check.param: 1.
eps: 2.753e-14, error: 0.
N.X.org: 100, N.X.unique: 87, L: 200, K: 4, p: 804, N.seg.site: 127.
logL: -1379, bic: 6461, aic: 4367, icl: 6469
identifier: EE
  Eta: 0.2700 0.1898 0.2801 0.2602
  Tt: 0.003074
  n.class: 27 19 28 26
> RRand(ret.2$class.id, X.class)
```

```
   Rand adjRand  Eindex 
 1.0000  1.0000  0.1209 
```

Now, we use the function `.EMControl()` to generate a new control that uses "RndEM" for initialization. It also changes the phyloclustering model to use the "EV" variant (see Table 1). The data was simulated under "EE" conditions, so this is an over-parameterized model. Such models may also tend to infer degenerate clusters, and again, more initializations may be required for good results. From the output, we observe the mixing proprtion `Eta` of the second cluster is smaller than others. In addition, the evolutionary time `Tt` of this cluster is unusually larger compared to the others. These are both indications of a degenerate cluster, and indeed, no sequences were assigned to this cluster.

```
> EMC.3 <- .EMControl(init.procedure = "RndEM", identifier = "EV")
> ### The same as
> ### EMC.3 <- .EMC
> ### EMC.3$init.procedure <- .init.procedure[3]
> ### EMC.3$identifer <- .identifier[3]
> set.seed(1234)
> (ret.3 <- phyclust(X, 4, EMC = EMC.3))
Phyclust Results:
code type: NUCLEOTIDE, em method: EM, boundary method: ADJUST.
init procedure: RndEM, method: randomMu.
model substitution: JC69, distance: D_JC69.
iter: 104 51836 0, convergence: 0, check.param: 1.
eps: 4.278e-13, error: 0.
N.X.org: 100, N.X.unique: 87, L: 200, K: 4, p: 807, N.seg.site: 127.
logL: -1453, bic: 6621, aic: 4519, icl: 6627
identifier: EV
  Eta: 0.2696 0.01149 0.2844 0.4461
  Tt: 0.002230 4.75 0.003663 0.003924
  n.class: 27 0 28 45
> RRand(ret.3$class.id, X.class)
   Rand adjRand  Eindex 
 0.9002  0.7640  0.1698 
```

There is a convenient function `find.best()` (Section 4.4) that is useful for finding the highest likelihood fit among multiple calls to `phyclust()` with varying arguments. This function runs `phyclust()` repeatedly on combinations of selected initialization options by updating an internal `EMC` control object in each iteration. Please be patient, as this function may take some time to complete.

4.4. The ms+seqgen+phyclust approach

Model selection includes identifying the number of clusters, type of evolutionary model, and phyloclustering assumptions (Table 1). We could use information criteria to choose among models, but the parameter space is mixed continuous and discrete so the theory justifying these critiera does not apply. A more elaborate procedure to assess the number of clusters for a dataset is based on the parametric bootstrap technique and sequential hypothesis testing (Maitra and Melnykov 2010).

The basic idea is to resample datasets from the fitted model using the functions ms() and seqgen(), and refit the resampled dataset by phyclust(). The same fitting method is applied to each dataset, producing a distribution of parameter estimates. The bootstrap.seq.data() function is a tool for this procedure; it takes a fitted model, the output of a previous call to phyclust(). The following example bootstraps the toy dataset once assuming $K = 2$ clusters.

```
> set.seed(1234)
> ret.4 <- phyclust(X, 2)
> (seq.data.toy.new <- bootstrap.seq.data(ret.4))
code.type: NUCLEOTIDE, n.seq: 100, seq.len: 200.
> (ret.4.new <- phyclust(seq.data.toy.new$org, 2))
Phyclust Results:
code type: NUCLEOTIDE, em method: EM, boundary method: ADJUST.
init procedure: exhaustEM, method: randomMu.
model substitution: JC69, distance: D_JC69.
iter: 30 2947 0, convergence: 0, check.param: 1.
eps: 1.41e-14, error: 0.
N.X.org: 100, N.X.unique: 56, L: 200, K: 2, p: 402, N.seg.site: 69.
logL: -685.3, bic: 3222, aic: 2175, icl: 3222
identifier: EE
  Eta: 0.48 0.52
  Tt: 0.001354
  n.class: 48 52
```

Output ret.4 is the result from phyclust(), seq.data.toy.new is a new dataset bootstrapped from the model in ret.4, and ret.4.new is the fit of the bootstrapped dataset. Generally, we need to repeat these steps several times to obtain a distribution of parameter estimates. For example, the following code uses $B = 10$ bootstraps to obtain the distributions of log-likelihood for $K = 1$ and $K = 2$, and we can perform a likelihood ratio test to assess the number of clusters.

```
> ### This code may take a long time to run.
> set.seed(1234)
> ret.K1 <- find.best(X, 1)
> ret.K2 <- find.best(X, 2)
> (logL.diff <- ret.K2$logL - ret.K1$logL)
[1] 663.2635
> set.seed(1234)
> B <- 10
> boot.logL.diff <- NULL
> for(b in 1:B){
>   seq.data.toy.new <- bootstrap.seq.data(ret.K1)
>   ret.H0 <- find.best(seq.data.toy.new$org, 1)
>   ret.H1.H0 <- find.best(seq.data.toy.new$org, 2)
>   boot.logL.diff <- c(boot.logL.diff, ret.H1.H0$logL - ret.H0$logL)
> }
> boot.logL.diff
 [1] 20.17399 24.94449 25.50644 24.30299 25.94984 19.03801 19.20099 19.40565
 [9] 18.88133 37.16275
> sum(logL.diff < boot.logL.diff)
[1] 0
```

Here, the difference of likelihood values `logL.idff` is 663.2635 between models $K = 1$ and $K = 2$. We bootstrap from a fitted model of $K = 1$, and it is 0 out of 10 results indicating the $K = 2$ performs better than $K = 1$. As the B increases, we may more confident to reject $K = 1$.

5. Using the `haplo.post.prob()` function for Hap-Clustering

Haplotype grouping (Tzeng 2005) for SNP datasets can be viewed as an alternative method to phyloclustering. The author's R code has been integrate into **phyclust**, and the original function has been renamed `haplo.post.prob()`. The example used by the author is the Crohn's disease dataset (Hugot *et al.* 2001), which is also included in the **phyclust** package. The original description of the author's R code is in the install directory `phyclust/doc/Documents/tzeng-Readme.txt` or in the source code directory `phyclust/inst/doc/Documents/tzeng-Readme.txt`.

The following example returns the same results as Tzeng (2005), where the predicted number of clusters based on her information criterion is 13. The function returns a list object, here stored in `ret`. The list element `ret$haplo` stores information about the SNP sequences, `ret$FD.id` and `ret$RD.id` store the full and reduced dimensional indices, `ret$FD.post` and `ret$RD.post` store the full and reduced dimensional posterior probabilities, and `g.truncate` shows the number of clusters, the truncated results as described in Tzeng (2005).

```
> data.path <- paste(.libPaths()[1], "/phyclust/data/crohn.phy", sep = "")
> my.snp <- read.phylip.snp(data.path)
> ret <- haplo.post.prob(my.snp$org, ploidy = 1)
> str(ret)
List of 6
 $ haplo     :List of 6
  ..$ haplotype: num [1:39, 1:8] 0 1 1 0 1 1 0 1 1 0 ...
  ..$ hap.prob : num [1:39] 0.00454 0.00181 0.11797 0.00635 0.00635 ...
  ..$ post     : num [1:1102] 1 1 1 1 1 1 1 1 1 1 ...
  ..$ hap1code : int [1:1102] 1 1 1 1 1 2 2 3 3 3 ...
  ..$ hap2code : int [1:1102] 1 1 1 1 1 2 2 3 3 3 ...
  ..$ indx.subj: int [1:1102] 1 2 3 4 5 6 7 8 9 10 ...
 $ FD.id     : int [1:39] 3 9 18 22 27 28 30 31 34 35 ...
 $ RD.id     : int [1:13] 3 9 18 22 27 28 30 31 34 35 ...
 $ FD.post   : num [1:1102, 1:39] 0 0 0 0 0 0 0 1 1 1 ...
 $ RD.post   : num [1:1102, 1:13] 0 0 0 0 0 1 1 1 1 1 ...
 $ g.truncate: int 13
> getcut.fun(sort(ret$haplo$hap.prob, decreasing = TRUE),
>            nn = my.snp$nseq, plot = 1)
```

The `getcut.fun()` produces a plot based on the information criterion, which can be used to visualize the truncated dimension. In Figure 5, the horizontal line indicates the cut point of 13 haplotypes.

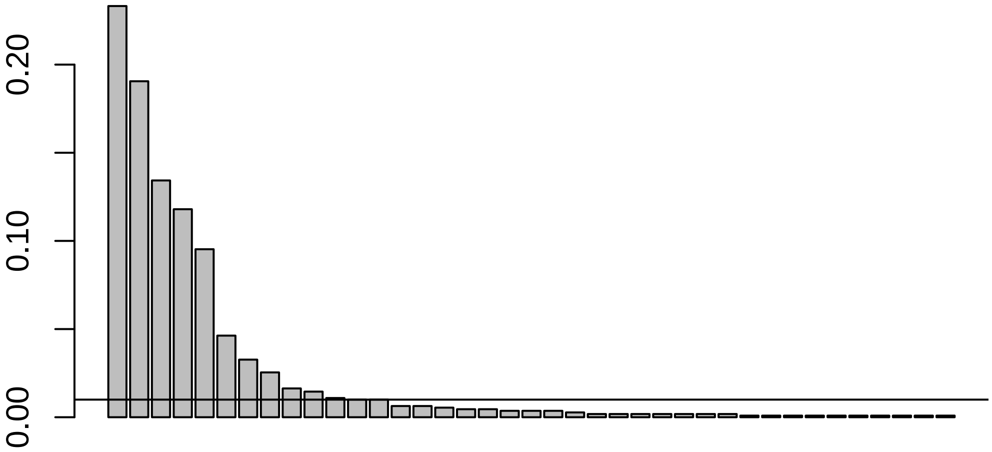

Figure 5: A getcut plot for the Crohn's disease dataset.

References

Baccam P, Thompson R, Li Y, Sparks W, Belshan M, Dorman K, Wannemuehler Y, Oaks J, Cornette J, Carpenter S (2003). "Subpopulations of Equine Infectious Anemia Virus Rev Coexist In Vivo and Differ in Phenotype." *J Virol*, **77**(22), 12122–12131.

Dempster A, Laird N, Rubin D (1977). "Maximum Likelihood Estimation from Incomplete Data via the EM Algorithm." *J R Stat Soc. B.*, **39**(3), 1–38.

Felsenstein J (2004). *Inferring Phylogenies.* Sinauer Associates.

Fraley C, Raftery A (2002). "Model-Based Clustering, Discriminant Analysis, and Density Estimation." *J Am Stat Assoc*, **97**, 611–631.

Hasegawa M, Kishino H, Yano T (1985). "Dating of the Human-Ape Splitting by a Molecular Clock of Mitochondrial DNA." *J Mol Evol*, **22**(2), 160–174.

Hubert L, Arabie P (1985). "Comparing partitions." *Journal of Classification*, **2**, 193–218.

Hudson R (2002). "Generating Samples under a Wright-Fisher Neutral Model of Genetic Variation." *Bioinformatics*, **18**, 337–338.

Hugot J, Chamaillard M, Zouali H, Lesage S, Cezard J, Belaiche J, Almer S, Tysk C, O'Morain C, Gassull M, Binder V, Finkel Y, Cortot A, Modigliani R, Laurent-Puig P, Gower-Rousseau C, Macry J, Colombel J, Sahbatou M, Thomas G (2001). "Association of NOD2 Leucine-Rich Repeat Variants with Susceptibility to Crohn's Disease." *Nature*, **411**.

Jukes TH, Cantor CR (1969). "Evolution of Protein Molecules." In HN Munro, JB Allison (eds.), *Mammalian Protein Metabolism*, volume 3, pp. 21–132. Academic Press, New York.

Kimura M (1980). "A Simple Method for Estimating Evolutionary Rates of Base Substitutions through Comparative Studies of Nucleotide Sequences." *J Mol Evol*, **16**, 111–120.

Maitra R, Melnykov V (2010). "Simulating Data to Study Performance of Finite Mixture Modeling and Clustering Algorithms." *J Comput Graph Stat.* (in press).

Paradis E, Claude J, Strimmer K (2004). "APE: analyses of phylogenetics and evolution in R language." *Bioinformatics*, **20**, 289–290.

Rambaut A, Grassly N (1997). "Seq-Gen: An Application for the Monte Carlo Simulation of DNA Sequence Evolution along Phylogenetic Trees." *Comput Appl Biosci*, **13**(3), 235–238.

Saitou N, Nei M (1987). "The Neighbor-Joining Method: A New Method for Reconstructing Phylogenetic Trees." *Mol Biol Evol*, **4**(4), 406–425.

Tzeng JY (2005). "Evolutionary-Based Grouping of Haplotypes in Association Analysis." *Genet Epidemiol*, **28**, 220–231.