A Quick Guide for the phyclust Package

(Based on Version 0.1-2)

Wei-Chen Chen
Iowa State University

Contents

Acknowledgement

The author thanks Dr. Karin S. Dorman and Dr. Ranjan Maitra in Iowa State University for their generous support, and is funded by Dr. Maitra's grand, NSF CAREER DMS-0437555.

1. Introduction

Without further notifications, this document is written for major functions and it should work consistently for the later version.

This is a quick guide for the **phyclust**, and I demonstrate major functions in this document. They includes reading and writing sequence data, two famous programs `ms` and `seq-gen` (Hudson 2002; Rambaut and Grassly 1997) for generating a coalescent tree and sequences based on the tree that both programs have been incorporated into the **phyclust**, the main function `phyclust()` for finding sequence structures, and Haplo-Clustering (Tzeng 2005). More information about theory, examples for other tool functions and new added functions can be found on our website: Phylogenetic Clustering at `http://thirteen-01.stat.iastate.edu/snoweye/phyclust/`.

In the Section 2, I introduce the basic data structures of the **phyclust** and I/O functions for reading and writing basic PHYLIP and FASTA files. In the Section 3, I redo the "ms+seqgen" approach in R. In the Section 4, I briefly describe the Phylogenetic Clustering, visualization functions, the main function `phyclust()`, the auxiliary function `.EMControl()` for models, initializations, optimizations, and EM algorithms, and propose a "ms+seqgen+phyclust" approach. In the Section 5, I display the function `haplo.post.prob()` for Hap-Clustering. In the Section 6, I discuss some important issues which are in development or will be implemented in the next version.

1.1. Installation

You can install directly from CRAN at `http://cran.r-project.org` or download the **phyclust** from our website. In most systems, you can install the **phyclust** by typing the command into the R's terminal as

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

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

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

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

1.2. Need help

You can look and check more examples from the help pages or our website: `http://thirteen-01.stat.iastate.edu/snoweye/phyclust/`. Also, you can mail to `phyclust@gmail.com`. All commands are welcome, and bugs for the **phyclust** package or suggestions for Phylogenetic Clustering will be fixed and implemented in the new version.

2. Sequence Data Input and Output

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

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

There are three input sources of sequence data in the **phyclust**:

1. Read the data from a text file in the PHYLIP format (Section 2.2).
2. Read the data from a test file in the FASTA format (Section 2.3).
3. Simulated by `ms()` and `seqgen()` (Section 3).

The reading functions `read.*()` will return a list object with class `seq.data` (Section 2.2), and transfered data based on the standard coding and stored in a major element `org` exactly used in most functions of the **phyclust**. There are two ways to output sequence data in the **phyclust**, either the PHYLIP format or the FASTA format.

2.1. Standard coding

I use several internal objects to store default ids, and two of them are related to sequence structure, `.nucleotide` and `.snp`, which store the mapping information in `data.frame`. They are used to transfer data when reading and writing sequences. By typing the names, we can see the details as

```
> .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    -
```

The headings `nid` and `sid` are standard ids used in **phyclust**, and `code` and `code.l` are general syntax for nucleotide $\{A, G, C, T\}$ and SNP $\{1, 2\}$ sequences. The standard ids will be directly passed to the kernel of **phyclust** in C for efficient optimizations, so sequences are coded by integers stated from 0. Note that I use "`-`" to indicate gaps and other non general syntax. The computations and functions to deal with "`-`" are developing.

2.2. PHYLIP format

An example is "Great pony 524 EIAV rev dataset" (Baccam *et al.* 2003), and you can view the file as

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

Here is the first 5 sequences and the first 50 sites. The first line says that there are 146 sequences and 405 sites in this files. The sequences are started from the second line, and the first 10 characters are reserved for the sequence's 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, the `read.phylip()` will read in a PHYLIP file and assume the file contains nucleotide sequences. It will read in sequences and store in a list object with class `seq.data`, and the element `org.code` store the original data in a character matrix, and the element `org` store the transfered original data in a numerical matrix. The transfered data are based on the standard coding in the Section 2.1. The following is an example to read the pony524 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"
```

Another example is "Crohn's disease SNP dataset" (Hugot *et al.* 2001), and the following is an example to read in SNP sequences by changing `code.type` to SNP.

```
> 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.
```

2.3. FASTA format

An example is "Great pony 625 EIAV rev dataset" (Baccam *et al.* 2003) Here is the first one sequences and all 406 sites. It start with ">" and followed by sequence's id and descriptions, then is followed by couple lines containing sequence itself.

```
>AF512608 Equine infectious anemia virus isolate R93.3/E98.1 gp45 and rev
GATCCTCAGGGCCCTCTGGAAAGTGACCAGTGGTGCAGGGTCCTTCGGCAGTCACTACCT
```

```
GAAGAAAAAATTCCATCGCAAACATGCATCGCGAGAAGACACCTGGGACCAGGCCCAACA
CAACATACACCTAGCAGGCGTGACCGGTGGATCAGGGAACAAATACTACAGGCAGAAGTA
CTCCAGGAACGACTGGAATGGAGAATCAGAGGAGTACAACAGGCGGCCAAAGAGCTGGAT
GAAGTCAATCGAGGCATTTGGAGAGAGCTACATTTCCGAGAAGACCAAAAGGGAGATTTC
TCAGCCTGGGGCGGTTATCAACGAGCACAAGAACGGCACTGGGGGGAACAATCCTCACCA
AGGGTCCTTAGACCTGGAGATTCGAAGCGAAGGAGGAAACATTTAT
```

By default, the `read.fasta()` will read in a FASTA file and assume the file contains nucleotide sequences. As `read.phylip()`, it also return a list object with class `seq.data`. The following is an example to read the pony524 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. Save sequences

To save sequences in files, you can use functions `write.*()` which are analogical to functions `read.*()` but input a data matrix `X` and a file name `filename`. The following I save 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

The **phyclust** incorporates two famous outsourced `C` programs **ms** (Hudson 2002) and **seq-gen** (Rambaut and Grassly 1997). The original source code and documents are available on the author's websites. For **ms**, the pdf file (download from the author's website) in the installed directory `phyclust/doc/Documents/msdoc.pdf` or in the source code directory `phyclust/inst/doc/Documents/msdoc.pdf` For **seq-gen**, the html file (download from the author's website) in the installed 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.

In the file msdoc.pdf, Dr. Hudson demonstrated examples to use **ms** to generate coalescent trees and piped them to **seq-gen** to generate sequences in the command mode. The **phyclust** directly uses their options in R functions `ms()` and `seqgen()`, and also modify partial source code to use R's library. Now, they can be distributed with R across platforms without recompiling problems. Moreover, combining with the `phyclust()` function, we can have a `ms+seqgen+phyclust` approach in the Section 4.4 for simulation and bootstrap studies.

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

Almost all options are carried from the command mode program **ms**, and input as an option `opts` in `ms()`. Just call the function, it will return and show you all options.

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

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

3.2. Use the `seqgen()` function to generate sequences

Almost all options are carried from the command mode program **seq-gen**, and input as an option `opts` in the `seqgen()` function. Just call the function, it will return and show you all options. The `seqgen()` function requires to take in a rooted tree either NEWICK format or an object with class `phylo`.

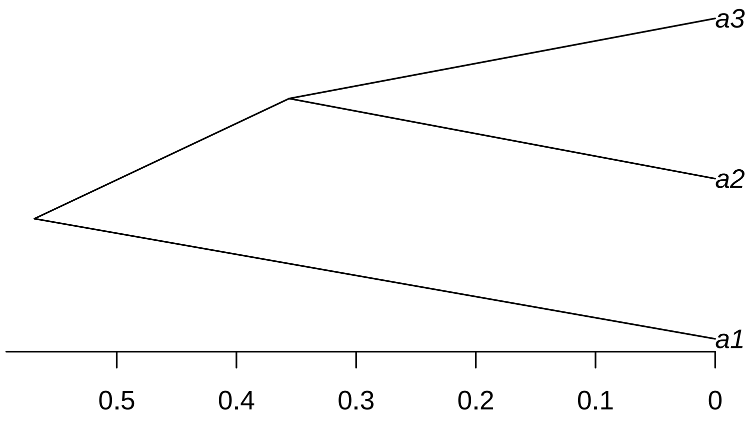

Figure 1: A diagram of a simple coalescent tree.

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

In the following, I demonstrate the `ms+seqgen` approach to generate sequences according a coalescent tree. This returns a character vector with class `seqgen` and contains 5 sequences, and each has 40 bases (`-l40`). The option `-mHKY` is for the HKY85 model (Hasegawa *et al.* 1985), but it is equivalent to the JC69 model (Jukes and Cantor 1969) if no further setting is submitted.

```
> 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 does not necessary to take in a tree from the `ms()` function, but the `ms()` function provides varied ways to construct a tree in different shapes based on the coalescent theory. The `seqgen()` function also allows to input an ancestral sequence and evolves the sequence along the given tree by inputing an option `input`. Originally, it uses a file to store the information in the **seq-gen** package, and I implement a function to utilize this option described in the Section 3.3.

3.3. Give an ancestral sequence to the `ms+seqgen`

The **phyclust** package provides two functions `gen.seq.HKY()` and `gen.seq.SNP()` to implement the `ms+seqgen` approach under a wide-range parameters. A rooted tree is required and an ancestral sequence is an option.

The following example generates a tree first, and gives an ancestral sequence `anc.HKY`. The process in the `seqgen()` will follow the parameters κ (`kappa`) and $\pi_A, \pi_G, \pi_C, \pi_T$ (`pi.HKY`) to evolve the ancestral sequence (`anc.HKY`).

```
> # 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
```

Note that the details of the `gen.seq.HKY()` may be a good example for advance users to develop more flexible conditions such as recombinations, migrations and island models. Basically, it passes an option `input` to the `seqgen()`, and it can be the ancestral sequence or other options used in the **seq-gen** program. The `input` takes in a character vector (including the tree) where each element contains one line, and it will be store/write to a temporary file in the `seqgen()` for further processing.

```
### Source code from 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)

### Source code from 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 a model-based approach based on evolution theories to determine population structures in molecular level. Let $\boldsymbol{X} = (x_{nl})_{N\times L}$ be the data matrix containing N sequences observed of L sites where $n = 1,\ldots,N$ and $l = 1,\ldots,L$. Denote the sequence $\boldsymbol{x}_n = (x_{n1},\ldots,x_{nL}) \in \mathfrak{X}$ and $x_{nl} \in \mathcal{S}$ where $\mathfrak{X}$ contains all possible sequences and $\mathcal{S}$ contains bases, e.g. $\mathcal{S} = \{\mathrm{A},\mathrm{G},\mathrm{C},\mathrm{T}\}$ for nucleotide sequences.

A finite mixture distribution for model-based clusterings is

$$f(\boldsymbol{x}_n|\boldsymbol{\eta},\boldsymbol{\Theta}) = \sum_{k=1}^{K} \eta_k f_k(\boldsymbol{x}_n|\Theta_k)$$

where $f_k()$ is the density for kth component, $\boldsymbol{\eta} = \{\eta_1,\ldots,\eta_K\}$ is the mixture proportion summing to one, and $\boldsymbol{\Theta} = \{\Theta_1,\ldots,\Theta_K\}$ contains parameters for components. By the Continuous Time Markov Chain theory, $f_k()$ is modeled as transition probability $p_{\boldsymbol{\mu}_k,\boldsymbol{x}_n}(t)$ of mutation process (Felsenstein 2004). A sequence $\boldsymbol{x}_n$ evolves from a representative $\boldsymbol{\mu}_k = (\mu_{k1},\ldots,\mu_{kL}) \in \mathfrak{X}$ dominating the kth cluster where $\mu_{kl} \in \mathcal{S}$, and process evolves with parameters $\boldsymbol{Q}_k$ in time t_k which are allowed to differ, so that $\Theta_k = \{\boldsymbol{\mu}_k,\boldsymbol{Q}_k,t_k\}$, and the transition probability matrix can be computed by $\mathbb{P}(t_k) = e^{\boldsymbol{Q}_k t_k}$ fot constructing likelihood functions. This model can be solved by EM algorithms (Dempster *et al.* 1977), sequences can be classified by the maximum posterior probabilities, and the number of clusters can be assessed by bootstrapping (Maitra and Melnykov 2010).

Usually, the $\boldsymbol{\mu}_k$'s are different with each other and represent the central sequences of subpopulations. The major evolution models used in Q_k for nucleotide sequences supported in the **phyclust** include `JC69` (Jukes and Cantor 1969), `K80` (Kimura 1980), and `HKY85` (Hasegawa *et al.* 1985) which are indicated in `.substitution`. I use an identifier (`.identifer`) to indicate possible combinations of models for $\boldsymbol{Q}_k$ and t_k and list in the Table 1.

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`	$\boldsymbol{Q}_1 \neq \boldsymbol{Q}_2 \neq \cdots \neq \boldsymbol{Q}_K$	$t_1 \neq t_2 \neq \cdots \neq t_K$

Note that there is an evolution distance model `.edist.model` that I use to indicate the model for computing distance of paired sequences, and they may differ to `.substitution`. There are more options in the **ape** package (Paradis *et al.* 2004).

The `.show.option()` function will list all options available in the **phyclust** package as the following. These options can be used in the `.EMControl()` function to generate a template for the `phyclust()` function.

```
> .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
```

All options will be explained in the help page and the short explanation will be given on our website. Some options may perform better than others in different situations. For EM algorithms, several initializations are necessary to obtain a better result, see the Section 4.3.

4.1. Illustrate data

The toy dataset has 100 nucleotide sequences and 200 sites in 4 clusters where the ancestral tree height 0.15 and the descendant tree height 0.09, and sequences are evolved by a HKY85 model (Hasegawa *et al.* 1985). The 4 clusters are isolated in groups, but they also carry common information about their ancestral sequences and differ with each other at some mutated sites.

The following code shows a plot in the Figure 2 to illustrate the sequences. Each row represents a sequence in the order of the dataset, the matrix `X`, and each column represents a site. Assume the first sequence in the first cluster of the toy dataset is the common/consensus sequence. The dashed lines split the clusters. The row fully drawn with colors is the common sequence, and colors represent 4 different nucleotides. Other rows are drawn the mutations indicated by colors comparing to the common sequence. The bottom row indicates the segregating sites which are sites containing at least one mutations with the origin dots.

```
> 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 following code provides a plot in the Figure 3 to show the number of mutations by comparing all sequences to the common sequence. The top plot is for the whole dataset, the second to the bottom plots are for the sequences in the first to the fourth clusters indicating by colors.

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

The following code shows a plot in the Firgure 4 to illustrate the distance method which gives a rough structure of dataset. The `phyclust.edist()` function takes in a data matrix `X`, compute and return pairwise distances for all sequences where `.edist.model[3]` is `D_HAMMING` and the HAMMING distance is used as a distance measure. I apply a neighbor-joining method to build a tree based on the distance matrix. The colors drawn on the leaf branches indicate the clusters.

```
> (ret <- phyclust.edist(X, edist.model = .edist.model[3]))
Class 'dist'  atomic [1:4950] 4 3 4 7 2 4 5 5 8 2 ...
```

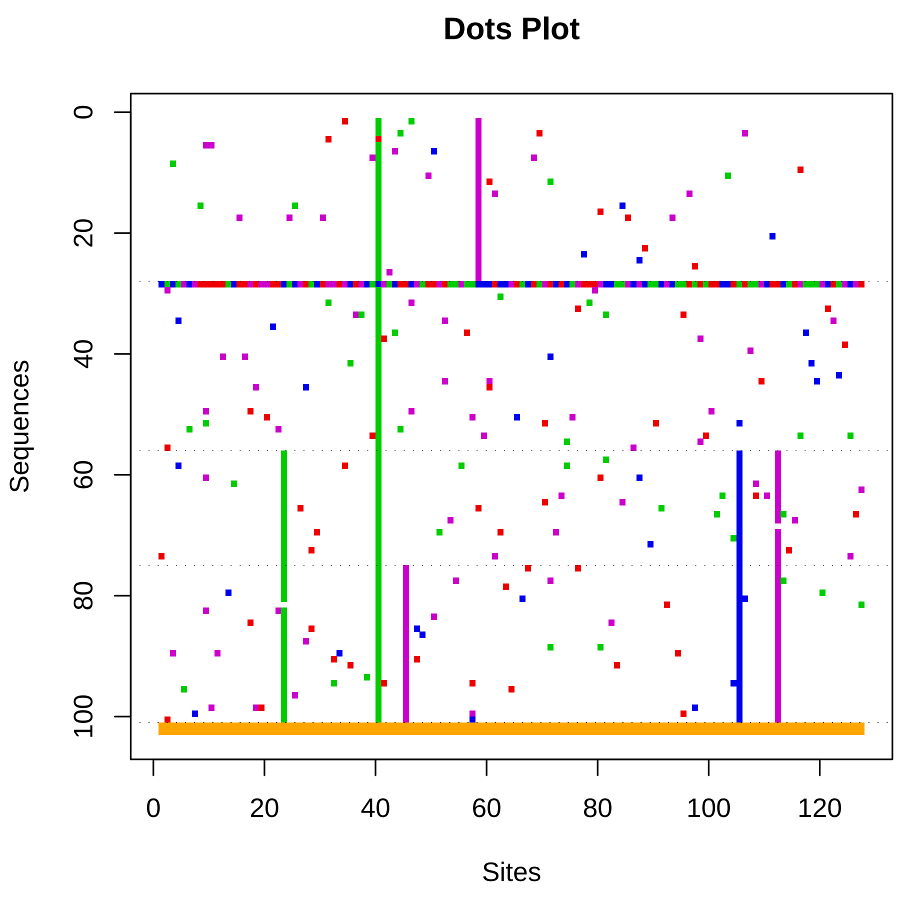

Figure 2: A dot plot for the toy dataset.

```
  ..- attr(*, "Size")= int 100
  ..- attr(*, "Diag")= logi FALSE
  ..- attr(*, "Upper")= logi FALSE
  ..- 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. Use the `phyclust()` function

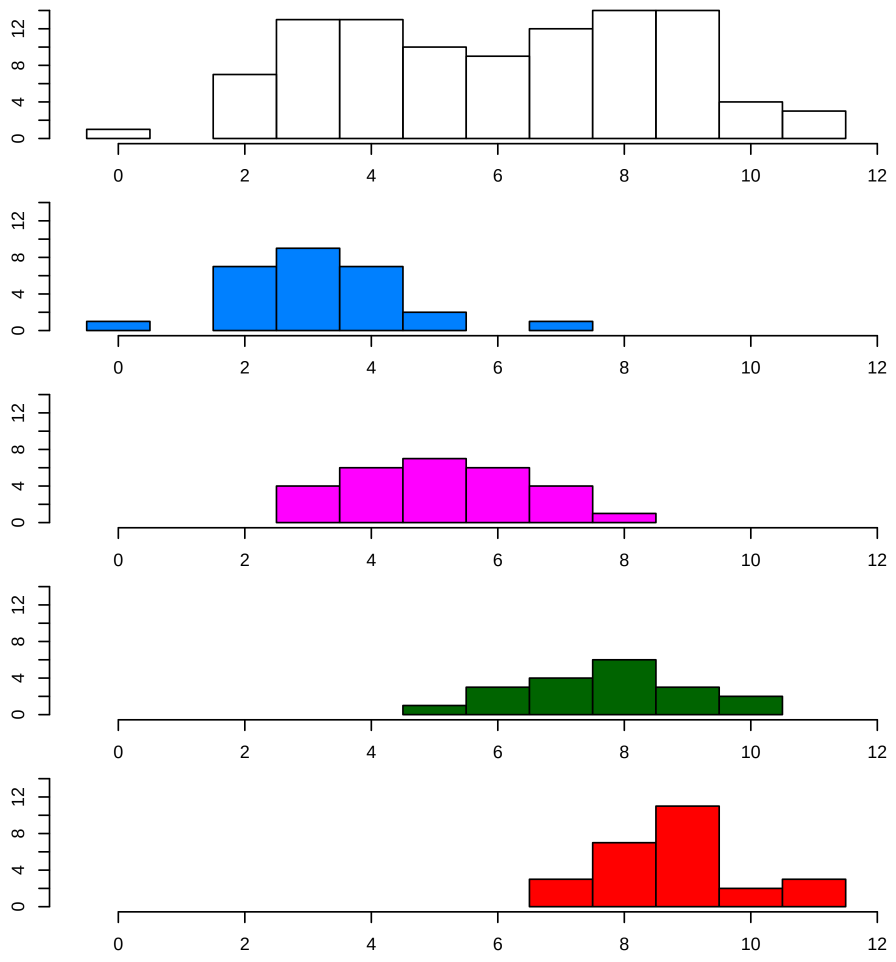

Figure 3: A histogram plot for the toy dataset.

I use the toy dataset to demonstrate the `phyclust()` function. Basically, the `phyclust()` function takes in two required objects, a data matrix `X` and the number of clusters `K`, then it will fit a default model for `X` based on `EMC = .EMC` which specifies models and optimizations. I will introduce this default control `.EMC` and the function `.EMControl()` to generate it in the Section 4.3. In the following example, I fit a default model with 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
```

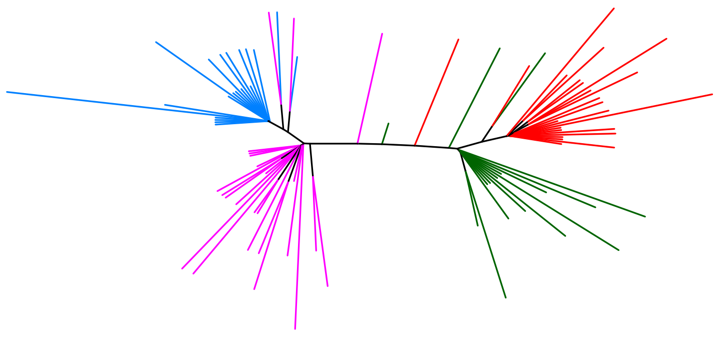

Figure 4: A NJ tree for the toy dataset.

```
identifier: EE
  Eta: 0.4360 0.01149 0.284 0.2700
  Tt: 0.003325
  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"
```

From the above reports, the default settings are used and the result does not fit well (with a degenerated cluster, see `n.class`) due to the initialization problem. The initialization procedure is `exhaustEM` and the initialization method is `randomMu`, so it randomly picks 4 sequences as the center of clusters and run the EM algorithm to convergence. While the EM algorithms do not guarantee to converge to global optimizations, more initializations should be explored to seek a better solution efficiently. The adjusted Rand index (Hubert and Arabie 1985), `adjRand`, is about 0.7653. The `phyclust` function returns a list object with class `phyclust` and it can be as an input of other functions such as the function `bootstrap.star.trees.seq()` for bootstrapping. in the Section 4.4.

4.3. Use the `.EMControl()` function

The `.EMControl()` function provides a list object as the default value for the `phyclust()` function. The internal object `.EMC` is a template. Each element indicates a configuration for evolution models, identifier, initialization, optimizations, and EM algorithms. See the help page for details, and visit our website for examples.

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

You can either modify from the template `.EMC` or use the function `.EMControl()` to generate a new control. First, the following example modifies an object coping from the template. It uses "emEM" as an initialization procedure, and the result of `ret.2` has a higher likelihood value than that of `ret.1`. The adjusted Rand index is also 1 that the prediction has a perfect match to the dataset.

```
> 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
```

Second, the following use the function `.EMControl()` to generate a new control that uses "RndEM" as an initialization procedure, and fit a model with an "EV" identifier. An over fitted model can also cause a degenerated cluster and usually needs more initializations to have a better result. From the output, the `Eta` of the second cluster is smaller than others, and the `Tt` gives an evolving time for sequences away from the ancestor of the cluster, and the second cluster has a longer time than others. This makes the second cluster is degenerated.

```
> 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
```

Note that an convenient function `find.best()` is useful to search the best result based on the highest likelihood value by repeatedly running on possible combinations of the `.EMControl()` function. This function may also take time to obtain a result.

4.4. The ms+seqgen+phyclust approach

Usually, the assessment for a fitted model includes the number of clusters, the evolution models, and the identifiers for clusters. All of these may rely on information criteria, but it may not accurate and sometimes it may give a wrong answer. A more elegant procedure is based on the parameter bootstrap technique (Maitra and Melnykov 2010). The basic idea is to bootstrap sequences by the functions `ms()` and `seqgen()` and re-sample from a fitted model, a result of the `phyclust` function.

The `bootstrap.star.trees.seq()` function implements this procedure that it takes in a fitted model, `pcobj`, a list object with class `phyclust` and utilizes the functions `ms()` and `seqgen()` to re-sample new datasets. We can perform the same fitting method on all new datasets to obtain an expected distribution of parameters, and compared to the observed distribution obtained from the fitted model.

The following gives an example how to obtain a new datasets from a model with 2 clusters based on the toy dataset. The function `bootstrap.star.trees.seq()` returns a list object contains two elements `trees` and `seq` for all clusters. Combining `seq` and using the function `read.seqgen` can read in the new dataset and save as a list object with class `seq.data`.

```
> set.seed(1234)
> ret.4 <- phyclust(X, 2)
> ret.all <- bootstrap.star.trees.seq(ret.4)
> str(ret.all)
List of 2
 $ trees:List of 2
  ..$ :List of 5
  .. ..$ edge       : int [1:102, 1:2] 53 54 54 55 56 57 57 58 58 56 ...
  .. ..$ Nnode      : int 51
  .. ..$ tip.label  : chr [1:52] "42" "52" "27" "47" ...
  .. ..$ edge.length: num [1:102] 0.00000 0.00385 0.00000 0.00000 0.00000 ...
  .. ..$ n.tip      : int 52
  .. ..- attr(*, "class")= chr "phylo"
  ..$ :List of 5
  .. ..$ edge       : int [1:94, 1:2] 49 50 50 49 51 52 53 54 54 55 ...
  .. ..$ Nnode      : int 47
  .. ..$ tip.label  : chr [1:48] "16" "43" "10" "38" ...
  .. ..$ edge.length: num [1:94] 0.00000 0.00385 0.00385 0.00000 0.00000 ...
  .. ..$ n.tip      : int 48
  .. ..- attr(*, "class")= chr "phylo"
 $ seq  :List of 2
  ..$ :Class 'seqgen'  chr [1:53] " 52 200" "42        GAGATCTTGACCGCTTT ...
  ..$ :Class 'seqgen'  chr [1:49] " 48 200" "16        GAGATCTTGACCGCTTT ...
> toy.new <- c(paste(ret.4$N.X.org, ret.4$L, sep = " "),
+              ret.all$seq[[1]][-1], ret.all$seq[[2]][-1])
> ### Not necessary, but keep consistence.
> ### class(toy.new) <- "seqgen"
> (X.new <- read.seqgen(toy.new))
code.type: NUCLEOTIDE, n.seq: 100, seq.len: 200.
> (ret.5 <- phyclust(X, 2))
Phyclust Results:
code type: NUCLEOTIDE, em method: EM, boundary method: ADJUST.
init procedure: exhaustEM, method: randomMu.
model substitution: JC69, distance: D_JC69.
iter: 39 3035 0, convergence: 0, check.param: 1.
eps: 1.616e-12, error: 0.
N.X.org: 100, N.X.unique: 87, L: 200, K: 2, p: 402, N.seg.site: 127.
logL: -1571, bic: 4993, aic: 3946, icl: 4994
identifier: EE
  Eta: 0.4444 0.5556
  Tt: 0.003846
  n.class: 44 56
```

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

Haplotype Grouping (Tzeng 2005) for SNP datasets is a simplified/degenerated method of Phyloclustering. The author's code has been integrate into the **phyclust** package, and the original function has been renamed as `haplo.post.prob()`. The example used by the author is the Crohn's disease dataset (Hugot *et al.* 2001) which is also built in the package.

The following example returns the same results as Tzeng (2005), and the predicted number of clusters based on the information criterion is 13. The function returns a list object and stores in `ret` that `ret$haplo` stores information for the SNP sequences, `ret$FD.id` and `ret$RD.id` stores full and reduced dimensional index, `ret$FD.post` and `ret$RD.post` stores full and reduced dimensional posterior probabilities, and `g.truncate` show the truncated results.

```
> 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()` also illustrates a plot based on the information criterion to decide the truncated dimension, and the Firgure 5 shows the results, 13 haplotypes will be used in the `haplo.post.prob()` function.

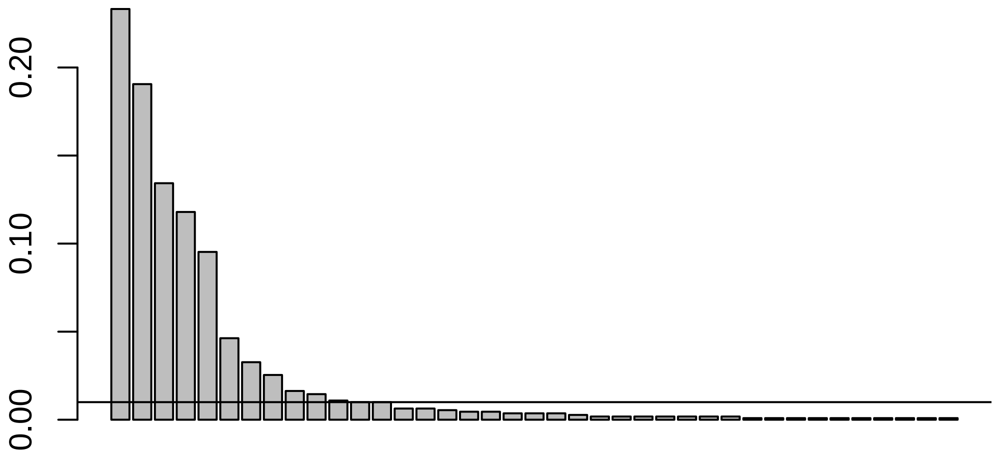

Figure 5: A getcut plot for the Crohn's disease dataset.

6. What is Next

Combining functions `ms()`, `seqgen()` and `phyclust()`, some functions automatically for simulations and bootstraps are being developed that can be used to examine and test different models and evolution conditions. Gaps will be considered and involved in the models. Semi-supervised clustering can be easily extended from the core of **phyclust**.

My future goal is to provide efficient tools and educational utilities for science researches in statistics ways. Visualization for large dataset (Pemberton *et al.* 2008; Conrad *et al.* 2006) can be implemented in R and evolution models for microsatellite dataset (Rosenberg *et al.* 2002; Shringarpure and Xing 2009) can be built by extending **phyclust**'s core.

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.

Conrad D, Jakobsson M, Coop G, Wen X, Wall J, Rosenberg N, Pritchard J (2006). "A Worldwide Survey of Haplotype Variation and Linkage Disequilibrium in the Human Genome." *Nat Genet*, **38**(11), 1251–1260.

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.

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." Accepted.

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

Pemberton T, Jakobsson M, Conrad D, Coop G, Wall J, Pritchard J, Patel P (2008). “Using Population Mixtures to Optimize the Utility of Genomic Databases: Linkage Disequilibrium and Association Study Design in India.” *Ann Hum Genet*, **72**, 535–546.

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.

Rosenberg N, Pritchard J, Weber J (2002). “Genetic Structure of Human Populations.” *Science*, **298**, 2381–2385.

Shringarpure S, Xing E (2009). “mStruct: Inference of Population Structure in Light of Both Genetic Admixing and Allele Mutations.” *Genetics*, **182**, 575–593.

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