The Gap Procedure

Irene Vrbik, David A Stephens, Michel Roger, and Bluma G Brenner

2019-07-15

Contents

Introduction

The Gap Procedure is a distance-based clustering algorithm for finding groups among HIV-1 DNA sequences data. Loosely speaking, this algorithm offers a simplified alternative to phylogenetic clustering approaches that rely on the reconstruction of a phylogenetic tree. This alogrithm operates on the assumption that groups are non-overlapping and well-separated. As such, this simplistic approach to clustering will not be appropriate for all applications.

Herein, a quick tutorial of the Gap Procedure is presented. We include the installation procedure, code used for generating genetic sequence data, a quick demonstartion of a cluster analysis using the Gap Procedure, and some post analysis testing for validating clusters. The `GapProcedure` package has been tested on Mac OSX 10.10.5 and Ubuntu 12.04 (R version 3.2.2). If you are encountering any errors, please feel free to contact me at irene.vrbik@math.mcgill.ca.

Getting Started

The Gap Procedure is performed using the software environment R which is available for free download here. For ease of distribution, the required functions, along with data sets, and help files have been assembled into an R package. This section describes how to install the pacakges required for this vignette. Note that are "`#>`" and "`##`" are used to denote code output and comments, respectively.

Installation Instructions

To use the `Gap Procedure` you must first download the compressed R file (file ending in `".tar"`) from where am I putting it? While in R console, you can install the pacakge by typeing:

```
install.packages("path/to/file", repos = NULL, type="source")
```

Example on Windows:

```
install.packages("C:/Users/irene/Desktop/GapProcedure_0.1.tar", repos=NULL, type="source")
```

Example on Mac:

```
install.packages("/Users/irene/Desktop/GapProcedure_0.1.tar", repos=NULL, type="source")
```

Once installed, do you can load the library by typing:

```
library("GapProcedure")
```

Note that the "`install.packages`" step only needs to be done once. However, the library will have to be loaded—that is, the "`library(GapProcedure)`" line must be run—at the start of each new R session. Help files can be seen by typing

```
?GapProcedure
```

The following examples will also rely on the `ape` (Paradis, Claude, and Strimmer 2004) and `phyclust` (Chen and W.C. 2011) package. This, along with many other packages covering a wide variety of topics, are distributed through the Comprehensive R Archive Network CRAN. Packages uploaded to the CRAN repository can be installed directly from the R console by simply calling the package name inside the `install.packages` function. Hence we can install and load these packages by typing:

```
install.packages(c("ape", "phyclust"))
```

Before proceeding to the following sections make sure that all of the pacakges are loaded by typing:

```
library("GapProcedure")
library("ape")
library("phyclust")
```

Importing Sequences

There are a number of ways to import sequence data into R. Below we describe how import a FASTA or NEXUS formated files using the `ape` package.

The following code assumes that the `ape` package has been loaded and that your file has been saved to under the name "`seq`" (extension will depend on the format of your data). Note that if you are using Windows, the forward slashes will need to be replaced by backwards slashes.

If your file is in "`fasta`" format, use the following command:

```
seqdata <- read.dna(file="<path>/seq.fasta", format = "fasta")
```

If your file is in "`NEXUS`" format, use the following command:

```
seqdata <- as.DNAbin(read.nexus.data(file="<path>/seq.nex"))
```

The Gap Procedure requires that sequences are aligned are `DNAbin` format (type `?DNAbin` into the R consolde for details). This can be verified by typing the following:

```
class(seqdata)
seqdata <- as.matrix(seqdata)
```

The first line should return `DNAbin` and the second line should run without errors.

Simulation

The standard convention for finding phylogenetic clusters among genetic HIV sequence data relies on the reconstruction of a phylogenetic tree. Graphically speaking, clusters usually correspond to clades in the tree having high support—measured either by bootstrap percentages or Bayesian posterior probabilities—and sufficiently small genetic distances. While these approaches have been ubiquitous in the HIV-1 literature, they are computationally burdensome and relying the user-specification of threshold values.

The Gap Procedure aims at finding nonrandom groupings of sequences which closely agree with the aforementioned methods, but do not require reconstructing a phylogenetic tree. Using simulated data, we explain the driving mechanisms behind the Gap Procedure and show when they are appropriate to use. Simulated data are created by reconstructing a phylogenetic tree with branches representing the estimated number of nucleotide substitutions. The "tips" (i.e., external nodes) of the tree represent sampled HIV-1 patients and internal nodes can be viewed as the source of a chain of infections. We refer to all sequences rooted by a common interior node, as *descendants* to the so-called *ancestor node*. This section demonstrates how to generate a phylogenetic tree and create a genetic seqeuence data set by mutating DNA sequences along its topology.

Generating a Phylogenetic Tree

To create data that resemble an HIV epidemic, we first generate a tree using the `seqgen()` function available in the `phyclust` package; the details for these options are provided in (Hudson 2002). The phylogenetic tree is constructed using these four steps:

1. Generate an *ancestor tree* with tips equal to the number of desired clusters
   - The tip "$\texttt{a}_g$" represents the ancestor node for the gth cluster; see Figure 1.
2. Generate a *descendant tree* for each cluster which is rooted at ancestor node $\texttt{a}_g$.
   - The number of tips in the gth tree will represent the number of members in cluster g.
   - The size (i.e., number of members) of a cluster is generated according to a multinomial distribution with N trials and $p_1 = p_2 = \cdots = p_G$.
3. Create the *complete tree* by attaching the G descendant trees to the tips of the ancestor tree.
4. Scale the tree so that the height complete tree (`height.c`) is 1, and the height of the ancestor tree is equal to `ratio.a` $= \frac{\text{height of the ancestor tree}}{\text{height of the ancestor tree}}$

The following code generates a phylogenetic tree comprised of four clusters.

Step 0: Specify the simulation parameters (e.g., Number of desired clusters, mememberss, etc., ...)

```
G <- 4                  # the number of clusters
n <- 25                 # rough number of observations per group (see N.G)
N <- n*G                # total number of observations
ratio.a <- 0.75         # the relative height of ancestor tree in the complete tree
height.c <- 1           # the height of the complete tree
seed <- 1234            # a random seed for reporducibility
```

The following generates a vector of length G containing the number of sequences in each cluster according to a multinomial distribution.

```
# generates the number of memebers per class according to a multinomial distribution
set.seed(seed)
(N.G  <-  as.vector(rmultinom(1, size = N, prob = rep(1/G,G))))
#> [1] 20 28 27 25
```

For example, the number of members assigned to cluster 3, is 27.

Step 1: Generate the Ancestor Tree

Phylogenetic trees typically involve bifurcating branches, i.e., exactly two diverging sequences; however, it may be the case that the rapid diversification of HIV is better modeled using *multifurcating* trees (Salemi and Vandamme 2003). Since clustering HIV sequences is our ultimate goal, we test our approach on multifurcating ancestor trees wherein multiple sequences emanate from a common node (or *root sequence*). The `ms()` function available in the `phyclust` package generates a random tree with `nsam` tips. Tips are renamed to have the form `a`$< g >$, where g is the cluster label. The `as.star.tree()` function (also available in the `phyclust` package) forces equal branch lengths in ancestor trees; the resulting star-like phylogeny is shown in Figure 1.

```
set.seed(seed)
ms.anc <- ms(nsam=G, 1, opts = paste("-T -G", 1, sep = " "))
tree.anc <- read.tree(text = ms.anc[3]) #Newick tree format
tree.anc$tip.label <- paste("a", 1:G, sep = "")
anc.tree <- as.star.tree(tree.anc)
plot(anc.tree)
```

Step 2: Generate the Descendant Trees

The following code generates G random decendant trees. Tips are renamed to have the form $d < k > . < g >$, where k represents the kth meber of the gth cluster. Note that these trees are not forced to have equal branch length (see Figure 2).

```
set.seed(seed)
tree.dec <- list()
tree.org <- tree.anc
tree.star <- tree.anc
for (k in 1:G) {
  ms.dec <- ms(nsam=N.G[k], 1, opts = paste("-T -G", 1, sep = " "))
  tree.dec[[k]] <- read.tree(text = ms.dec[3])
  tree.dec[[k]]$tip.label <- paste("d", k, ".", 1:N.G[k], sep = "")
}
plot(tree.dec[[1]])
```

Step 3 and 4: Generate the Scaled Complete Tree

Using functions available in the `phyclust` pacakge, the complete tree is obtained by attaching the decendant trees (*via* `bind.tree`) to the corresponding ancestor node (i.e., the tips of the ancestor tree). The `rescale.rooted.tree()` function is used to rescale the trees such that the total height of the complete tree is equal to `height.c`, and the ratio of the ancestor to complete tree height is equal to `rate.a`.

```
tmp.height <- treeHeight(anc.tree) #get.rooted.tree.height(anc.tree) # need my function "tree.height"
comp.tree <- rescale.rooted.tree(anc.tree, height.c*ratio.a/tmp.height)
tmp.tree <- anc.tree
for (k in 1:G) {#k=1
  tmp.tree <- tree.dec[[k]]
  tmp.height <- treeHeight(tmp.tree)
  tmp.tree <- rescale.rooted.tree(tmp.tree, height.c*(1 - ratio.a)/tmp.height)
  comp.tree <- bind.tree(comp.tree, tmp.tree,
                         where = which(comp.tree$tip.label == paste("a", k, sep = "")))
}
```

The following code is used to generate the complete tree given in Figure 2. The colours of the tips correspond to the generated cluster memberships. In the context of this simulation, the gth cluster corresponds to the tips of the decendant tree sharing the ancestor node $\mathtt{a}_g$. Graphically speaking, the clusters corresponds to a particular clade in the phylogenetic tree.

```
plot(comp.tree, adj=0, label.offset=0.02, lwd=2, cex=0.8); axis(1)
X.class <- rep(1:G, N.G)    # returns the generated class labels for the tips of the complete tree
tiplabels(text=X.class, cex=0.6, col="black", bg=X.class+1, frame="circle")
```

Generating Genetic (DNA) Sequence Data

To simulate our data, we mutate a randomly generate root DNA sequences along the complete tree generated in the previous section. Sequences are mutated according to a General Time Reversible model which assumes rate heterogeneity and a proportion of invariable sites. The rate heterogeneity model assumes that sites evolve according to random variable $R \sim \Gamma(\alpha, \beta)$ parameterized by a single shape parameter α, where $\alpha = \beta$. In accordance with Posada and Crandall (2001), we set these parameters to values typical for HIV-1 Subtype B on the *pol* gene; namely, we set α (`alpha.rate`) equal to 0.7589, the proportion of invariant sites to (`inv.sites`) equal to 0.4817, and specify the rate matrix to the object `Q.mat` defined below:

```
L <- 800                                # length of desired sequence
pi.HKY <- c(0.39, 0.17, 0.22, 0.22)     # equilibrium probs (piA,  piC, piG, piT)
inv.sites <- 0.4817                     # percent of invariant cites
alpha.rate <- 0.7589                    # alpha for the gamma distribution
GTR <- matrix(0, nrow=4, ncol=4)
dimnames(GTR) <- list(c("A","C","G","T"),c("A","C","G","T"))
GTR[lower.tri(GTR)] <- c(3.37,14.50,1.44,1.21,14.50,1.00)
GTR <- GTR + t(GTR)
Q.mat <- GTR%*%diag(pi.HKY)
```

The following code is used to simulate the evolution of nucleotide sequences along the complete tree `comp.tree`, according to the model described above. For details see `?seqgen` and the original documenation of seq-gen found here.

```
my.ttips <- comp.tree$Nnode + 1
rate.vals <- c(GTR["A","C"],GTR["A","G"],GTR["A","T"],GTR["C","G"], GTR["C","T"], GTR["G","T"])
set.seed(seed) # set random see here (Random seed parameter "-z" does not work in seqgen)
(myopts <- paste("-mGTR",  " -l", L, " -a", alpha.rate, " -i",
                 inv.sites," -r", pcsv(rate.vals), " -f",
                 pcsv(pi.HKY), " -u", my.ttips + 1, " -q", sep = ""))
#> [1] "-mGTR -l800 -a0.7589 -i0.4817 -r3.37,14.5,1.44,1.21,14.5,1 -f0.39,0.17,0.22,0.22 -u101 -q"
sim1.ret <- seqgen(opts = myopts, rooted.tree = comp.tree)

sim1.dat <- read.seqgen(sim1.ret)
```

The ouputed sequences are of the class `seqgen`. Before we can apply the Gap Procedure on these sequences we must fist convert them to `DNAbin` format using the `seqgen2DNAbin` from the `GapProcedure` package.

```
x <- seqgen2DNAbin(sim1.ret)
# shuffle for good measure
set.seed(seed)
shuffled.index <- sample(100)
x <- x[shuffled.index,]
class(x)
#> [1] "DNAbin"
X.class <- X.class[shuffled.index]
```

This data can be quickly accessed in `data.frame` format using the following command.

```
data(simulation)
Y.class <- simulation[,1]
y <- as.DNAbin(as.alignment(as.matrix(simulation[,-1])))
```

Clustering Using the Gap Procedure

The Gap Procedure is a distance-based clustering methods which uses gaps in sorted pairwise distances to determine natural clusters within the data. Our data, $\mathbf{X} = \{X_1, \ldots, X_N\}$, consist of N DNA sequences of equal length L. Let $\mathbb{D}$ be a $N \times N$ distance-matrix whose ijth element is equal to $d(X_i, X_j)$, the genetic distance between sequences X_i and X_j. The goal of the Gap Procedure is partition the N sequences into G wel-separated non-overlapping groups such that sequences within a cluster are more similar to each other than to those sequences between clusters. The following line will run the Gap Procedure from start to finish.

```
GP <- GapProcedure(x)
```

The `GapProcedure` function returns three objects: `dist`, `classifcation`, and `AlgorithmOutputs`. In this section, we break down the `GapProcedure` function step-by-step.

Computing the Pairwise Distance Matrix

Unless otherwise specified, the matrix of pairwsie distances is calculated within the `GapProcedure` function. The current implementation of this package allows for the adjusted Jukes and Cantor (1969), and the adjusted Kimura (1980) distances. These adjustments take into account ambiguous nucleotides (see the Additional Files to the associated Gap Procedure paper). To calculate the distance matrix using one of these methods, the `submod` argument is specified to `"aJC69"` or `"aK80"`, respectively (make note of the use of quotations). If the default setting (`"aK80"`) is used, the `submod` argument does not need to be specified.

```
# computes the pairwise distance matric using adjusted Kimura 1980 calculationes
GP <- GapProcedure(x)
# GP is equivalent to GP2
GP2 <- GapProcedure(x, submod = "aK80")
# computes the pairwise distance matric using adjusted Jukes and Cantor 1969 calculationes
GP3 <- GapProcedure(x, submod = "aJC69")
```

The distance matrix can be called from the output object using `$dist` (see below). Since the generated data does not contain any ambiguious DNA characters, these adjusted distance calculations are indentical to the non-adjusted counterparts which can be calculated, for example, using the `dist.dna` function in the `ape` package.

```
ak80dist <- GP$dist
k80dist <- dist.dna(x,model="K80")
all.equal(ak80dist, as.matrix(k80dist), check.attributes=FALSE)
#> [1] TRUE
aj69dist <- GP3$dist
j69dist <- dist.dna(x,model="JC69")
all.equal(aj69dist, as.matrix(j69dist), check.attributes=FALSE)
#> [1] TRUE
```

The user has the option of specifying their own pairwise distance matrix in the `distance.matrix` field. For instance, if the output from the `dist.dna` function with the TN93 (Tamura and Nei 1993) model would like to be used, we could do so by typing:

```
tn93dist <- dist.dna(x,model="TN93")
GP4 <- GapProcedure(x, distance.matrix = as.matrix(tn93dist))
```

If both a `distance.matrix` and a `submod` are specified, the former will be used.

Searching for the Gap

The `GapProcecedure` uses gaps in sorted pairwise distances to infer groups of genetically similar sequences. Let $\mathbb{D}$ be the $N \times N$ distance-matrix whose ijth element is equal to $d(X_i, X_j)$, the genetic distance between sequences X_i and X_j. In reference to sequence X_i, let

$$\boldsymbol{d}_{i\cdot} = \left(d(X_i, X_{[1]}), d(X_i, X_{[2]}), \ldots, d(X_i, X_{[N-1]})\right),$$

denote the sorted vector of pairwise distances between X_i and X_j (for all $j \neq i$) such that $d(X_i, X_{[1]}) \leq d(X_i, X_{[2]}) \leq \cdots \leq d(X_i, X_{[N-1]})$. We denote the difference between two adjacent elements in $\boldsymbol{d}_{i\cdot}$ by $\delta_{ij} = d(X_i, X_{[j+1]}) - d(X_i, X_{[j]})$.

For each individual sequence, the Gap Procedure defines a set of *nearest neighbours* that are subsequently used to determine a partition of the data. To be more specific, let $c_i = \max\{\delta_{i1}, \delta_{i2}, \ldots, \delta_{ik}\}$, where $k < N \times 0.9$. In other words, c_i corresponds to largest "gap" in the first 90% of $\boldsymbol{d}_{i\cdot}$ values[1].

Defining Nearest Neighbours

If we define k^* such that $\delta_{ik^*} \geq \delta_{ik}$ for all $k \neq k^*$ and $d_i = d(X_i, X_{[k^*]})$, the then the nearest neighbour matrix $\mathbb{N} = \{n_{ij}\}$ can be defined as indicator matrix whose ijth element is given by

$$n_{ij} = \begin{cases} 1 & \text{if } d(X_i, X_j) \leq d_i, \\ 0 & \text{otherwise.} \end{cases}$$

A graphical representation of this can be seen in Figure 4. Note that the point labels correspond to the sequence index. In the example below, if sequence index j falls to the left of the vertical line, $n(1, j)$ is 1, otherwise, $n(1, j)$ is 0.

The $\mathbb{N}$ matrix can be obtained using this following command.

```
Nmat <- GP$AlgorithmOutputs$extra$NeighourMatrix
# the row corresponding to first sequence
Nmat[1,]
#>   [1] 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 1 1 1 1 0 0 0
#>  [36] 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 1 0 0 0 0 1 1 0
#>  [71] 1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1
# returns the nearest neighbours to sequence 1
which(Nmat[1,]==1)
#>  [1]   1   3   6  19  20  25  27  29  30  31  32  40  45  52  58  59  60
#> [18]  63  68  69  71  80  82  83  85  89  91 100
```

Each row of $\mathbb{N}$ defines a bi-partition of the data into "neighbours" and "non-neighbours". The relative frequencies of these bi-partitions are subsequently used to identify phylogenetic clusters. More specifically, we store the unique bi-partitions (i.e., rows of $\mathbb{N}$) in matrix $\mathbb{U}$ (`Umat` below). Each row of $\mathbb{N}$ is matched to a row in $\mathbb{U}$ and stored in the vector `uindex`. The relative frequencies of these bi-partitions stored in matrix $\mathbb{Z}$ (`Zmat` below).

```
# unique patterns matrix: eliminates all repeat rows in Nmat
Umat <- mgcv::uniquecombs(Nmat)

# maps the rows of Nmat to their matching row in Umat
uindex <- attr(Umat,"index")

# multiplies the rows of Umat by their frequency in Nmat
```

[1] As discussed in Additional files, this restriction was established to mitigate the effect of outlying observations.

```
Pmat <- Umat*as.vector(table(uindex))

# The relative frequency matrix (with each column summing to 1)
Zmat <- sweep(Pmat, 2,  colSums(Pmat), "/")
```

Clustering Sequences

The function `map2cluster()` is used convert this probability matrix into a vector of class labels. Specifically, X_i is classified into cluster g if $z_{gi} > z_{g'i}$ for all $g \neq g'$,

```
# assigns sequence to the cluster corresponding to the column in Zmat with the highest probability
cls <- map2cluster(Zmat)
```

The above matrices can be extracted from the `GapProcedure` output as follows:

```
GP.Umat <- GP$AlgorithmOutputs$extra$UniqueMatrix # same as Umat; all.equal(Umat, GP.Umat)
GP.Pmat <- GP$AlgorithmOutputs$extra$PropMatrix   # same as Pmat; all.equal(Pmat, GP.Pmat)
GP.Zmat <- GP$AlgorithmOutputs$FreqMatrix         # same as Zmat; all.equal(Zmat, GP.Zmat)
GP.cls <- GP$classification                       # NOT the same as cls
```

Notice that the classification labels produced by the Gap Procedure are different from those given in `cls`. That is because the algorithm maps the class labels to the smallest set of positive integer possible. In this case, the class lables found in `cls` (1, 2, 3, 5) are maped to the class labels used in `GP.cls` (1, 2, 3, 4).

Cluster Assessment

To compare these results with the simulated class, we use the Adjust Rand Index (ARI; Hubert and Arabie 1985). In this particular analysis, a value of 1 corresponds to perfect agreement with the simulated clusters; the expected value of the ARI is 0 under random classification. This measure can be computed using the `RRand()` function availble in the `phyclust` package. As demonstrated by the code below, the Gap Procedure attains perfect classifcation on this data set.

```
RRand(cls, X.class)
#>    Rand adjRand  Eindex
#>  1.0000  1.0000  0.1198
```

In order to ensure that the clusters produced by the Gap Procedure are meaningful, it is necessary to varify that there is sufficient genetic similarity (resp. diversity) between sequences within (resp. between) clusters. As a general guideline, we suggest looking at the side-by-side boxplots of these features to ensure that clusters are distinct and non-overlapping. Specifically we

References

Chen, and W.C. 2011. *Overlapping Codon Model, Phylogenetic Clustering, and Alternative Partial Expectation Conditional Maximization Algorithm. Ph.D. Diss., Iowa Stat University.* http://gradworks.umi.com/34/73/3473002.html.

Hubert, L., and P. Arabie. 1985. "Comparing Partitions." *Journal of Classification* 2: 193–218.

Hudson, Richard R. 2002. "Generating Samples Under a Wright–Fisher Neutral Model of Genetic Variation." *Bioinformatics* 18 (2). Oxford Univ Press: 337–38.

Jukes, Thomas H, and Charles R Cantor. 1969. "Evolution of Protein Molecules." Academic press.

Kimura, Motoo. 1980. "A Simple Method for Estimating Evolutionary Rates of Base Substitutions Through Comparative Studies of Nucleotide Sequences." *Journal of Molecular Evolution* 16 (2). Springer: 111–20.

Paradis, E., J. Claude, and K. Strimmer. 2004. "APE: Analyses of Phylogenetics and Evolution in R Language." *Bioinformatics* 20: 289–90.

Posada, David, and Keith A Crandall. 2001. "Selecting Models of Nucleotide Substitution: An Application to Human Immunodeficiency Virus 1 (HIV-1)." *Molecular Biology and Evolution* 18 (6). SMBE: 897–906.

Salemi, Marco, and Anne-Mieke Vandamme. 2003. *The Phylogenetic Handbook: A Practical Approach to Dna and Protein Phylogeny*. Cambridge University Press.

Tamura, Koichiro, and Masatoshi Nei. 1993. "Estimation of the Number of Nucleotide Substitutions in the Control Region of Mitochondrial Dna in Humans and Chimpanzees." *Molecular Biology and Evolution* 10 (3). SMBE: 512–26.