

# Microbial pangenomics in R - A case study

Lars Snipen and Kristian Hovde Liland

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Motivation . . . . .	2
1.2	This case study . . . . .	2
1.3	Organizing data and scripts . . . . .	2
<b>2</b>	<b>Preparing data</b>	<b>2</b>
2.1	Genome overview table . . . . .	2
2.2	The genome sequence data . . . . .	3
2.3	The protein sequence data . . . . .	4
2.4	Preparing protein sequence files . . . . .	5
<b>3</b>	<b>Comparing sequences</b>	<b>6</b>
3.1	Direct comparison using BLAST . . . . .	6
3.2	Indirect comparison using HMMER3 . . . . .	7
<b>4</b>	<b>Clustering sequences</b>	<b>9</b>
4.1	Clustering based on BLAST results . . . . .	9
4.1.1	BLAST distances . . . . .	9
4.1.2	Hierarchical clustering . . . . .	10
4.2	Clustering based on Pfam domains . . . . .	12
4.2.1	Domain sequences . . . . .	12
4.2.2	Clustering . . . . .	12
4.3	Direct or indirect comparison? . . . . .	14
<b>5</b>	<b>The pan-matrix</b>	<b>14</b>
<b>6</b>	<b>The pangenome tree</b>	<b>16</b>
<b>7</b>	<b>Pangenome size</b>	<b>18</b>
7.1	Binomial mixture models . . . . .	19
7.2	Other analyses . . . . .	20

# 1 Introduction

## 1.1 Motivation

The best way of learning about a new R package is to use it. This case study can serve as a template for later studies, or you can just step through parts of it to see how we make use of some of the functions in the **micropan** package. For more detailed explanation of the various functions, see the Help files. There are also some external softwares we make use of here, see the package vignette for how to install this.

## 1.2 This case study

In this case study we have chosen data from the species *Mycoplasma pneumoniae*. The only reasons we have chosen this species is that the genomes are small and at the time of writing this there are a few (4) complete and a few (3) draft genomes publicly available. Since we have only 7 genomes, and these are pretty small, computations will be fast.

We will focus on the workflow which is necessary to compute the pan-matrix, which is the central data structure in a pangenome study. Then we show some example of analyses based on a pan-matrix. We start out by the preparation of data, i.e. how to download genomes and predict genes, and then how to prepare the files for further computations. You may have your own genomes and protein sequence data, and only the last step is of any real interest. Next, we consider two ways of comparing sequences, using BLAST or HMMER3. Then we cluster the sequences based on either the BLAST- or the HMMER3-approach. Then pan-matrices can be computed, and finally we just show how this can be used to create pangenome trees and estimate core- and pangenome sizes.

## 1.3 Organizing data and scripts

When working with microbial pangenomics you need to handle quite large amounts of data, and a minimum of discipline with respect to organization will usually pay dividends. We prefer to organize each study in a separate folder, with a similar set of subfolders each time. We name the root folder for this study `Mpneumoniae/`.

# 2 Preparing data

Under the root folder we create a subfolder named `data/`. In this folder we will store the sequence data for every genome in the study.

## 2.1 Genome overview table

In the `data/` subfolder we like to have a small textfile containing a table listing some information about each genome in the study. This file is typically created in a spreadsheet and saved as a textfile, and can be read into R using the `read.table` function. In Table 1 we show an example of such a table. It may of course have many more columns, and we will come back to the data in this table

Table 1: A small table listing the genomes in the study. We recommend that all studies have such a textfile containing a tabular listing of the genomes. The first column lists the name of the FASTA-file where the genomes will be stored. The second column holds the GenBank accession number. Only the first 4 genomes are completed, and has a GenBank accession number. The third column is the GenBank accession for the master-record of the WGS data for the 3 draft genomes. The fourth column is named GID and contains just a unique integer for each genome. The two last columns contain the strain short-name and some colors we may use when displaying the genomes later. Many more columns may of course be added, but these will be used below.

File	Accession	MasterRecord	GID	Strain	Color
Mpneumoniae.M129.fsa	U00089.2		1	M129	cyan
Mpneumoniae.309.fsa	AP012303.1		2	309	blue
Mpneumoniae.FH.fsa	CP002077.1		3	FH	blue
Mpneumoniae.M129-B7.fsa	CP003913.1		4	M129-B7	cyan
Mpneumoniae.PO1.fsa		ANAA000000000	5	PO1	blue
Mpneumoniae.PL1428.fsa		ANAB000000000	6	PL1428	blue
Mpneumoniae.19294.fsa		ANIQ000000000	7	19294	blue

below. In this case study we assume this table has been saved as tab-separated columns in a file named `Mpneumoniae.txt` and put in the `data/` subfolder.

We start R and set the `Mpneumoniae/data/` as our working folder, and the table is read into R by

```
library(micropan)
options(stringsAsFactors=FALSE)
genome.table <- read.table("Mpneumoniae.txt", sep="\t", header
=TRUE)
```

We will make use this table later.

## 2.2 The genome sequence data

In the `data/` folder we create a subfolder named `genomes/`. Here we put the FASTA-files containing the genome sequence data. If you have all the genome or protein sequences you will need for your analysis you may skip the rest of this subsection.

Whole-genome data can be collected from many public databases. In this package we have a facility for downloading genomes from the National Centre for Biotechnology Information [5]. The Genome database at NCBI is considered one of the most comprehensive collections of public whole-genome data. At the same time, NCBI also provides an interface for downloading, either manually or by the use of the Entrez programming utilities (E-utilities, see <http://www.ncbi.nlm.nih.gov/books/NBK25501/>). It is the latter we use here.

Let us start out with the 4 completed genomes listed first in Table 1. We have a GenBank accession number for each genome, and have assigned a filename to each of them. We can download them using the function `entrezDownload`:

```
for(i in 1:4){
  out.file <- file.path("genomes",genome.table$File[i])
  entrezDownload(genome.table$Accession[i],out.file)
}
```

The function `entrezDownload` will retrieve the data with the accession number given in the first input, and store the result as a FASTA formatted file named by the second input. Obviously, both you and the servers at NCBI need to be online for this to succeed, you may occasionally have to retry at a later time to retrieve all the data.

Draft genomes are stored as a set of (many) contigs, each having its own accession number. If you have the accession number of the WGS master record, the function `getAccessions` can be used to retrieve the list of accession numbers for all contigs. In our table `genome.table` the three last genomes are draft genomes, and each of them has a master record accession number. We download them as follows:

```
for(i in 5:7){
  contig.acc <- getAccessions(genome.table$MasterRecord[i])
  out.file <- file.path("genomes",genome.table$File[i])
  entrezDownload(contig.acc,out.file)
}
```

The master record accessions will these days typically consist of 4 uppercase letters and then 8 zeros. If you scan the NCBI Genome table of prokaryotic projects, you will find accessions like `ANAA01`. Then, just replace the last 1 with 0 and add 6 more 0's. This system will not last forever, they never do.

If you inspect the variable `contig.acc` after you run the code above you will see it is a text containing many accession numbers separated by comma. In some cases the completed genomes also have more than one accession number, e.g. there are plasmids or more than one chromosome. In all cases you list multiple accession numbers as a single text where the accessions are separated by comma. There is, however, one exception. The Entrez programming utility at NCBI dislikes a query containing a huge number of accessions. For some draft genomes there are hundreds, sometimes thousands, of contigs. We have found that splitting these into chunks of 500 accession numbers is a solution, and the `getAccessions` function will automatically do this.

## 2.3 The protein sequence data

In pangenomics we focus on the protein coding genes. In the `data/` folder we create another subfolder named `proteins/`. If you have all the protein sequences you will need for your analysis you may skip the rest of this subsection.

For completed genomes there are always some protein sequences available for download somewhere. For draft genomes this is far from always the case. Anyway, even if annotations are available they have been made by different people at different labs at different times and using different types of software. For these reasons it may be preferable to predict genes from scratch in all genomes using the same approach in all cases.

There are several tools for prokaryotic gene prediction, and we have chosen the Prodigal software [3] as an option here. We are not suggesting this is perfect or universally best, but it gives fairly good results compared to all other tools we know of and is extremely fast and easy to use. The function `prodigalPredict` will invoke the Prodigal software directly from R. You need to provide as input the (full) name of the FASTA-file containing the genome sequence(s) and the (full) name of the file where you want the predicted proteins to be stored. You may also supply a third filename, and `prodigalPredict` will fill this file with the DNA-version of every coding gene. All output files are FASTA-files.

For each file in the `genomes/` subfolder we want to predict proteins and store these in a file with identical name, but in the `proteins/` subfolder:

```
for( i in 1:dim(genome.table)[1] ){
  cat("Predicting genes in", genome.table$File[i], "...\\n")
  in.file <- file.path("genomes",genome.table$File[i])
  out.file <- file.path("proteins",genome.table$File[i])
  prodigalPredict(in.file,out.file)
}
```

It is nice to have a short print-out inside the loop to monitor the progress, hence the `cat` statement. Upon completion of this code we should now have 7 FASTA-files in the `proteins/` subfolder, all containing valid amino acid sequences.

## 2.4 Preparing protein sequence files

**This subsection is important, and should not be skipped!**

Before we start to compare protein sequences we should make certain the sequence data files contain all the relevant information, and nothing more. The function `panPrep` will check all sequences in a FASTA-file to

- Convert all symbol to upper-case
- Discard protein sequences shorter than 10 amino acids long
- Replace any symbol not among the 20 single-letter amino acid alphabet symbols by **X**

We may also filter out sequences having a specified text/regular expression in their header, see `?panPrep` for details.

In Table 1 there is a column named GID. GID is short for Genome IDentifier, and is an integer which is unique for each genome in the study. By pasting the text "GID" and the integer we get a *tag* which is uniquely associated with every genome. This GID-tag must be present in the header of every sequence from the respective genome, and also in the name of the files associated with the genome. Whenever we start to compare sequences, it is imperative that each sequence has a name or description that immediately and without any redundancy links it to its genome.

In the `data/` folder we create another subfolder named `prepped/`. We want to read each file in the `proteins/` subfolder, prepare them, and write the results to a file with identical name in the `prepped/` subfolder:

```

for( i in 1:dim(genome.table)[1] ){
  cat("Preparing", genome.table$File[i], "...\\n")
  in.file <- file.path("proteins",genome.table$File[i])
  gid <- genome.table$GID[i]
  out.file <- file.path("prepped",genome.table$File[i])
  panPrep(in.file,gid,out.file)
}

```

The `prepped/` subfolder should now have files with identical names to those of the `proteins/` subfolder, except for the added GID-tag. If we open these FASTA-files we will also see that the header of the first sequence in the file with GID-tag `GID1` starts with the token `>GID1_seq1`, the second header starts with `>GID1_seq2` and so on. Hence, each sequence has a tag that links it to the genome (the GID part) and is also unique for each sequence in the genome (the `seq` part).

The GID-tag in the filenames and in the header of every sequences is absolutely essential for the later computations. The GID can be any integer, as long as it is unique to each genome in the study. It must be an integer such that the GID-tag matches the regular expression `GID[0-9]+` in R (e.g. `GID1`, `GID123`, `GID1000` are valid GID-tags).

In this case study we now have 3 subfolders under `Mpneumoniae/data/` (`genomes/`, `proteins/` and `prepped/`), all containing 7 FASTA-files. In all downstream analyses we will use the files in the `prepped/` folder.

### 3 Comparing sequences

Pangenomics involves the grouping of sequences into clusters, often referred to as *gene families*. We will prefer the term *clusters* here, since a gene family may have more specific interpretations (group of orthologs).

#### 3.1 Direct comparison using BLAST

One way of comparing sequences is to look at pairwise alignments between all possible sequence pairs. This may sound like a daunting exercise, and cannot in general be pursued for a larger data set. However, for a small to medium set of genomes it is a useful approach.

The BLAST software (<http://blast.ncbi.nlm.nih.gov>) is a standard tool for computing pairwise local alignments. Due to its heuristics it runs extremely fast. The main reason for this speed is that BLAST does not compute alignment between all pairs, only between those with a minimum similarity. However, this is good enough for our purpose, since we are only interested in the cases where sequences are fairly similar (the distinction between "very poor" and "extremely poor" similarity is without interest).

We prefer to take all proteins from one genome and turn it into a BLAST database. Then we scan all proteins from each genome against this database and store the results on a file. Next, another genome is converted to a database, and the scanning is repeated until all genomes have been scanned against all. From each result file we collect the largest score involving each listed pair of sequences. Those sequence pairs who are *not* listed in the results are too dissimilar to have

a BLAST alignment. These will implicitly get the score 0.0. It should be noted that we need to scan both genome A versus genome B and then genome B versus genome A since the heuristics of BLAST may give slightly different results for these two cases. We also need to scan genome A versus genome A, since different proteins within the same genome may also belong to the same cluster (paralogs). Thus, if we have  $G$  genomes, we will have to make  $G^2$  BLAST-scans.

Before we start the heavy computations we create a subfolder `blast/` under the `Mpneumoniae/` root folder. This is where we will store the results of the BLAST-scans. We set `Mpneumoniae/` as our R working folder.

The function `blastAllAll` will perform the all-against-all BLAST-scans of the genomes. Its first input must be a vector of filenames listing the (full) name of all the prepped protein files to consider. The second input must be the (full) path to the folder where we want the results to end up. It looks like this in our case:

```
in.files <- file.path("data/prepped",dir("data/prepped"))
out.folder <- "blast"
blastAllAll(in.files,out.folder)
```

From each protein file a BLAST database is constructed, and then all protein files are scanned against this database. The function gives a small output to monitor the progress. Depending on the number of proteins in the genomes, each scan takes from some seconds to some minutes on an average laptop computer.

The result files are plain text files. The names of these files have the format `GIDx_vs_GIDy.txt`, where  $x$  and  $y$  are integers. In our case study, having 7 genomes, there are 49 result files when `blastAllAll` is done. You can read a result file into R using the `readBlastTable` function.

If you have a look at the Help-file for `blastAllAll` you will see how you can speed up the computations by running several scans in parallel on a computer with multiple cores/processors. To accomplish this `blastAllAll` will never overwrite an existing result file in the `out.folder`. Thus, if you want to re-compute all results you must always remember to first delete the existing result files, or simply choose another `out.folder`.

## 3.2 Indirect comparison using HMMER3

Instead of comparing sequences directly by pairwise alignment, we can compare the sequences to a common set of *references*, and then compute their similarity based on how they match this reference. The advantages of this approach can be several, but an obvious one is that the workload scales linearly in the number of genomes, not quadratically as for the direct comparisons.

The reference we focus on here is a set of profile Hidden Markov Models (pHMM) each describing a family of sequences. We will use the Pfam-A database (<http://pfam.janelia.org/>) as reference, and each pHMM describes a conserved and functional part of a protein. We will call them *domains*, even if this is not strictly correct in all cases.

It is worth mentioning that the reference set could in principle be any collection of sequences or sequence models. For the comparisons to be of good value, we need to choose a reference set which is comprehensive and relevant for the proteins we are investigating. The Pfam-A database is relevant for all types of

proteins, since it is a collection of protein domains found in all branches of the tree of life. At present it contains around 13 000 pHMMs. You should, however, be aware that not all proteins listed for a genome will contain domains listed in Pfam-A (or any other domain database). There are several reasons for this. First, not all domains have been discovered, or they have been described too badly to be recognized in a given sequence. Second, a predicted protein in your genome may be a false positive and not a real protein at all. In the cases where a protein in your genome has no similarity to any of the reference sequences/-models, this sequence must be discarded from the downstream analysis since it will be incomparable to any other sequence. Whether this is a problem or an advantage depends on the reason for the lack of similarity as listed above. If your protein is actually a false positive, it *should* be discarded.

The scanning of sequences against a pHMM is done by the HMMER3 software (<http://hmmerr.janelia.org/>). Even if this software has been optimized for speed, it still takes quite some time to scan all proteins in a genome against the full Pfam-A database. Reducing the size of the reference, e.g. using a selection of pHMMs from Pfam-A, will speed up the scan, but for this study we will scan against the full Pfam-A database. In order to repeat the steps below on your computer you need to install the Pfam-A database, see <http://pfam.janelia.org> for how to do this. (In the `micropan` package we have enclosed a miniature version of Pfam-A, see `?hmmerrScan` for details).

Under the `Mpneumoniae` folder we create the subfolder `pfam/`, where we will put the results of the HMMER3-scan. The function `hmmerrScan` will take as first input a vector of (full) filenames of the protein files to scan. The second input is the (full) name of the database and the third input is the (full) name of the folder where the results should be stored:

```
in.files <- file.path("data/prepped",dir("data/prepped"))
db <- "/usr/share/pfam/Pfam-A.hmm" # edit this path
out.folder <- "pfam"
hmmerrScan(in.files,db,out.folder)
```

where the exact name and location (`/usr/share/pfam/Pfam-A.hmm`) of the Pfam-A database will of course vary from system to system. The function `hmmerrScan` gives a small output to monitor the progress. When running this example on a laptop it took around 5 minutes per genome.

The result files are plain text files, and they are stored in the `pfam/` folder. The names of these files have the format `GIDx_vs_Pfam-A.hmm.txt`, where `x` is an integer. In our case study, having 7 genomes, there are 7 result files when `hmmerrScan` is done. You can read a result file into R using the `readHmmer` function.

If you have a look at the Help-file for `hmmerrScan` you will see how you can speed up the computations by running several scans in parallel on a computer with multiple cores/processors. To accomplish this `hmmerrScan` will never overwrite an existing result file in the `out.folder`. Thus, if you want to re-compute all results you must always remember to first delete the existing result files, or simply choose another `out.folder`.



## 4 Clustering sequences

Depending on how the sequences have been compared, we can now cluster them.

### 4.1 Clustering based on BLAST results

#### 4.1.1 BLAST distances

Based on the results from the all-versus-all BLASTing, we can compute distances between sequences. Let  $S(i; j)$  be the score of the alignment between sequence  $i$  and  $j$ , where  $j$  was the database sequence. If there is no BLAST hit between these two sequences, this score is 0.0. The maximum value this score can take is  $S(j; j)$ , i.e. the alignment between sequence  $j$  and itself. The ratio  $S(i; j)/S(j; j)$  must always be a number between 0.0 and 1.0. Due to the heuristics of BLAST we also made the reciprocal scan, and can compute  $S(j; i)/S(i; i)$  as well. The distance between sequence  $i$  and  $j$  we define as

$$D(i, j) = \frac{1}{2} [2 - S(i; j)/S(j; j) - S(j; i)/S(i; i)] \quad (1)$$

This distance is 0.0 if and only if sequence  $i$  and  $j$  are identical. The maximum possible distance is 1.0 indicating the sequences have no detectable similarity.

The function `bDist` will take as input a vector of filenames listing all the result files from the BLAST-scan (see above), read these files and compute distances according to (1). It will return a `data.frame` where each row corresponds to a sequence pair. The two first columns contain the sequence tags and the third column is the distance between them. Only sequence pairs having BLAST alignments are listed, i.e. all those pairs *not* listed in this `data.frame` have distance 1.0 between them. This is how we use this function in our case study:

```
blast.files <- file.path("blast", dir("blast"))
blast.distances <- bDist(blast.files)
save(blast.distances, file="res/blast_distances.RData")
```

Notice that all BLAST result files (in the `blast` subfolder) must be given as input to `bDist`. Notice also that we save the results in a subfolder named `res`. In cases where we have many genomes, and thus many result files, the reading of results takes some time, and it is convenient to store the results instead of repeating the whole procedure in a later session.

The variable `blast.distances` is now a `data.frame` with 3 columns and 139543 rows. The third column contains the distances, and it is always a good idea to make a histogram of these distances to verify that it looks reasonable:

```
hist(blast.distances[,3], breaks=50, col="tan4",
      xlab="BLAST distance", ylab="Number of distances")
```

The resulting histogram is shown in Figure 1. Notice there is a large number of sequence pairs having distance close to 0.0. Actually, even more pairs have distance 1.0 but these are never listed in this `data.frame`. The shape of this histogram will vary somewhat from study to study, but there should always be a

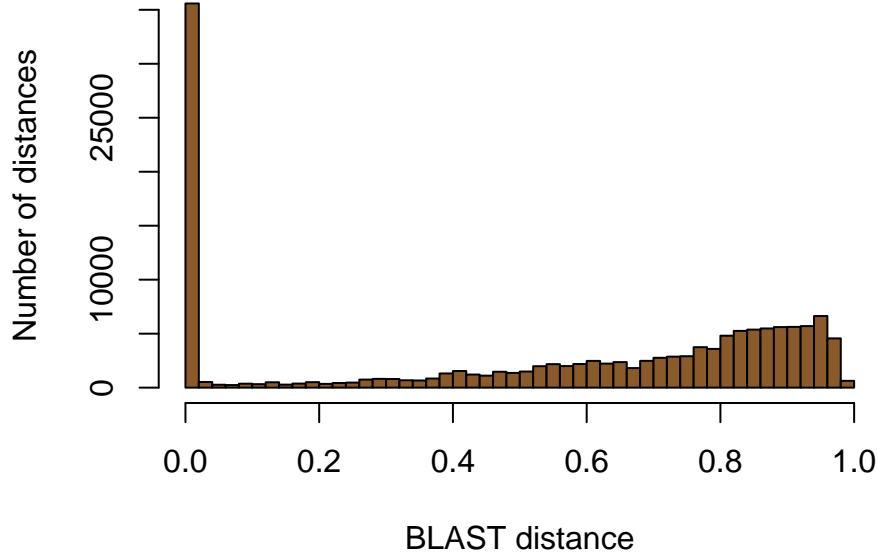


Figure 1: The histogram shows the BLAST distances computed for this case study.

large number of very small distances. If not, it means all proteins in all genomes are quite different, which is really strange for a pangenome.

#### 4.1.2 Hierarchical clustering

The function `bClust` will cluster the sequences based on the `data.frame` produced above by `bDist`. It performs a hierarchical clustering of the sequences, assuming all pairs not listed have distance 1.0 between them. Central to any hierarchical clustering is the *linkage* function and the *threshold* we choose for grouping the sequences. The linkage specifies how to compute the distance between clusters, the threshold specifies the maximum distances we tolerate inside the cluster. Notice that the threshold is here a BLAST distance, always a number between 0.0 and 1.0.

The default linkage is the *single linkage*. In this case the distance between group A and B is the shortest distance between any member from A to any member in B. The threshold  $T$  means that sequence  $s$  can belong to group A if there is at least one member in A with distance smaller than  $T$  to  $s$ . The single linkage grouping is the fastest to compute, but has a tendency of producing large and heterogenous clusters. Two very different sequence can belong to the same cluster simply because one of them looks like something, that looks like something,...,that looks like something, thats looks like the other. If no distance

in this chain is above  $T$  it could create a cluster. The default input to `bClust` is to use single linkage and a threshold at 1.0, which will produce the largest (and fewest) clusters possible for the given data.

The *average linkage* means that the distance between group A and B is the distance between the center of each group. Also, a sequence  $s$  can belong to group A if its distance to the center of A is less than the threshold  $T$ . In some sense, the threshold now specifies the 'radius' of the group, and there is a limit to how dissimilar two sequences of the same group can be. A potential problem with this linkage can occur if some of the genomes in the study are extremely closely related. Then the members (proteins) from these genomes are very similar, and will make the center of every group be biased towards their 'corner' of the pangenome. Then some proteins who should have been member of a group may fall outside.

The *complete linkage* means that the distance between group A and B is the largest possible distance between a member from A and a member from B. A sequence  $s$  can belong to group A if and only if its distance to all other members of the group is below  $T$ . This means the threshold  $T$  directly specifies the maximum 'diameter' of the groups. It also means all groups will be homogenous, no sequence in a cluster is very different from any other. The problem with this strict regime is of course that some sequences always fall outside all clusters, and we get a larger number of ORFan (singleton) clusters. Choosing the complete linkage with a strict (small) threshold is the oppsite strategy of the default, and produces many small clusters instead of a few large.

Here we have chosen to use a complete linkage and the rather liberal threshold of  $T = 0.75$  to cluster the sequences:

```
cluster.blast <- bClust(blast.distances, linkage="complete",
  threshold=0.75)
```

The output, named `cluster.blast` here, is simply a vector of integers, with one element for each sequence in the data set. The **name** of each element identifies the sequence, and two sequences having the same integer belongs to the same cluster. In this case study `cluster.blast` has 9573 elements since this is the total number of protein sequences in the seven genomes. We can quickly see how many clusters we got:

```
> length(unique(cluster.blast))
[1] 1210
```

If we look at the first seven elements they are

```
> cluster.blast[1:7]
GID1_seq1    GID2_seq1    GID3_seq1    GID4_seq1    GID5_seq407
      2000      2000      2000      2000      2000

GID6_seq1164  GID7_seq1174
      2000      2000
```

All have the value 2000 indicating they belong to the same cluster. From the name above each integer we see there is exactly one member from each of the seven genomes, i.e. this is a perfect example of a *core cluster* with one ortholog from each genome. Notice it is sequence number 1 in the first four (completed) genomes, indicating it is just downstream of the replication start. In the draft genomes the contigs are un-ordered and the sequence number is pretty random. The actual number that indicates the clustering (the value 2000 above) has no meaning at all, it is just a marker.

## 4.2 Clustering based on Pfam domains

An alternative to the clusters produced by the direct comparison BLASTing approach, is a clustering based on the Pfam-A domains. This approach has been used for pangenomics by [8], and we describe a similar procedure here.

### 4.2.1 Domain sequences

In the previous section we scanned all proteins of every genome against the Pfam-A database using the HMMER3 software and the `hmmesScan` function. Every protein with at least one hit in the Pfam-A database can be described by the sequence of domains occurring along its length. We can think of this as a high-level alternative to the amino acid sequence. Instead of listing the sequential occurrence of amino acids, we list the sequential occurrence of Pfam-A domains. We call this the *domain sequence* of the protein. Many proteins will have only one single domain, but still we call it a domain sequence. In the cases where we have multiple domains in a protein, we only list those who are non-overlapping, and their order of appearance is essential. A number of proteins will have no Pfam-A hits. These sequences are discarded from this analysis. This may sound like a loss of information, but on the other hand, some of these sequences are likely to be false positive gene predictions anyway.

### 4.2.2 Clustering

We cluster the proteins by their domain sequence, i.e. only proteins having identical domain sequence belong to the same cluster. This may seem like a strict rule, but in fact it is not. The reason is that the pHMMs describing the domains allow a considerable degree of variation in the protein sequence matching the model, and two protein may appear quite different and still share the same domains.

To compute the domain sequence clusters we first have to read the results of the HMMER3 scan against the Pfam-A database:

```

pfam.files <- file.path("pfam",dir("pfam"))
pfam.table <- NULL
for(i in 1:length(pfam.files)){
  tab <- readHmmer(pfam.files[i])
  tab <- hmmerCleanOverlap(tab)
  pfam.table <- rbind(pfam.table,tab)
}

```

The function `readHmmer` reads the result file and returns it as a `data.frame`. The function `hmmerCleanOverlap` is used here to filter out overlapping hits from this table. If two hits overlap on the same protein, the poorest hit (smallest score) is discarded. You may omit this step, but we prefer to include it. Notice that we read the results for each genome and store everything in one large `data.frame`. This can be saved for later. If we want to add more genomes to this analysis, we scan their proteins against Pfam-A, read the results, and just add these to the existing `data.frame` without repeating any of the previous work.

Once we have the table of results, the clustering is straightforward:

```
cluster.pfam <- dClust(pfam.table)
```

where the `dClust` function produces the domain sequence clustering. The result `cluster.pfam` is a vector of integers similar to the one we get from `bClust`. Each integer corresponds to a sequence identified by its `name`, and those sequences having the same integer belong to the same cluster. The number of clusters is

```

> length(unique(cluster.pfam))
[1] 445

```

which is down to almost one third of what we got with the BLAST approach. Again we can look at the first entries:

```

> cluster.pfam[1:7]
GID1_seq1126 GID2_seq1108 GID3_seq1110 GID4_seq1119
              1              1              1              1

GID5_seq110  GID6_seq526  GID7_seq934
              1              1              1

```

and we notice this cluster, with 7 members, has one member from each genome, i.e. it is a core cluster.

The clustering vector also contains an attribute named `"cluster.info"`. This holds the actual Pfam-A domains behind each group. The cluster above has the marker value 1. If we look at element number 1 in this vector we get:

```

> attr(cluster.pfam,"cluster.info")[1]
[1] "PF00004.24"

```

Only one Pfam-A accession number is listed. This means that the core cluster above contains the 7 proteins all sharing one single domain. If the domain sequence contains several domains, they are listed in their order of appearance separated by comma. If we look up the Pfam-A accession number PF00004.24, it is described as *ATPase family associated with various cellular activities*. In this way we can get some kind of description of each cluster, not only its content.

### 4.3 Direct or indirect comparison?

We have seen two ways of clustering sequences. Which one is better? There are pros and cons of both.

The direct approach tend to give smaller clusters, but this depends on the choice of linkage and threshold. With a complete linkage and a reasonable threshold many of the groups will be very close to a gene family in the sense that it contains a group of orthologs. The problem is to choose the proper threshold. It is also the same threshold for all clusters. We can easily imagine that some gene families are more divergent than others, and that different thresholds should ideally be used. Also, all false positive gene predictions will tend to produce many small clusters, usually singletons (ORFans). A problem with the direct comparison is that it scales quadratically. It is suitable for 50 genomes, possibly for 100, but horrible for 1000 genomes.

The indirect approach using the Pfam-A database produces fewer and larger groups. Many of these groups are much larger than just a group of orthologs. In this case study the largest group contains 167 proteins, i.e. on average more than 20 in each genome! These proteins share a single domain described only as *Lipoprotein* in the Pfam-A database. Still, with this in mind this clustering says something important about the pangenome. Each domain sequence is associated with some potential *function*, and the number of such groups and their sizes has biological interest. It is a descriptor of functional diversity. Another point is that each pHMM in Pfam-A describes a tolerated variation around the consensus pattern. This means that different clusters actually have different 'spread' and in many ways the problem of different divergence in different protein families is solved by this approach. The indirect comparison also scales linearly, and adding a genome to the analysis will take the same computational effort each time, regardless of how many you have collected before.

## 5 The pan-matrix

Having the clustering (in one way or another) we can construct the pan-matrix, which is the central data structure in a pangenome study.

The pan-matrix is a matrix with one row for each genome and one column for each cluster. Cell  $(i, j)$  in this matrix contains an integer indicating the number members that cluster  $j$  has in genome  $i$ . The function `panMatrix` computes the pan-matrix from a clustering vector. We compute two pan-matrices, based on both clustering procedures described above:

```
pm.blast <- panMatrix(cluster.blast)
pm.pfam  <- panMatrix(cluster.pfam)
```

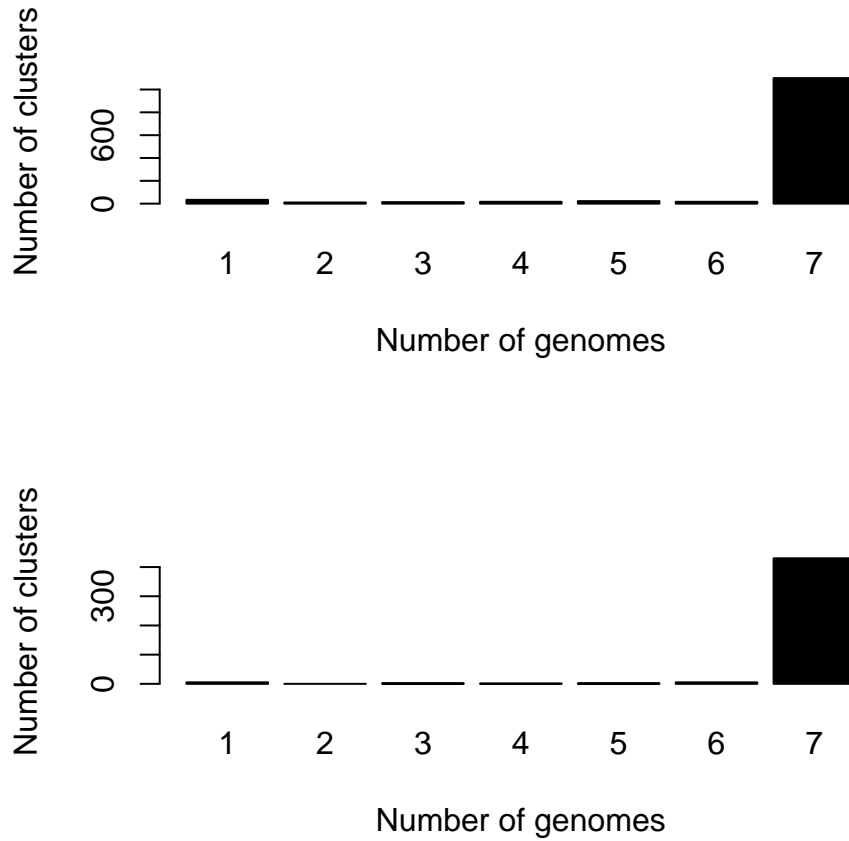


Figure 2: The pan-matrices plotted by their generic plotting function. The upper panel shows `pm.blast` and the lower panel `pm.pfam`. The sum of bar-heights is 1210 in the upper panel and 445 in the lower panel.

Both will have 7 rows, but `pm.blast` has 1210 columns while `pm.pfam` has 445.

The `panMatrix` function returns a `Panmat` object, which is a small extension to a matrix. It has a generic plotting function. This will produce a bar-plot of how many clusters are found present in 1,2,...,all genomes in the data set. The code:

```
par(mfrow=c(2,1))
plot(pm.blast)
plot(pm.pfam)
```

produces a figure like the one shown in Figure 2. We observe that in both cases the core clusters dominate, i.e. those who have members in all 7 genomes. These 7 genomes must be very similar, and only a small number of clusters have an ability to separate the genomes. The actual numbers behind the barplots are

given by the generic `summary` function:

```
> summary(pm.blast)
33 clusters found in 1 genomes
10 clusters found in 2 genomes
13 clusters found in 3 genomes
16 clusters found in 4 genomes
22 clusters found in 5 genomes
16 clusters found in 6 genomes
1100 clusters found in 7 genomes

> summary(pm.pfam)
5 clusters found in 1 genomes
0 clusters found in 2 genomes
2 clusters found in 3 genomes
1 clusters found in 4 genomes
2 clusters found in 5 genomes
5 clusters found in 6 genomes
430 clusters found in 7 genomes
```

## 6 The pangenome tree

Based on the pan-matrix we can make a pangenome tree as described in [7]. First we compute the distance between genomes using the `distManhattan` function. This takes a pan-matrix as input, and produces a `dist` object, see `?distManhattan` for details.

```
par(mfrow=c(1,2))
D <- distManhattan(pm.blast)
panTree(D)
D <- distManhattan(pm.pfam)
panTree(D)
```

The `panTree` function will perform an average linkage hierarchical clustering based on the input distances, and draw a simple dendrogram tree. In Figure 3 we show the result of this code. The horizontal axes is a Manhattan distance (since we used `distManhattan`). In this case it is simply the number of clusters in which the genomes differ in present/absent status, i.e. if a distance between two genomes is 0 it means they contain the exact same clusters. Note that the distance between groups (branches) is an average linkage distance, i.e. distance to the 'average' member of the branch.

From these trees we see that genomes GID1 and GID4 are very similar, for the domain sequence tree in the right panel they are actually identical. Also, GID6 is the genome which is most different from the rest.

The GID-tag is not a very informative label to have in the tree, and we may also want to add some coloring of the genomes depending on our prior knowledge of them. In the `genome.table` that we created in the very beginning of this case study, we have such information linked to the GID-tags. Here is some code that produces a nicer version of the BLAST-based pangenome tree from above:



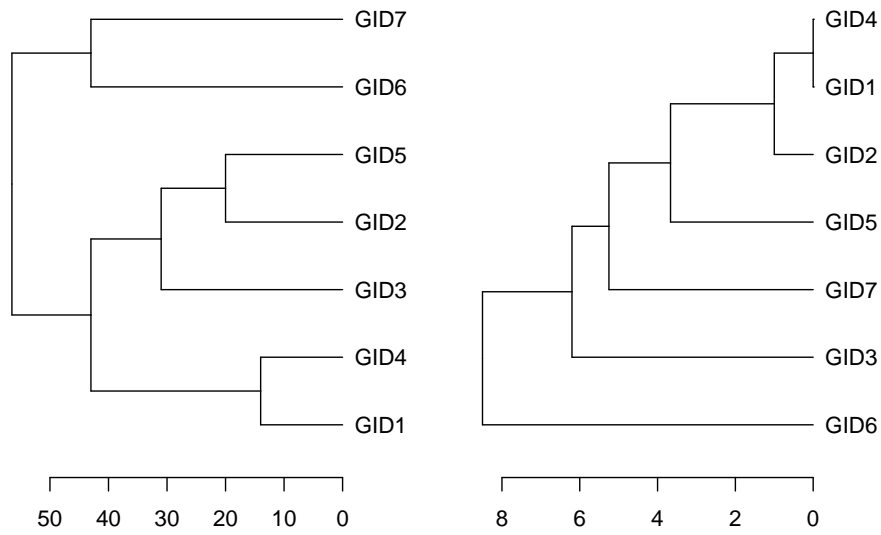


Figure 3: The pangenome trees based on the pan-matrices. The left panel shows the tree based on `pm.blast` and in the right panel it is based on `pm.pfam`. The horizontal axes are the Manhattan distances between (groups of) genomes.

### Pangenome tree for *Mycoplasma pneumoniae*

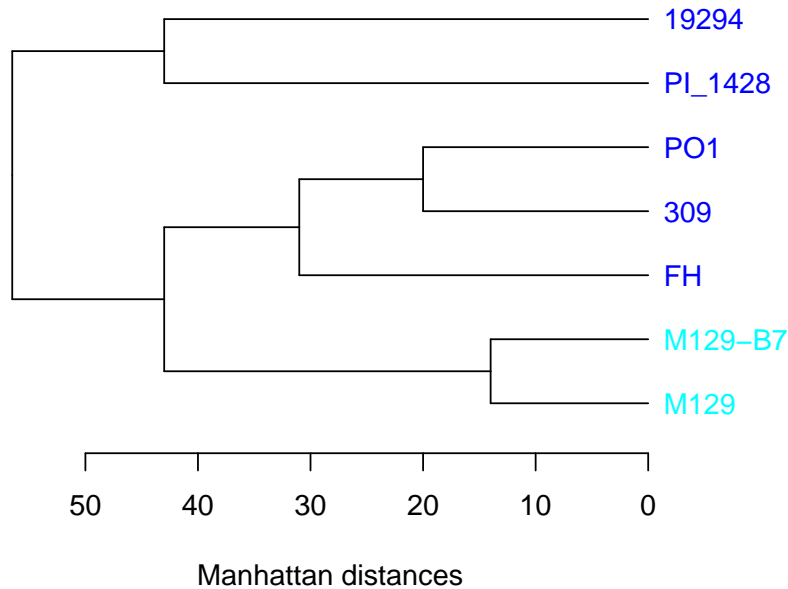


Figure 4: The BLAST-based pangenome tree with more informative labels and colors.

```
D <- distManhattan(pm.blast)
idx <- match(rownames(pm.blast), paste("GID", genome.table$GID, sep=" "))
panTree(D, genome.names=genome.table$Strain[idx], genome.col=
  genome.table$Color[idx], xlab="Manhattan distances", main="
  Pangenome tree for Mycoplasma pneumoniae")
```

The result is shown in Figure 4. Notice that we computed the index vector `idx` and used it to order the strain names and colors in the call to `panTree`. The genomes are listed in a specific order in the pan-matrix and the corresponding `dist` object named `D`. Never expect this ordering to be the same as in `genome.table`! The rownames of the pan-matrix contains the GID-tags, and this is used to map into the `genome.table`.

## 7 Pangenome size

The pangenome size is the number of clusters to be found in the population if all strains were sequenced. This number will never be observed, and we have to try to estimate it from the present data. With only 7 genomes as in this case study, this estimate is bound to be highly uncertain, but we can still show the procedure. The core size is the number of core clusters in the population, i.e. the number of clusters found in every single strain, and must also be estimated.

## 7.1 Binomial mixture models

Both quantities can be estimated by a probabilistic approach using binomial mixture models as described in [2, 6]:

```
binomix <- binomixEstimate(pm.blast,K.range=2:7)
```

The `binomixEstimate` function takes as input a pan-matrix and a vector of integers, `K.range`, specifying the number of components to try out. A binomial mixture model is fitted for each number in `K.range`. For each model is also computed the BIC-criterion, and the model producing the smallest BIC-value is in theory the one that describes the data best without overfitting. The function returns a `Binomix` object, which is a small extension to a `list` of two elements. First we only focus on the one called `BIC.table`:

```
> binomix$BIC.table
```

	Core.size	Pan.size	BIC
2 components	1099	1211	1227.697
3 components	1096	1267	1151.008
4 components	1092	1281	1164.656
5 components	1094	1340	1178.482
6 components	1020	1339	1192.698
7 components	1090	1303	1206.967

This table lists some results for each fitted model. The last column is the BIC-value, and we notice that the 3-component model in the second row is optimal, having the smallest BIC-value (1151.008). Thus, we focus only on this row. The `Core.size` column give us the estimate of the core size as 1096 and the `Pan.size` estimate is 1267. We used the BLAST-based pan-matrix as input, and this contained 1210 clusters already observed. The estimate of pangenome size indicates that only a small number of additional clusters (57) is expected to be found if more genomes of this population is sequenced.

We can get a summary of the `Binomix` object:

```
> summary(binomix)
```

Minimum BIC model at 3 components

For this model:

8% of the gene clusters have detection probability 0.074

6% of the gene clusters have detection probability 0.665

86% of the gene clusters have detection probability 1

The percentages and probabilities listed here are found in `binomix$Mix.list[[2]]`. The `Mix.list` is a `list` of fitted mixture models (one for each component), and since the minimum *BIC* is achieved for the second model, this is the one listed by `summary`.

The optimal model has 3 components, i.e. it describes the observed data as a combination of 3 binomial densities each having a distinct detection probability. The detection probability of a gene cluster is the probability that it occurs in a genome. Gene clusters having detection probability 1.0 will always occur in all genomes. These are the core gene clusters. In the fitted model above these

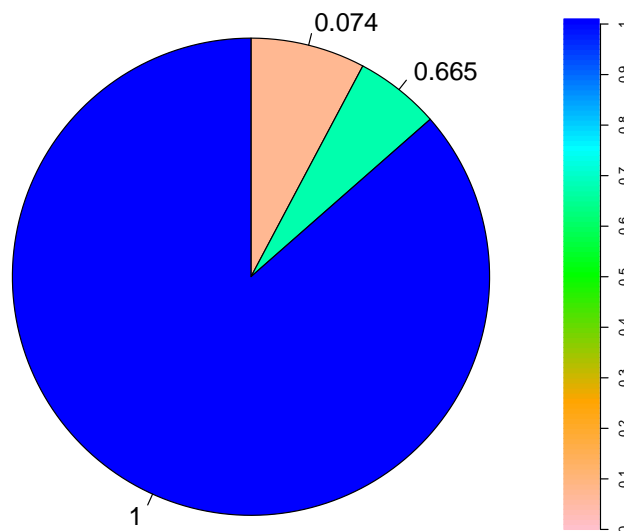


Figure 5: A fitted binomial mixture model can be displayed as a pie chart. Each sector corresponds to a binomial density. Its color indicates its detection probability. Its size indicates how large fraction of the pangenome will have this detection probability. The colors have been chosen to illustrate core genes (dark blue), shell genes (greenish) and cloud genes (orange/pink).

make up 86% of all gene clusters. In addition to the core there is a smaller fraction (6%) of gene clusters occurring frequently, but not always, having a probability of 0.665 of being observed. This is what we typically denote *shell* genes. Finally, there is also a small percentage (8%) of gene clusters with a very small detection probability (0.074), usually denoted *cloud* genes. These are the clusters that are more or less unique to each genome, and the majority of the new clusters we expect to see in the future will be of this type.

We can also plot the `Binomix` object:

```
> plot(binomix)
```

The result is shown in Figure 5.

## 7.2 Other analyses

It is not uncommon to plot the number of gene clusters as a rarefaction curve, i.e. the cumulative number of unique clusters as we consider more and more genomes. The function `rarefaction` will produce a corresponding object, and the generic `summary` and `plot` functions for these objects produce text and graphical output. Here we make a plot:

```
> r <- rarefaction(pm.blast,n.perm=100)
> plot(r)
```

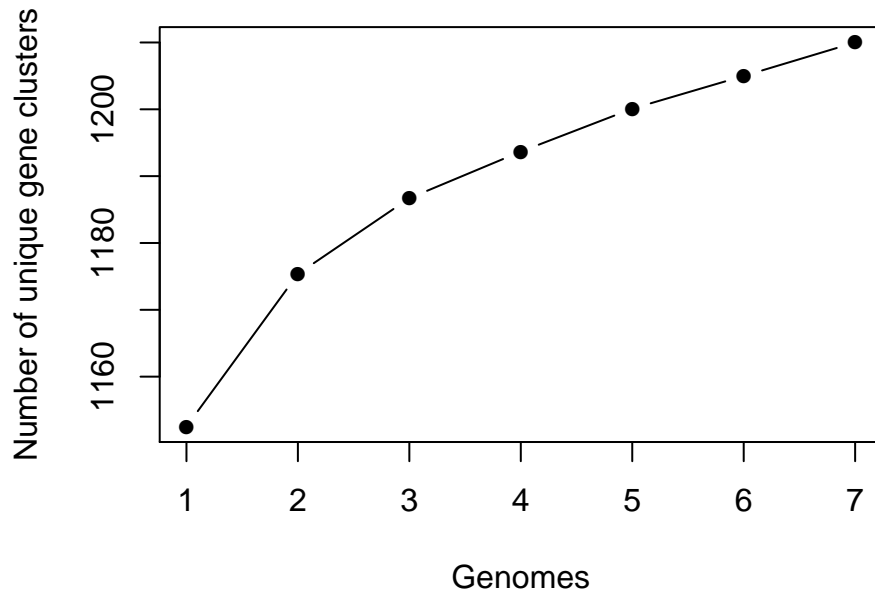


Figure 6: A rarefaction curve for the seven genomes in this case study.

and the result is shown in Figure 6. Notice that we specified `n.perm=100` above. The shape of the curve in Figure 6 depends on the ordering of the genomes, and here we average over 100 random permutations. If you specify `n.perm=1` the ordering will be as in the pan-matrix (row 1 to row N).

It was suggested by [9] that a pangenome can be classified as *open* or *closed* depending on the shape of the rarefaction curve. They used a heaps-law type of model that we can fit to the data. One of the parameters in this model, named `alpha`, is the indicator we look at. If its value is below 1.0 the pangenome is classified as open. This means regardless of how many new genomes we sequence, there will always be new unique gene clusters, i.e. the rarefaction curve will never level out. If `alpha` is larger than 1.0 the pangenome is closed and the shape of the rarefaction curve indicates it will reach a plateau. The function `heaps` fits a heaps-law model to the data:

```
> h <- heaps(pm.blast, n.perm=100)
> h
Intercept      alpha
64.323844    1.479499
```

Only `alpha` is of interest here, and as we can see this is clearly above 1.0 indicating a closed pangenome. Notice that we again used 100 permutations of the genome-ordering. This is a minimum, to obtain stable estimates use as

many as possible, limited by a reasonable computing time.

The size of the pangenome was estimated by a binomial mixture model above. As an alternative, we can use the Chao lower bound [1] estimator:

```
> pan.size <- chao(pm.blast)
> pan.size
[1] 1264
```

Compared to the estimate of the binomial mixture model (1267) it is slightly lower, which is quite typical since this is a lower bound estimator.

If we turn the focus away from the gene clusters and to the genomes, a pangenome tree as displayed in Figure 3 is an illustration of the relations. In [4] it was suggested that *genomic fluidity* is a quantity that characterizes the population. This can be computed as:

```
> f <- fluidity(pm.blast, n.sim=100)
> f
$Mean
[1] 0.0183972

$Std
[1] 0.00680975
```

where both the mean value and its standard deviation is computed. The fluidity is always a value between 0.0 and 1.0. A large fluidity means a larger diversity between the genomes. This measure only looks at presence/absence of unique gene clusters. It is computed from many random pairs of genomes, and `n.sim` is the number of pairs to consider. A smallish number suffice, and `n.sim=100` is enough in most cases.

For genomes  $A$  and  $B$ , let  $A$  and  $B$  symbolize the sets of gene clusters in the two genomes, respectively. Then genomic fluidity between  $A$  and  $B$  is defined as

$$F(A, B) = \frac{|A \cup B| - |A \cap B|}{|A| + |B|} \quad (2)$$

where the numerator is the number of gene clusters found in  $A$  but not in  $B$  plus those found in  $B$  but not in  $A$ . The denominator is simply the sum of the size of each set. The classical *Jaccard distance* is defined as

$$J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \quad (3)$$

which means the only difference to the fluidity is the slightly different denominator. Hence, computing Jaccard distances and investigating their distribution may be a good alternative to the genomic fluidity:

```
> J <- distJaccard(pm.blast)
> mean(J)
[1] 0.03896226
```

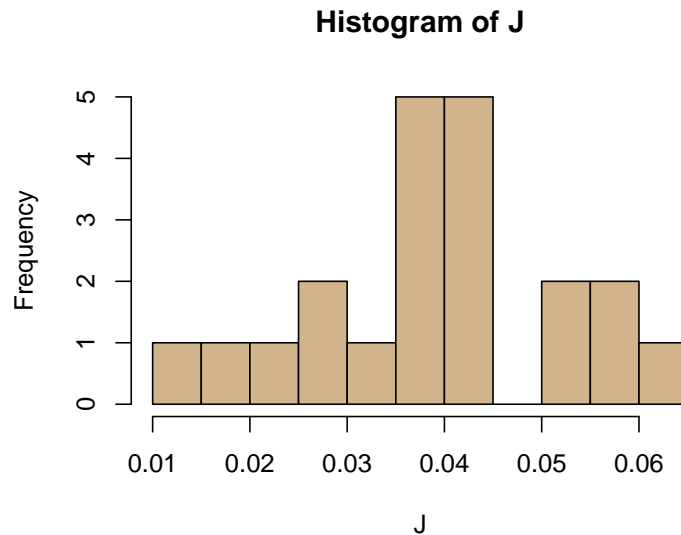


Figure 7: A histogram of the Jaccard distances between the genomes.

The mean Jaccard distance of around 4% means that out of the total number of unique gene clusters two 'average' genomes have  $(|A \cup B|)$ , around 4% is unique to one of them, the remaining 96% is shared by both. A histogram of these distances:

```
> hist(J,breaks=10,col="tan")
```

tells us that the variation in distances is fairly small, and even the largest pairwise distance is not very different from the average, see Figure 7.

## References

- [1] Chao, A. (1987). Estimating the population size for capture-recapture data with unequal catchability, *Biometrics*, **43**, pp783-791.
- [2] Hogg, J.S., Hu, F.Z., Janto, B., Boissy, R., Hayes, J., Keefe, R., Post, J.C., Erlich, G.D. (2007). Characterization and modelling of the *Haemophilus influenzae* core- and supra-genomes based on the complete genomic sequences of Rd and 12 clinical nontypeable strains. *Genome Biology*, **8**.
- [3] Hyatt, D., Chen, G., LoCascio, P.F., Land, M.L., Larimer, F.W., Hauser, L.J. (2009). Prodigal: prokaryotic gene recognition and translation initiation site identification. *BMC Bioinformatics*, **11**:119.
- [4] Kislyuk, A.O., Haegeman, B., Bergman, N.H., Weitz, J.S. (2011). Genomic fluidity: an integrative view of gene diversity within microbial populations. *BMC Genomics*, **12**:32.

- [5] <http://www.ncbi.nlm.nih.gov/genome>
- [6] Snipen, L., Almoe, T., Ussery, D.W. (2009). Microbial comparative pan-genomics using binomial mixture models. *BMC Genomics*, **10**:385.
- [7] Snipen, L., Ussery, D.W. (2010). Standard operating procedure for computing pangenome trees. *Standards in Genomic Sciences*, **2**, pp135-141.
- [8] Snipen, L. Ussery, D.W. (2012). A domain sequence approach to pangenomics: Applications to *Escherichia coli*. *F1000 Research*, **1**:19.
- [9] Tettelin, H., Riley, D., Cattuto, C., Medini, D. (2008). Comparative genomics: the bacterial pan-genome. *Current Opinions in Microbiology*, **12**, pp472-477.