Finding Chimeric Sequences

Erik S. Wright
University of Wisconsin
Madison, WI

August 29, 2015

Contents

1 Introduction **1**

2 Chimera Finding Problems **1**

3 Getting Started **2**

4 Obtaining/Building A Reference Database **2**

5 Steps to Find Chimeras **4**
5.1 Assigning Sequences To A Reference Group 4
5.1.1 Classifying Sequences 4
5.1.2 Determining Reference Groups 4
5.1.3 Assign to Groups 4
5.2 Forming the Sequence Database 5
5.2.1 Importing Sequences 5
5.2.2 Adding Group Names 6
5.3 Finding Chimeras 6

6 Session Information **7**

1 Introduction

This document illustrates how to detect chimeric sequences using the DECIPHER package through the use of the `FindChimeras` function. The function finds chimeric sequences present in a database (*dbFile*) of sequences by making use of a reference database (*dbFileReference*) of good sequences. The examples in this document are directed towards finding chimeric 16S rRNA sequences, although a similar strategy could be used with any type of sequence. For 16S sequences the functionality of `FindChimeras` is implemented as a web tool available at `http://DECIPHER.cee.wisc.edu`.

2 Chimera Finding Problems

Chimeras are a common type of sequence anomaly that result from the use of PCR amplification. Chimeras masquerade as normal sequences, but actually are made up of several different sequence parts that do not exist together in reality. Undetected chimeras can have insidious effects, such as causing incorrect assessments of sequence diversity or primer/probe specificity. Chimeras are very difficult to detect, and several detections

methods have been proposed to varying degrees of success. The basic problem can be described as the search for one or more regions of a sequence that do not belong with the rest of the sequence.

The `FindChimeras` algorithm works by finding suspect sequence fragments that are uncommon in the group where the sequence belongs, but very common in another group where the sequence does not belong. Each sequence in the *dbFile* is tiled into short sequence segments called fragments. If the fragments are infrequent in their respective group in the *dbFileReference* then they are considered suspect. If enough suspect fragments from a sequence meet the specified constraints then the sequence is flagged as a chimera.

3 Getting Started

To get started we need to load the DECIPHER package, which automatically loads several other required packages.

```
> library(DECIPHER)
```

Help for any DECIPHER function, such as `FindChimeras`, can be accessed through

```
> ? FindChimeras
```

If this vignette was installed on your system then the code in each example can be obtained by

```
> browseVignettes("DECIPHER")
```

4 Obtaining/Building A Reference Database

The first step necessary to use `FindChimeras` is to obtain a reference database (*dbFileReference*) of good sequences. In order for the algorithm to work accurately, it is crucial that the reference database is free of any chimeras. The examples given in this document use 16S rRNA sequences, and a suitable reference database can be found at `http://DECIPHER.cee.wisc.edu/Download.html`. The 16S reference database must be unzipped after download.

If a reference database is not present then it is feasible to create a one if a significantly large repository of sequences is available. By using the input database as the reference database it is possible to remove chimeras from the database. Iteratively repeating the process can eventually result in a clean reference database that is chimera free.

The reference database is required to have several fields of information that are not found in a newly created sequence database:

1. *bases* and *nonbases* as provided by the function `IdLengths`.

2. *origin* as provided by the function `FormGroups`. The origin column contains the taxonomic rank above the level shared by the whole group. For example, the origin of the group *Nanoarchaeum* would be *Root; Archaea; Nanoarchaeota*. Without the taxonomic rank it would be possible to substitute numbers. For example, the rank 3.1.4.2 with a group named 4 that included 4.1 and 4.2 would have origin 3.1. These numbers could be generated using `IdClusters` with multiple *cutoffs*.

Below is an example of building a reference database *de novo* from a FASTA file of LSU rRNA (23S) sequences downloaded from SILVA (http://www.arb-silva.de/). Of course, the reference database must match the type of the query sequences, so in the remainder of this vignette we use a comprehensive SSU rRNA (16S) database. The information in this section is only given as an example of how to build a reference database from scratch.

```
> dbConn <- dbConnect(SQLite(), "SILVA-117 23S Reference Database.sqlite")
> Seqs2DB("SILVA 23S RefDB.fasta", "FASTA", dbConn, "")

50675 total sequences in the table DNA.
Time difference of 192.56 secs

[1] 50675
```

When importing a GenBank file the rank (classification) information is automatically imported from the ORGANISM field. The classification is simply the levels of taxonomic rank separated by semicolons. In this case the classification must be parsed from each sequence's description information following the ">" symbol.

```
> class <- dbGetQuery(dbConn, "select description from DNA")
> f <- function (x) {paste(x[2:length(x)], collapse="")}
> class <- unlist(lapply(strsplit(class$description, " ", fixed=TRUE), f))
> Add2DB(data.frame(rank=class), dbConn)
Expression:  alter table DNA add column rank TEXT
Expression:  update or replace DNA set rank = :rank where row_names = :row_names

Added to table DNA:  "rank".
Time difference of 0.88 secs

[1] TRUE
```

We can now screen the reference database for chimeras. This process can be repeated until no new chimeras can be found in the reference database. After completing this process the reference database will be ready for checking new sequences.

```
> lengths <- IdLengths(dbConn, add2tbl=TRUE)
  |=============================================================| 100%
Lengths counted for 50675 sequences.
Added to DNA:  "bases", "nonbases", and "width".

> groups <- FormGroups(dbConn, add2tbl=TRUE)
  |=============================================================| 100%
Updating column: "id"...
Updating column: "origin"...
Formed 72 distinct groups.
Added to table DNA: "id", "origin".
Time difference of 917.76 secs

> chimeras <- FindChimeras(dbConn, dbFileReference=dbConn, add2tbl=TRUE)
Found 102 possible chimeras.
Added to table DNA:  "chimera".
Time difference of 13387.34 secs

> dbDisconnect(dbConn)
```

5 Steps to Find Chimeras

5.1 Assigning Sequences To A Reference Group

5.1.1 Classifying Sequences

In order to assign new sequences to a group in the reference database, it is necessary to first classify the sequences. There are a wide variety of classification methods that would work for this purpose. Here we make use of the RDP Multiclassifier available from `http://rdp.cme.msu.edu/classifier/`. After downloading the multiclassifier.jar we need to use it to classify our sequences. Be sure to change the path names to those on your system by replacing all of the text inside quotes labeled "<<path to ...>>" with the actual path on your system.

```
> output.file <- tempfile()
> command <- paste("java -Xmx1g -jar '",
 "<<path to multiclassifier.jar>>",
 "' --hier_outfile=/dev/null --conf=0 --assign_outfile='",
 output.file,
 "' '",
 "<<path to FASTA file>>",
 "'",
 sep="")
> cat("\n\n", command, "\n\n", sep="")
```

```
java -jar '/multiclassifier.jar' -hier_outfile=/dev/null -conf=0 -assign_outfile='/tmp/file1'
'/seqs.fas'
```

```
> system(command)
```

5.1.2 Determining Reference Groups

Next we need to determine which groups are present in the reference database and the number of sequences present in each group.

```
> dbRef <- dbConnect(SQLite(), '<<path to reference database>>')
> t <- dbGetQuery(dbRef, "select distinct rank, id from DNA group by rank")
> y <- dbGetQuery(dbRef, paste("select distinct id,
         count(*) as count from DNA", "where chimera is null group by id"))
> t$count <- 0
> for (i in 1:dim(t)[1])
 t$count[i] <- y$count[which(y$id==t$id[i])]
```

5.1.3 Assign to Groups

We can now assign each of the classified sequences to a group in the reference database.

```
> r <- readLines(output.file)
> z <- data.frame(id=I(character(length(r))),
         row.names=1:length(r))
> confidence <- .51
> for (i in 1:length(r)) {
         u <- unlist(strsplit(r[i], "\t"))
```

```
        confidenceLevel <- 0
        for (k in seq(5, length(u), 3)) {
                if (as.numeric(u[k]) < confidence) {
                        confidenceLevel <- k
                        break
                }
        }

        if (confidenceLevel == 0) { # full confidence
                r[i] <- paste(u[seq(3, length(u), 3)],
                        collapse="; ")
        } else { # unclassified at some level
                r[i] <- paste(u[seq(3, (confidenceLevel - 3), 3)],
                        collapse="; ")
                r[i] <- paste(r[i],
                        "unclassified",
                        sep="; ")
        }

        index <- grep(r[i],
                t$rank,
                fixed=TRUE)

        if (length(index) == 0)
                z$id[i] <- "unknown_lineage"
        else
                z$id[i] <- as.character(t$id[index[1]])
}
```

At this point we should have the group assignment for each sequence.

```
> head(z)
                   id
1      Lysinibacillus
2      Staphylococcus
3      Staphylococcus
4   Carnobacteriaceae
5      Peptococcaceae
6 Erysipelotrichaceae
```

5.2 Forming the Sequence Database

5.2.1 Importing Sequences

Next, we need to load the sequences into a sequence database (*dbFile*). This database will be used directly by the `FindChimeras` function. Here we can either set `dbConn` to a file location to write the database, or if there is enough memory simply create the database in memory.

```
> # specify a path for where to write the sequence database
> dbConn <- "<<path to write sequence database>>"
> # OR create the sequence database in memory
```

```
> dbConn <- dbConnect(SQLite(), ":memory:")
> Seqs2DB("<<path to FASTA file>>", "FASTA", dbConn, "unknown")


Reading FASTA file from line 0 to 99999

175 total sequences in database.
Time difference of 0.2 secs

[1] 175
```

5.2.2 Adding Group Names

At this point we can add the group names that we determined previously to the sequence database we just created.

```
> Add2DB(z, dbConn)
```

5.3 Finding Chimeras

Finally, we can use the `FindChimeras` function to search for chimeras in our sequences. Two sets of parameters are available for 16S sequences. The defaults apply to sequences longer than 1,000 nucleotides, while the second command applies to sequences that are shorter in length.

```
> # full-length sequences
> chimeras <- FindChimeras(dbConn, dbFileReference=dbRef, add2tbl=TRUE,
         maxGroupSize=1e4)
> # OR for short-length sequences
> chimeras <- FindChimeras(dbConn, dbFileReference=dbRef, minLength=40,
         minSuspectFragments=2, add2tbl=TRUE, maxGroupSize=1e4)


Group of Sequence (Number of Sequences):
  |==============================================================| 100%

Searching Other Groups:
  |==============================================================| 100%

Found 19 possible chimeras.
Added to table DNA:  "chimera".
Time difference of 1665.8 secs
```

If any chimeras were found then they are stored in the `chimeras data.frame`. There are several ways to determine which sequences were found to be chimeric.

```
> # outputs the number of chimeras that were found
> cat("Number of chimeras =", dim(chimeras)[1])
Number of chimeras = 19
> # get the ordering of chimeric sequences in the set
> row.names(chimeras)
 [1] "147" "89"  "87"  "90"  "88"  "132" "139" "97"  "5"   "120" "123" "13"
[13] "118" "114" "173" "174" "175" "18"  "124"
```

```
> # allows viewing the chimera results in a web browser
> BrowseDB(dbConn)
[1] TRUE
> # returns the FASTA description of the chimeric sequences
> dbGetQuery(dbConn, "select description from DNA where chimera is not null")
1       uncultured bacterium; Fin_CL-100633_OTU-22.
2       uncultured bacterium; Fin_CL-030731_OTU-12.
3        uncultured bacterium; Fin_CL-050647_OTU-9.
4       uncultured bacterium; Pro_CL-100643_OTU-10.
5        uncultured bacterium; LO1_CL-100643_OTU-3.
6        uncultured bacterium; LO3_CL-100641_OTU-3.
7       uncultured bacterium; UWH_CL-010711_OTU-13.
8   uncultured bacterium; Chlplus_CL-120529_OTU-35.
9       uncultured bacterium; UWH_CL-110523_OTU-35.
10      uncultured bacterium; UWH_CL-100636_OTU-11.
11 uncultured bacterium; Chlminus_CL-090519_OTU-14.
12      uncultured bacterium; UWL_CL-080514_OTU-34.
13      uncultured bacterium; UWL_CL-080545_OTU-12.
14       uncultured bacterium; Pro_CL-100633_OTU-7.
15      uncultured bacterium; UWL_CL-110645_OTU-15.
16        uncultured bacterium; UWL_CL-09061_OTU-5.
17       uncultured bacterium; Fin_CL-03079_OTU-21.
18      uncultured bacterium; UWL_CL-110518_OTU-11.
19      uncultured bacterium; UWL_CL-110548_OTU-32.


> dbDisconnect(dbRef)
[1] TRUE
> dbDisconnect(dbConn)
[1] TRUE
> unlink(output.file)
```

6 Session Information

All of the output in this vignette was produced under the following conditions:

- R version 3.2.2 (2015-08-14), `x86_64-pc-linux-gnu`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: BiocGenerics 0.14.0, Biostrings 2.36.4, DBI 0.3.1, DECIPHER 1.14.5, IRanges 2.2.7, RSQLite 1.0.0, S4Vectors 0.6.4, XVector 0.8.0
- Loaded via a namespace (and not attached): KernSmooth 2.23-15, tools 3.2.2, zlibbioc 1.14.0