Finding Chimeric Sequences

Erik S. Wright

October 29, 2019

Contents

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.codes`.

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 detection methods have been proposed with 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 DECIPHER is installed on your system, the code in each example can be obtained via:

```
> 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.codes/Download.html`.
If a reference database is not present then it is feasible to create 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 classifications 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 ribosomal RNA (rRNA) 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.

```
> dbRef <- dbConnect(SQLite(), "<<path to reference database>>")
> Seqs2DB("<<path to reference sequences>>", "FASTA", dbRef, "")
645151 total sequences in table Seqs.
Time difference of 172.81 secs
```

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

```
> ns <- dbGetQuery(dbRef, "select description from Seqs")
> ns$rank <- gsub("(.+) (.+)?", "\\2", ns$description)
> ns$rank <- gsub("(.+)?;(.*)$", "\\2\nRoot;\\1", ns$rank)
> ns$description <- gsub("(.+) (.+)?", "\\1", ns$description)
> Add2DB(ns, dbRef)
Expression:
update Seqs set description = :description where row_names = :row_names

Expression:
alter table Seqs add column rank TEXT

Expression:
update Seqs set rank = :rank where row_names = :row_names

Added to table Seqs:  "description" and "rank".

Time difference of 883.32 secs
```

It is necessary to perform two more steps before we can use the reference database for chimera finding:

```
> lengths <- IdLengths(dbRef, add2tbl=TRUE)
  |=============================================================| 100%
Lengths counted for 645151 sequences.
Added to Seqs:  "bases", "nonbases", and "width".

Time difference of 3924.83 secs

> groups <- FormGroups(dbRef, add2tbl=TRUE)
  |=============================================================| 100%
Updating column: "identifier"...
Updating column: "origin"...
Formed 1760 distinct groups.
Added to table Seqs: "identifier", "origin".

Time difference of 19860.67 secs
```

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, at which point the reference database will be ready for checking new sequences.

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

Note that sequences in the reference database must be in the same orientation as the query sequences. The function `OrientNucleotides` can be used to reorient the query sequences, if necessary.

5 Steps to Find Chimeras

5.1 Assigning Sequences to Reference Groups

5.1.1 Training the Classifier

In order to assign new sequences to a group in the reference database, it is necessary to first classify the sequences. We can use the functions `LearnTaxa` and `IdTaxa` to assign groups to the query sequences. See the "Classify Sequences" vignette for a more detailed description of this process.

```
> ids <- dbGetQuery(dbRef, "select identifier, origin from Seqs")
> ids <- paste(ids$origin, ids$identifier, sep=";")
> dna <- SearchDB(dbRef, type="DNAStringSet", remove="all")
> trainingSet <- LearnTaxa(dna, ids)
```

5.1.2 Creating a Query Sequence Database

Chimera finding requires two database tables, one for the reference sequences and one for the query sequences. Now that we have finished creating the chimera-free reference sequence database and training a classifer to its taxonomy, it is not time to make the query sequence database. This can be accomplished by specifying a new table within the reference sequence database or creating a new database file for the query sequences, as shown here.

```
> dbConn <- dbConnect(SQLite(), '<<path to query database>>')
> Seqs2DB('<<path to query sequences>>', 'FASTA', dbConn, '')
```

5.1.3 Assigning to Groups

We can now classify each of the query sequences and assign them to the corresponding group in the reference database. Note that the query sequences need to be in the same orientation as sequences in the reference sequence database.

```
> query <- SearchDB(dbConn, remove="all")
> groups <- IdTaxa(query, trainingSet, strand="top", threshold=0, processors=1)
> groups <- sapply(groups, function(x) tail(x$taxon, n=1))
> groups <- data.frame(identifier=groups, row.names=seq_along(groups), stringsAsFactors=FALSE)
> Add2DB(groups, dbConn)
```

5.2 Finding Chimeras

Finally, we can use the `FindChimeras` function to search for chimeras in our sequences.

```
> # full-length sequences
> chimeras <- FindChimeras(dbConn, dbFileReference=dbRef, add2tbl=TRUE)


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

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

```
Found 19 possible chimeras.
Added to table Seqs:  "chimera".
Time difference of 981.21 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 indices of chimeric sequences in the set
> as.numeric(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 Seqs 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
```

6 Session Information

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

- R version 3.6.1 (2019-07-05), `x86_64-apple-darwin15.6.0`
- Running under: `OS X El Capitan 10.11.6`

- Matrix products: default
- BLAS: `/Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib`
- LAPACK: `/Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: BiocGenerics 0.32.0, Biostrings 2.54.0, DECIPHER 2.14.0, IRanges 2.20.0, RSQLite 2.1.2, S4Vectors 0.24.0, XVector 0.26.0
- Loaded via a namespace (and not attached): DBI 1.0.0, KernSmooth 2.23-16, Rcpp 1.0.2, backports 1.1.5, bit 1.1-14, bit64 0.9-7, blob 1.2.0, compiler 3.6.1, crayon 1.3.4, digest 0.6.22, memoise 1.1.0, pillar 1.4.2, pkgconfig 2.0.3, rlang 0.4.1, tibble 2.1.3, tools 3.6.1, vctrs 0.2.0, zeallot 0.1.0, zlibbioc 1.32.0