Getting Started DECIPHERing

Erik S. Wright

October 30, 2017

Contents

1 About DECIPHER

DECIPHER is a software toolset that can be used for deciphering and managing biological sequences efficiently using the R statistical programming language. The program features tools falling into five categories:

- Sequence databases: import, maintain, view, and export a massive number of sequences.
- Sequence alignment: accurately align thousands of DNA, RNA, or amino acid sequences. Quickly find and align the syntenic regions of multiple genomes.
- Oligo design: test oligos in silico, or create new primer and probe sequences optimized for a variety of objectives.
- Manipulate sequences: trim low quality regions, correct frameshifts, reorient nucleotides, determine consensus, or digest with restriction enzymes.

- Analyze sequences: find chimeras, classify into a taxonomy, predict secondary structure, and create phylogenetic trees.

DECIPHER is available under the terms of the GNU Public License version 3.

2 Design Philosophy

2.1 Curators Protect the Originals

One of the core principles of DECIPHER is the idea of the non-destructive workflow. This revolves around the concept that the original sequence information should never be altered: sequences are exported looking identical to how they were when they were first imported. Essentially, the sequence information in the database is thought of as a backup of the original sequence file and no function is able to directly alter the sequence data. All of the workflows simply *add* information to the database, which can be used to analyze, organize, and maintain the sequences. When it comes time to export all or part of the sequences they are preserved in their original state without alteration.

2.2 Don't Reinvent the Wheel

DECIPHER makes use of the Biostrings package that is a core part of the Bioconductor suite. This package contains numerous functions for common operations such as searching, manipulating, and reverse complementing sequences. Furthermore, DECIPHER makes use of the Biostrings interface for handling sequence data so that sequences are stored in `XStringSet` objects. These objects are compatible with many useful packages in the Bioconductor suite.

A wide variety of user objectives necessitates that DECIPHER be extensible to customized projects. R provides a simple way to place the power of thousands of packages at your fingertips. Likewise, R enables direct access to the speed and efficiency of the programming language C while maintaining the utility of a scripting language. Therefore, minimal code is required to solve complex new problems. Best of all, the R statistical programming language is open source, and maintains a thriving user community so that direct collaboration with other R users is available on several Internet forums.

2.3 That Which is the Most Difficult, Make Fastest

A core objective of DECIPHER is to make massive tasks feasible in minimal time. To this end, many of the most time consuming functions are parallelized to make use of multiple processors. For example, the function `DistanceMatrix` gets almost a 1x speed boost for each processor core. A modern processor with 8 cores can see a factor of close to eight times speed improvement. Similar speedups can be achieved in many other DECIPHER functions by setting the *processors* argument. This is all made possible through the use of OpenMP in C-level code.

Other time consuming tasks are handled efficiently. The function `FindChimeras` can uncover sequence chimeras by searching through a reference database of over a million sequences for thousands of 30-mer fragments in a number of minutes. This incredible feat is accomplished by using the *PDict* class provided by Biostrings. Similarly, the `SearchDB` function can obtain the one-in-a-million sequences that match a targeted query in a matter of seconds. Such high-speed functions enable the user to find solutions to problems that previously would have been extremely difficult or nearly impossible to solve using antiquated methods.

2.4 Stay Organized

It is no longer necessary to store related data in several different files. DECIPHER is enabled by RSQLite, which is an R interface to SQLite databases. DECIPHER creates an organized collection of sequences and their associated information known as a sequence database. *SQLite* databases are flat files, meaning they

can be handled just like any other file. There is no setup required since *SQLite* does not require a server, unlike many other database engines. These attributes of *SQLite* databases make storing, backing-up, and sharing sequence databases relatively straightforward.

Separate projects can be stored in distinct tables in the same sequence database. Each new table is structured to include every sequence's description, identifier, and a unique key called the *row_name* all in one place. The sequences are referenced by their *row_names* or *identifier* throughout most functions in the package. Using *row_names*, new information created with DECIPHER functions can be added as additional database columns to their respective sequences' rows in the database table. To prevent the database from seeming like a black box there is a function named `BrowseDB` that facilitates viewing of the database contents in a web browser. A similar function is available to view sequences called `BrowseSeqs`.

The amount of DNA sequence information available is currently increasing at a phenomenal rate. DECIPHER stores individual sequences using a custom compression format, called *nbit*, so that the database file takes up much less drive space than a standard text file of sequences. The compressed sequences are stored in a hidden table that is linked to the main information table that the user interacts with regularly. For example, by default sequence information is stored in the table "Seqs", and the associated sequences are stored in the table "_Seqs". Storing the sequences in a separate table greatly improves access speed when there is a large amount of sequence information. Separating projects into distinct tables further increases query speed over that of storing every project in a single table.

3 Functionality

The functions of DECIPHER can be grouped into several categories based on intended use:

1. Primary functions for interacting with a sequence database:
   (a) `Add2DB`
   (b) `DB2Seqs`
   (c) `SearchDB`
   (d) `Seqs2DB`
2. Secondary functions for typical database tasks:
   (a) `IdentifyByRank`
   (b) `IdLengths`
3. Functions related to forming consensus:
   (a) `ConsensusSequence`
   (b) `Disambiguate`
   (c) `IdConsensus`
4. Phylogenetics and sequence comparison:
   (a) `DistanceMatrix`
   (b) `IdClusters`
   (c) `MaskAlignment`
   (d) `ReadDendrogram`
   (e) `StaggerAlignment`
   (f) `WriteDendrogram`

5. Visualization with a web browser:
   (a) `BrowseDB`
   (b) `BrowseSeqs`
6. Manipulating sequences:
   (a) `CorrectFrameshifts`
   (b) `OrientNucleotides`
   (c) `RemoveGaps`
   (d) `TrimDNA`
7. Analyzing sequences:
   (a) `DigestDNA`
   (b) `PredictDBN`
   (c) `PredictHEC`
8. Multiple sequence alignment:
   (a) `AdjustAlignment`
   (b) `AlignDB`
   (c) `AlignProfiles`
   (d) `AlignSeqs`
   (e) `AlignTranslation`
9. Comparison of sequences that are not collinear (i.e., genomes):
   (a) `AlignSynteny`
   (b) `FindSynteny`
   (c) *Synteny-class*
10. Functions related to chimeras (PCR artifacts):
    (a) `CreateChimeras`
    (b) `FindChimeras`
    (c) `FormGroups`
11. Functions related to DNA microarrays:
    (a) `Array2Matrix`
    (b) `CalculateEfficiencyArray`
    (c) `DesignArray`
    (d) `NNLS`
12. Functions related to probes for fluorescence *in situ* hybridization (FISH):
    (a) `CalculateEfficiencyFISH`
    (b) `DesignProbes`
    (c) `TileSeqs`

13. Functions related to primers for polymerase chain reaction (PCR):
    (a) `AmplifyDNA`
    (b) `CalculateEfficiencyPCR`
    (c) `DesignPrimers`
    (d) `DesignSignatures`
    (e) `MeltDNA`
14. Classifying sequences into a taxonomy:
    (a) `LearnTaxa`
    (b) `IdTaxa`
    (c) *Taxa-class*

4 Installation

4.1 Typical Installation (recommended)

1. Install the latest version of R from `http://www.r-project.org/`.
2. Install DECIPHER in R by entering:

```
> source("https://bioconductor.org/biocLite.R")
> biocLite("DECIPHER")
```

4.2 Update to the Latest Version

1. Update R to the latest version available from `http://www.r-project.org/`.
2. Reinstall DECIPHER in R by entering:

```
> source("https://bioconductor.org/biocLite.R")
> biocLite("DECIPHER")
```

4.3 Manual Installation

4.3.1 All platforms

1. Install the latest R version from `http://www.r-project.org/`.
2. Install Biostrings in R by entering:

```
> source("https://bioconductor.org/biocLite.R")
> biocLite("Biostrings")
```

3. Install RSQLite in R by entering:

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

4. Download DECIPHER from `http://DECIPHER.codes`.

4.3.2 Mac OS X

1. First install Command Line Tools in the App Store. Then, in R run:

```
> install.packages("<<path to Mac OS X DECIPHER.tgz>>", repos=NULL)
```

4.3.3 Linux

In a shell enter:

```
R CMD build --no-build-vignettes "<<path to DECIPHER source>>"
R CMD INSTALL "<<path to newly built DECIPHER.tar.gz>>"
```

4.3.4 Windows

Two options are available: the first is simplest, but requires the pre-built binary (DECIPHER.zip).

1. First Option:

```
> install.packages("<<path to Windows DECIPHER.zip>>", repos=NULL)
```

2. Second Option (more difficult):
   (a) Install Rtools from http://cran.r-project.org/bin/windows/Rtools/. Be sure to check the box that says edit PATH during installation.
   (b) Open a MS-DOS command prompt by clicking Start -> All Programs -> Accessories -> Command Prompt.
   (c) In the command prompt enter:

```
R CMD build --no-build-vignettes "<<path to DECIPHER source>>"
R CMD INSTALL "<<path to newly built DECIPHER.zip>>"
```

5 Example Workflow

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

```
> library(DECIPHER)
```

Help for any function can be accessed through a command such as:

```
> ? DECIPHER
```

To begin, we can import a FASTA, FASTQ, or GenBank file into a sequence database. Here we will import the sequences into an in-memory database that will be removed when we disconnect from the database. In-memory databases are useful for temporary examples, but typically we would specify the path to a file where we want to store the database. This is especially the case when there are many sequences, as they might not all fit into memory.
In this example, we will use a GenBank file that is included with DECIPHER installation. We must set the sequences' *identifier* when importing with `Seqs2DB`. Here we will identify the sequences with the word "Bacteria". The *identifier* is used by many DECIPHER functions to reference a specific subset of sequences in the database, and can be reset at a later time using a number of different methods:

```
> # access a sequence file included in the package:
> gen <- system.file("extdata", "Bacteria_175seqs.gen", package="DECIPHER")
> # connect to a database:
> dbConn <- dbConnect(SQLite(), ":memory:")
> # import the sequences into the sequence database
> Seqs2DB(gen, "GenBank", dbConn, "Bacteria")
Reading GenBank file chunk 1

175 total sequences in table Seqs.
Time difference of 0.17 secs
```

Now we can view the table of information we just added to the database in a web browser (Fig. 1):

```
> BrowseDB(dbConn)
```

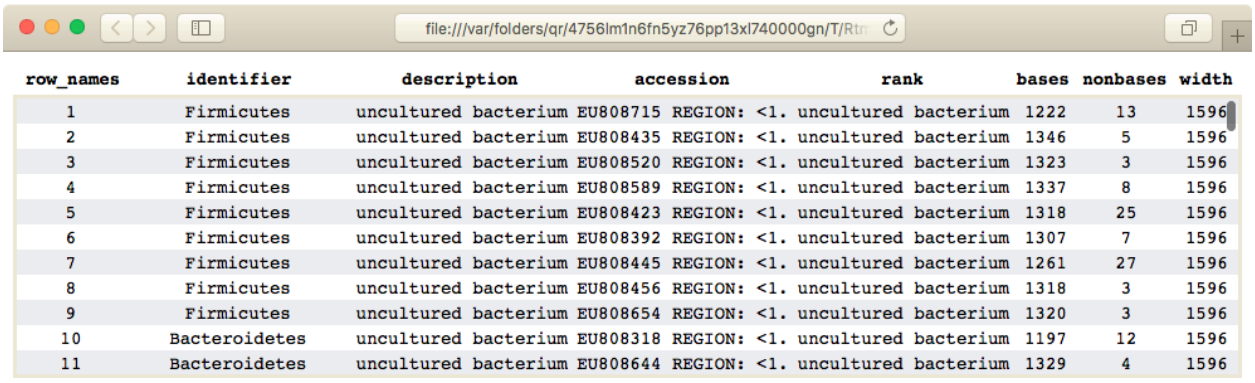
Suppose we wanted to count the number of bases in each sequence and add that information to the database:

```
> l <- IdLengths(dbConn)
Lengths counted for 175 sequences.

Time difference of 0.01 secs
> head(l)
  bases nonbases width
1  1222       13  1596
2  1346        5  1596
3  1323        3  1596
4  1337        8  1596
5  1318       25  1596
6  1307        7  1596
> Add2DB(l, dbConn, verbose=FALSE)
> BrowseDB(dbConn, maxChars=20)
```

Next let's identify our sequences by phylum and update this information in the database:

```
> r <- IdentifyByRank(dbConn, level=3, add2tbl=TRUE)
Updating column: "identifier"...
Formed 7 distinct groups.
Added to table Seqs: "identifier".
Time difference of 0.03 secs
> BrowseDB(dbConn, maxChars=20)
```



row_names	identifier	description	accession	rank	bases	nonbases	width
1	Firmicutes	uncultured bacterium	EU808715 REGION: <1.	uncultured bacterium	1222	13	1596
2	Firmicutes	uncultured bacterium	EU808435 REGION: <1.	uncultured bacterium	1346	5	1596
3	Firmicutes	uncultured bacterium	EU808520 REGION: <1.	uncultured bacterium	1323	3	1596
4	Firmicutes	uncultured bacterium	EU808589 REGION: <1.	uncultured bacterium	1337	8	1596
5	Firmicutes	uncultured bacterium	EU808423 REGION: <1.	uncultured bacterium	1318	25	1596
6	Firmicutes	uncultured bacterium	EU808392 REGION: <1.	uncultured bacterium	1307	7	1596
7	Firmicutes	uncultured bacterium	EU808445 REGION: <1.	uncultured bacterium	1261	27	1596
8	Firmicutes	uncultured bacterium	EU808456 REGION: <1.	uncultured bacterium	1318	3	1596
9	Firmicutes	uncultured bacterium	EU808654 REGION: <1.	uncultured bacterium	1320	3	1596
10	Bacteroidetes	uncultured bacterium	EU808318 REGION: <1.	uncultured bacterium	1197	12	1596
11	Bacteroidetes	uncultured bacterium	EU808644 REGION: <1.	uncultured bacterium	1329	4	1596

Figure 1: Database table shown in web browser

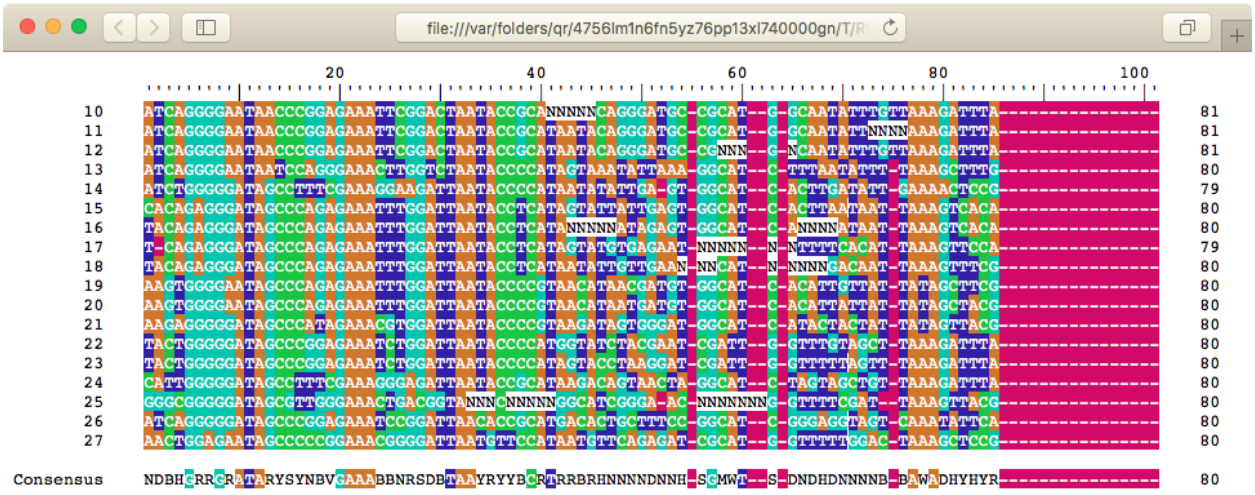


Figure 2: Sequences shown in web browser

We can now look at only those sequences that belong to the phylum *Bacteroidetes* (Fig. 2):

```
> dna <- SearchDB(dbConn, identifier="Bacteroidetes")
Search Expression:
select row_names, sequence from _Seqs where row_names in (select row_names
from Seqs where identifier is "Bacteroidetes")

DNAStringSet of length: 18
Time difference of 0 secs
> BrowseSeqs(subseq(dna, 140, 240))
```

Let's construct a phylogenetic tree from the *Bacteroidetes* sequences (Fig. 3):

```
> d <- DistanceMatrix(dna, correction="Jukes-Cantor", verbose=FALSE)
> c <- IdClusters(d, method="ML", cutoff=.05, showPlot=TRUE, myXStringSet=dna, verbose=FALSE)
```

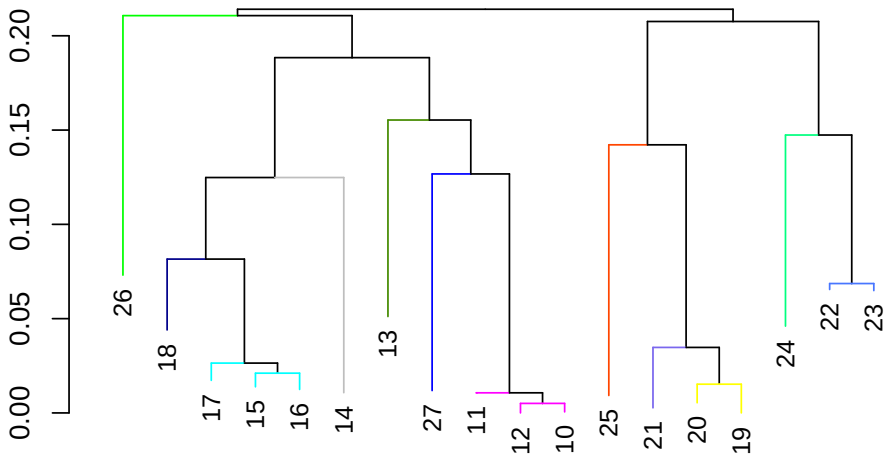


Figure 3: Maximum likelihood tree showing the relationships between sequences.

Here we created the database in-memory by connecting to ":memory:" above. This works fine for small databases, but it is necessary to initialize larger databases with a quoted file path rather than ":memory:". Optionally, we could use the command below to save an in-memory database to a file for long term storage. 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.

```
> sqliteCopyDatabase(dbConn, "<<path to database>>")
```

Finally, we should disconnect from the database connection. Since the sequence database was created in temporary memory, all of the information will be erased:

```
> dbDisconnect(dbConn)
```

6 Session Information

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

- R version 3.4.2 (2017-09-28), `x86_64-pc-linux-gnu`
- Running under: `Ubuntu 16.04.3 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.6-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.6-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: BiocGenerics 0.24.0, Biostrings 2.46.0, DECIPHER 2.6.0, IRanges 2.12.0, RSQLite 2.0, S4Vectors 0.16.0, XVector 0.18.0
- Loaded via a namespace (and not attached): DBI 0.7, Rcpp 0.12.13, bit 1.1-12, bit64 0.9-7, blob 1.1.0, compiler 3.4.2, digest 0.6.12, memoise 1.1.0, pkgconfig 2.0.1, rlang 0.1.2, tibble 1.3.4, tools 3.4.2, zlibbioc 1.24.0