Using DECIPHER v2.0 to Analyze Big Biological Sequence Data in R

by Erik S. Wright

Abstract In recent years, the cost of DNA sequencing has decreased at a rate that has outpaced improvements in memory capacity. It is now common to collect or have access to many gigabytes of biological sequences. This has created an urgent need for approaches that analyze sequences in subsets without requiring all of the sequences to be loaded into memory at one time. It has also opened opportunities to improve the organization and accessibility of information acquired in sequencing projects. The **DECIPHER** package offers solutions to these problems by assisting in the curation of large sets of biological sequences stored in compressed format inside a database. This approach has many practical advantages over standard bioinformatics workflows, and enables large analyses that would otherwise be prohibitively time consuming.

Introduction

With the advent of next-generation sequencing technologies, the cost of sequencing DNA has plummeted, facilitating a deluge of biological sequences (Hayden, 2014). Since the cost of computer storage space has not kept pace with biologists' ability to generate data, multiple compression methods have been developed for compactly storing nucleotide sequences (Deorowicz and Grabowski, 2013). These methods are critical for efficiently transmitting and preserving the outputs of next-generation sequencing machines. However, far less emphasis has been placed on the organization and usability of the massive amounts of biological sequences that are now routine in bioinformatics work. It is still commonplace to decompress large files and load them entirely into memory before analyses are performed. This traditional approach is quickly becoming infeasible as sequence sets swell in size, and alternative methods for storing, organizing, and analyzing sequences are needed.

A typical bioinformatics workflow begins with a set of biological sequences in one or more text files. These sequences are used as input to subsequent analysis steps, each of which generates text files as output (Schloss et al., 2011). Large workflows constructed in this manner can generate a plethora of text files, often resulting in unnecessary redundancy and disorganization. Fortunately, databases offer an organized means for storing related data, and underlie many commonly used bioinformatics software such as BLAST (Altschul et al., 1997). Nevertheless, biologists rarely use databases to curate their own sequences, in large part due to the difficulties associated with creating and accessing a database. Here I describe flexible user-friendly workflows for employing databases to efficiently analyze large sets of sequences via the R programming language.

Although R has traditionally been viewed as a statistical software, many add-on packages are available for analyzing biological sequence data. One representative, the **Biostrings** package (Pagès et al.), offers a suite of functions for reading, writing, searching, and manipulating DNA, RNA, or amino acid sequences. Sequences are stored in memory according to their corresponding XStringSet class, where "X" is specific to the type of sequences (e.g., "AA" for amino acid sequences). For example, a "DNAStringSet" can store the standard DNA bases ("A", "C", "G", or "T"), as well as ambiguity codes (e.g., "N" for any bases) and gap characters used in alignment ("-").

The DECIPHER package also makes use of XStringSet classes. However, unlike other R packages for biological sequence analysis, **DECIPHER** employs databases so that an entire sequence set does not need to simultaneously reside in memory. This enables **DECIPHER** to extend many analyses to millions of sequences without requiring extreme amounts of memory, and offers a means of handling the even more massive biological datasets of the future. The **DECIPHER** package also includes many advanced functions for oligonucleotide design (Wright et al., 2014a,b; Wright and Vetsigian, 2016), sequence alignment (Wright, 2015), and other common bioinformatics tasks. Despite its many applications, the core database functionality underpinning **DECIPHER** has not been previously described.

New users of **DECIPHER** are often unaccustomed to the use of a database to manage their own sequences. The purpose of this text is to describe the merits of this approach, and outline how sequence databases are configured by **DECIPHER** and can be used to improve analysis workflows. Sequences are stored independently within the database in a compressed format, which enables the database to be compact while maintaining fast random access to different sequences. The custom compression algorithm implemented in **DECIPHER** is compared to standard compression algorithms accessible within R. Finally, example uses of a sequence database are provided to demonstrate the power of this alternative workflow.

Merits of databases for storing biological sequences

A database, much like a spreadsheet, is an ideal way to maintain interconnected data. The concept of storing biological sequences in a relational database is not new (Xie et al., 2000), and underpins many popular bioinformatics programs and online web tools. However, end-users of these tools rarely directly employ databases for their own sequences, despite many practical advantages such as:

1. Organizational improvements:
   Databases can be arranged to minimize redundancy by maintaining an association between different columns. For example, the length of each sequence in the database can be easily added as a separate column.

2. Random access:
   Databases permit quick access to subsets of the data, without the need to seek through large text files in order to find the desired subset. For example, it is very fast to obtain the longest sequence in the database once a column with the length of each sequence has been added.

3. Concurrent users:
   A database can be queried concurrently by multiple users without interference. For example, one user can obtain the shortest sequence, while another simultaneously requests the longest sequence.

4. Reliable storage:
   Commands can be used that allow the database to revert changes in case of a mistake. **DECIPHER** workflows are designed to be non-destructive, so that the original sequence information is always preserved.

Several standard add-on packages are available to interface between R and popular database management systems (DBMSs) including MySQL, PostgreSQL, and SQLite (Ripley, 2001a). **DECIPHER** uses SQLite for a number of reasons. First, SQLite databases are flat-files that can easily be transferred between computers, or even emailed like a standard text file. Second, SQLite requires minimal setup on the part of the end-user, unlike some DBMSs. Third, support for the BLOB data type from R is currently only available from the **RSQLite** package. The BLOB type is used to store compressed sequences, which are raw (binary) type within R. Lastly, SQLite databases can contain an exceedingly large number of rows, up to 2^{64}, meaning that they are typically limited in size by the available disk space rather than the DBMS.

The use of SQLite as the sole DBMS results in a few limitations. First, users cannot be given separate access privileges, as all users are considered database administrators. Second, concurrent writes to the same database are not permitted, although concurrent reads are not a problem. These drawbacks cause few practical limitations for a typical group consisting of a few database users performing standard operations with **DECIPHER**. Furthermore, databases can be organized such that each sequencing project resides in its own table, which minimizes conflicts between users.

Anatomy of a DECIPHER database

DECIPHER uses a simple relational database schema involving two tables (Fig. 1): one that is highly "visible" to the user (named "Seqs" by default) containing information about the sequences, and a second "hidden" table (named "_Seqs") for storing compressed sequences and, if applicable, their corresponding quality scores. The tables are connected by a shared primary key, named "row_names", that enables fast lookup between the two tables. This split table design substantially increases access speed over using a single table, as the table being queried does not include any sequences, which are often large in size and can slow access to other data. Use of separate tables within the same database is provided by the argument tblName in each function.

Sequences are imported with the Seqs2DB function which supports three popular file formats as well as in-memory XStringSet objects. The destination table is automatically populated with appropriate text columns containing information stored in the file. For example, sequences imported from FASTA files store each sequence's record name in a column named "description". FASTQ files are imported similarly, but also store the quality information corresponding to each sequence in gzip compressed format. For GenBank files, the "description" column is obtained from the DEFINITION field and other fields can be imported as desired using the fields argument of Seqs2DB. By default, the ACCESSION and ORGANISM fields are imported as additional columns named "accession" and "rank" in the database.

All **DECIPHER** databases require an "identifier" to be specified during import. The identifier column is used extensively by **DECIPHER** functions, and is the recommended way to delineate groups of sequences. Common ways to identify sequences include: the file they were imported from
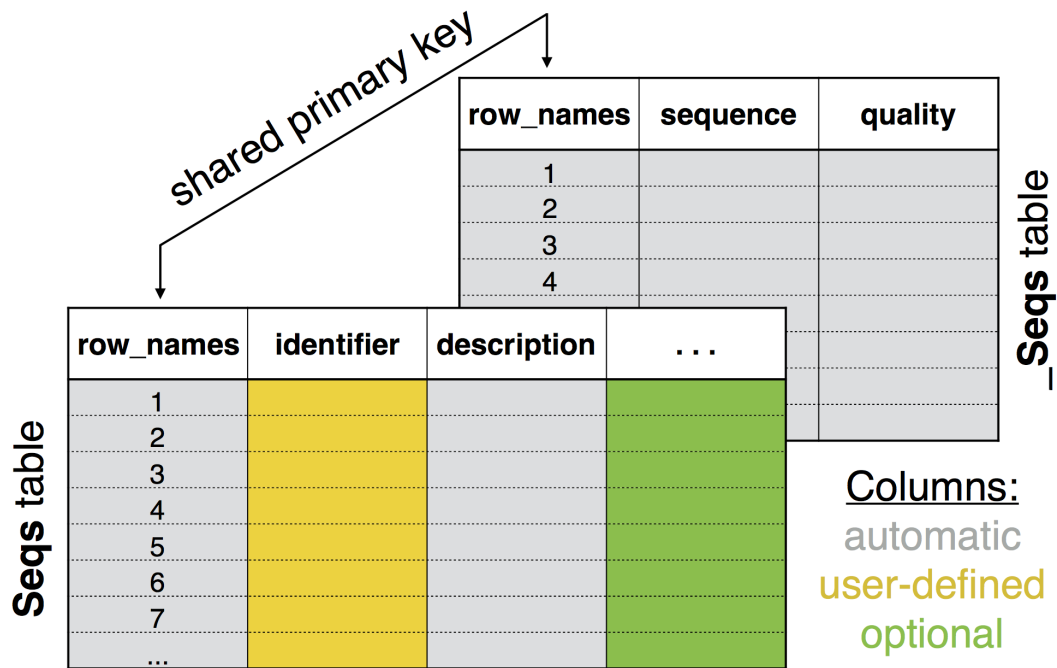
Figure 1: Database schema used by **DECIPHER**. Two tables are created when importing sequences with `Seqs2DB`: one that is highly visible to the user (named "Seqs" by default) and a second (named "_Seqs") that is largely hidden. The tables are connected by a shared primary key ("row_names") that enables fast lookup between the two tables. Columns containing the sequence descriptors, compressed sequences, and compressed quality scores (if applicable) are automatically generated. The user must specify an "identifier" during import, which can be changed later through a variety of methods. Additional columns of data may be added to the database using the `Add2DB` function.

(e.g., "file1", "file2", etc.), the cluster they belong to in a phylogeny (e.g., "cluster1", "cluster2", etc.), or the name of the organism from which the sequence originated (e.g., "E. coli", "Yeast", etc.). It is also possible to simply provide an empty character string (i.e., "") as the identifier during import, and then set the identifier later. **DECIPHER** includes several functions to help with identifying the sequences after they are imported. For example, the `IdClusters` function can assign phylogenetic cluster numbers. The `IdentifyByRank` function can parse the "rank" column from an imported GenBank file to identify sequences according to a given taxonomic rank (e.g., "species").

DECIPHER uses R's connection interface (Ripley, 2001b) to read files incrementally during import. Files compressed with gzip, bzip2, xz, or lzma compression are automatically detected and read appropriately without user intervention. The user can also provide a URL instead of a file path, which enables sequences to be imported from "http" or "ftp" sources without first needing to save the file locally. In the case of URLs, only uncompressed text files or files with gzip compression are supported. Notwithstanding this limitation, reading files directly from online sources, such as NCBI repositories, is often preferable to downloading files locally before importing them into a database as it prevents unnecessary redundancy. An example of importing a compressed GenBank file from online is shown below:

```
> library(DECIPHER)
>
> # specify the input file and database location
> gbk_file <- "ftp://ftp.ncbi.nlm.nih.gov/genomes/H_sapiens/RNA/rna.gbk.gz"
> db_file <- "~/Desktop/SeqsDB.sqlite"
> dbConn <- dbConnect(SQLite(), db_file)
>
> # import the sequences from online
> Seqs2DB(seqs = gbk_file,
+         type = "GenBank",
+         dbFile = dbConn,
+         identifier = "Human_mRNA")
162916 total sequences in table Seqs.
Time difference of 176.19 secs
```

row_names	identifier	description	accession	rank
1	curated RNA	Homo sapiens DEAD/H (Asp–Glu–Ala–Asp/His) box heli	NR_046018 XM_003403543	Homo sapiens Eukaryota; Metazoa; Chordata; Craniat
2	curated RNA	Homo sapiens WAS protein family homolog 7 pseudoge	NR_024540 XR_017611 XR_039986	Homo sapiens Eukaryota; Metazoa; Chordata; Craniat
3	curated RNA	Homo sapiens microRNA 6859-1 (MIR6859-1), microRNA	NR_106918	Homo sapiens Eukaryota; Metazoa; Chordata; Craniat
4	curated RNA	Homo sapiens microRNA 1302-2 (MIR1302-2), microRNA	NR_036051	Homo sapiens Eukaryota; Metazoa; Chordata; Craniat
5	curated RNA	Homo sapiens family with sequence similarity 138,	NR_026818	Homo sapiens Eukaryota; Metazoa; Chordata; Craniat
6	predicted RNA	PREDICTED: Homo sapiens uncharacterized LOC1053792	XR_948874	Homo sapiens Eukaryota; Metazoa; Chordata; Craniat
7	curated mRNA	Homo sapiens olfactory receptor, family 4, subfami	NM_001005484 XM_001123390	Homo sapiens Eukaryota; Metazoa; Chordata; Craniat
8	predicted RNA	PREDICTED: Homo sapiens uncharacterized LOC1053792	XR_948893	Homo sapiens Eukaryota; Metazoa; Chordata; Craniat
9	predicted RNA	PREDICTED: Homo sapiens uncharacterized LOC1053792	XR_948891	Homo sapiens Eukaryota; Metazoa; Chordata; Craniat
10	predicted RNA	PREDICTED: Homo sapiens uncharacterized LOC1053792	XR_948892	Homo sapiens Eukaryota; Metazoa; Chordata; Craniat

Figure 2: Screenshot of viewing a database table in a web browser using the `BrowseDB` function. Metadata information such as accession numbers and taxonomy are automatically imported from GenBank sequence files into separate table columns. The value in the "identifier" column controls sequence groupings in many **DECIPHER** functions, and can be set during or after import by a variety of methods. For easier viewing, the text in each field is truncated at a maximum of 50 characters by default.

The imported file contains known and predicted human RNA sequences. At this point it is useful to reset the values in the identifier column so that they can be referenced in downstream analyses. In this particular example, the prefix of the accession number can be parsed into one of four values, indicating whether the sequence is a predicted or hand-curated RNA or messenger RNA (mRNA) sequence. This information is updated in the database using the `Add2DB` function, which, like many other **DECIPHER** functions, displays the associated SQL commands when the input argument verbose is TRUE. Add2DB will add or update table columns in accordance with the column names and row names in a `"data.frame"` input. For example, in this case the values of identifier in the input `"data.frame"` are added to the rows with corresponding "row_names" in the database. To view the results of this modification, the database table can be displayed in a web browser with the `BrowseDB` function (Fig. 2).

```
> # reset the 'identifier' column based on the prefix of the accession number
> x <- dbGetQuery(dbConn, "select accession from Seqs")
> id <- substring(x$accession, 1, 2)
> id <- c(`NM` = "curated mRNA",
+         `NR` = "curated RNA",
+         `XM` = "predicted mRNA",
+         `XR` = "predicted RNA")[id]
> Add2DB(data.frame(identifier = id), dbConn)
Expression:
update or replace Seqs set identifier = :identifier where row_names =
:row_names

Added to table Seqs:  "identifier".
Time difference of 2.06 secs

> BrowseDB(dbConn, limit = 1000)
```

The nbit compression format for nucleotides

Many compression algorithms have been proposed for storing nucleotide sequences, some of which are specific to the FASTQ file format (Deorowicz and Grabowski, 2013). The most common compression method is gzip, owing to its reasonable compression ratio and high decompression rate. However, since gzip is a generalized method, it may be possible to obtain better compression ratios by using algorithms specific to DNA sequences. In particular, reference-based compression is generally preferable when a reference sequence is available (Jones et al., 2012). In re-sequencing projects a reference genome
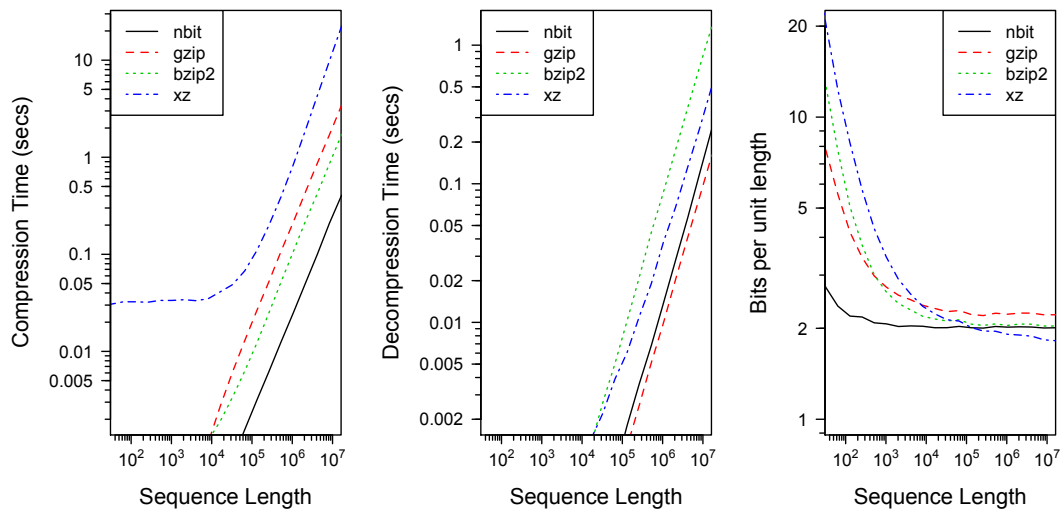
Figure 3: Comparison between different lossless compression algorithms on random subsequences of Human Chromosome II (HG18). **DECIPHER**'s custom **nbit** compression exhibited substantially faster compression rates than the other methods. The **nbit** algorithm offered a better compression ratio than any of the other methods for sequences less than about 100,000 nucleotides, beyond which xz compression provided more compaction, albeit at a substantially slower compression rate. The average of 100 replicates is shown for all methods, gzip (v1.2.8), bzip2 (v1.0.6), and xz (v5.0.7), computed using a single processor.

is always available, whereas it is often unavailable for new sequencing projects. In either case, compression of a file containing many similar sequences allows redundancy to be exploited to a greater extent than independently compressing sequences.

Several considerations were taken into account when designing a compression method for **DECIPHER**. First, the method must work well with a wide variety in the number of sequences and their lengths. Second, in order to allow random access and deposition, each sequence is stored independently in the database, ruling out exploiting redundancy between sequences. Third, compression and decompression rates were prioritized over achieving the maximal possible compression ratio. Fourth, support for all possible characters in the DNA and RNA alphabets was desired, in particular the gap character that is largely neglected by most DNA-specific compression formats. These considerations led to the development of the **nbit** compression method for DNA or RNA sequences. Compression with **nbit** uses a combination of 2-bit encoding for gapless bases (i.e., A=00, C=01, G=10, T=11), and 3-bit encoding for gappy regions. Although this encoding was inspired by prior work (Wandelt et al., 2014), **DECIPHER** uses a unique implementation that is customized to the package's goals.

In principle, the maximum achievable compression rate is 2-bits per base with this encoding, which is the theoretical limit (i.e., Shannon entropy) for four randomly drawn characters in equal proportions. However, DNA sequences contain appreciable information and are non-random in their construction, which permits additional compression to be achieved. To this end, the **nbit** algorithm incorporates runs of a single base or ambiguity codes (e.g., "NNN"), which are frequent in some sequences. Furthermore, long DNA sequences often contain exact repeats or reverse complement repeats, which can be stored compactly by referencing their prior occurrence. Finally, the **nbit** format includes a variable-sized cyclic redundancy check to ensure data integrity.

The **nbit** compression and decompression algorithms were implemented in the **DECIPHER** function Codec, which interconverts between character (uncompressed) and binary (compressed) data. In comparison to the three generalized compression methods available in R, the **nbit** algorithm exhibits a substantially faster compression rate, as shown in Figure 3. Compression sizes are generally better than those of the other methods, except for sequences longer than about 100,000 nucleotides where xz compression results in more compaction at the expense of substantially longer compression times. The user may also specify to use more than one processor with **nbit**, in which case the Codec function will compress/decompress sequences in parallel. Furthermore, the Codec function will automatically fall back to gzip compression when the sequences are incompressible with **nbit**, as in the case of amino acid sequences.
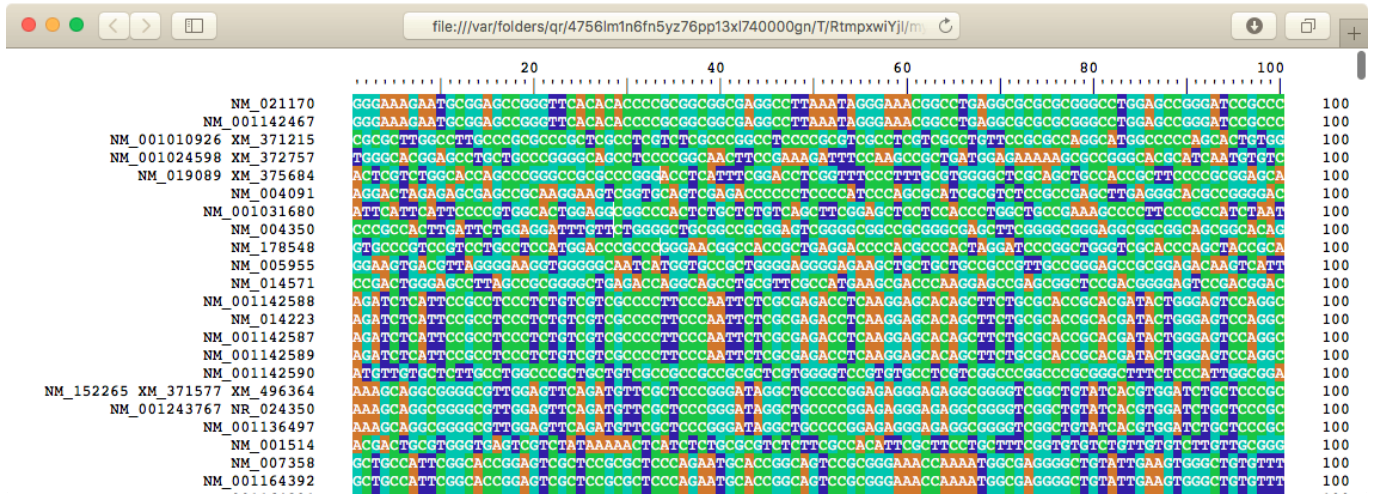
Figure 4: Sequences returned from a database query are displayed in a web browser using the `BrowseSeqs` function. The mRNA sequences shown here all match the query "transcription factor" and are named by their accession number(s). Sequence positions are colored according to their DNA base (A, C, G, or T) and wrapped at 100 nucleotides.

Example workflow with DECIPHER

One of the major advantages of using a sequence database is the ability to quickly access subsets of the sequences matching certain criteria. The `SearchDB` function can be used to easily build flexible queries and obtain the sequences meeting those specifications. `SearchDB` supports several common SQL clauses, including 'LIMIT', 'OFFSET', 'ORDER BY', and 'WHERE'. If unspecified, `SearchDB` can automatically detect the type of sequences (DNA, RNA, or AA) returned from the search. In addition, it can be used to quickly count the number of sequences matching a query, name the sequences based on the value in a specific table column, remove gaps ("-") from sequences, or replace characters not present in the specified sequence alphabet. As an example, the command below will find all of the "curated mRNA" sequences with "transcription factor" in their "description" and name the sequences by their accession number. The sequences can then be viewed in a web browser using the `BrowseSeqs` function, as shown in Figure 4.

```
> dna <- SearchDB(dbConn,
+               identifier = "curated mRNA",
+               nameBy = "accession",
+               clause = "description like '%transcription factor%'")
Search Expression:
select accession, _Seqs.sequence from Seqs join _Seqs on Seqs.row_names =
_Seqs.row_names where _Seqs.row_names in (select row_names from Seqs where
identifier is "curated mRNA" and description like '%transcription factor%')

DNAStringSet of length: 437
Time difference of 0.44 secs

> BrowseSeqs(dna, colWidth = 100)
```

Several **DECIPHER** functions make use of `SearchDB`'s `limit` argument to extract batches of sequences. The `limit` argument can be either a single numeric value indicating the maximum number of sequences to return, or a character string specifying two numbers separated by a comma giving the offset and limit. For example, the `IdLengths` function makes use of this feature to efficiently compute the lengths of all of the sequences in a database table. This length information is then added to the database in batches using the `Add2DB` function. A similar workflow is used by many **DECIPHER** functions so that all of the sequences do not have to be kept in memory simultaneously. An example of using the `IdLengths` function is shown below:

```
> l <- IdLengths(dbConn, add2tbl = TRUE)
  |================================================================| 100%
Lengths counted for 162916 sequences.
Added to Seqs:  "bases", "nonbases", and "width".
```

```
Time difference of 26.15 secs
```

Rather than using offset and limit values, several **DECIPHER** functions make use of the user-specified identifier to split the sequences into groups. For example, the IdConsensus function will create a single consensus sequence for the sequences corresponding to each identifier in a table, by accessing one identifier's sequences at a time. It is also possible to construct more sophisticated queries of the database using the clause argument of the SearchDB function. The clause will be appended to the query after the keyword 'WHERE'. Below is an example of using the clause argument to retrieve the longest 'curated RNA' sequence in the table, which is 91,671 nucleotides in length.

```
> SearchDB(dbConn,
+          identifier = "curated RNA",
+          clause = "bases = (select max(bases) from Seqs
+                    where identifier is 'curated RNA')")
Search Expression:
select row_names, sequence from _Seqs where row_names in (select row_names
from Seqs where identifier is "curated RNA" and bases = (select max(bases)
from Seqs where identifier is 'curated RNA'))

DNAStringSet of length: 1
Time difference of 0.25 secs

  A DNAStringSet instance of length 1
    width seq                                                     names
[1] 91671 AGGCAGAACGGTCGCCGCGTCGC...ATGAAAACTATGAAAACTGACTA 73201
```

As the above example demonstrates, a knowledge of basic SQL syntax is helpful to harness the full potential of **DECIPHER** for analyzing big biological sequence data in R. Nevertheless, it is still possible to automatically generate a wide variety of complex queries using only the input arguments of **DECIPHER**'s database functions. For example, the DB2Seqs function can be used to export a large number of sequences meeting certain criteria from the database. The sequences are exported in FASTA format, or FASTQ format if corresponding quality scores are available. The code below exports a gzip compressed FASTA file containing all of the "predicted mRNA" sequences, ordered by their length, and with each sequence record named by its accession number.

```
> out_file <- "~/Desktop/seqs.fas.gz"
> DB2Seqs(out_file,
+          dbConn,
+          identifier = "predicted mRNA",
+          nameBy = "accession",
+          orderBy = "bases",
+          compress = TRUE)
  |===================================================================| 100%

Wrote 61051 sequences.
Time difference of 52.22 secs
```

Access times vary greatly depending on where the database is stored. If the database is small enough, very fast read and write speeds can be obtained by keeping the entire database in memory. For moderate to large sets of sequences it is necessary to store the database on a drive, as demonstrated in the examples above. Note that access speed can be delayed considerably when the database is kept in a shared location, such as on a networked drive. Independently of the database's location, it is important to disconnect after finishing the database session. This would permanently destroy an in-memory database, but only reversibly closes a connection to a drive-based database, as shown here:

```
> dbDisconnect(dbConn)
[1] TRUE
```

Conclusions

DECIPHER is a versatile R package for the curation of biological sequence sets. Its Seqs2DB function constructs a database that can be used to efficiently store DNA, RNA, or amino acid sequences in compressed format. For DNA and RNA, **DECIPHER** employs a custom compression algorithm, called

nbit, that enables fast compression and decompression at a reasonable compression ratio. Using a database, it is possible to construct non-destructive workflows that handle sequences in batches so that they do not need to be kept in memory simultaneously. The SearchDB function can be used to automatically generate complex queries that return sequences from a database table. Collectively, **DECIPHER** functions make it possible to process millions of biological sequences with relative ease. This capability will only become more useful as improvements in sequencing power continue to outpace growth in memory capacity.

Bibliography

S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997. [p352]

S. Deorowicz and S. Grabowski. Data compression for sequencing data. *Algorithms for Molecular Biology*, 8(1):1–13, 2013. [p352, 355]

E. C. Hayden. Technology: The $1,000 genome. *Nature*, 507(7492):294–295, Mar. 2014. [p352]

D. C. Jones, W. L. Ruzzo, X. Peng, and M. G. Katze. Compression of next-generation sequencing reads aided by highly efficient de novo assembly. *Nucleic Acids Research*, 40(22):e171–e171, Dec. 2012. [p355]

H. Pagès, P. Aboyoun, R. Gentleman, and S. DebRoy. *Biostrings: String objects representing biological sequences, and matching algorithms*. R package version 2.38.0. [p352]

B. D. Ripley. Using Databases with R. *R News*, 1(1):18–20, Jan. 2001a. [p353]

B. D. Ripley. Connections. *R News*, 1(1):16–17, Jan. 2001b. [p354]

P. D. Schloss, D. Gevers, and S. L. Westcott. Reducing the Effects of PCR Amplification and Sequencing Artifacts on 16S rRNA-Based Studies. *PloS one*, 6(12):e27310, Dec. 2011. [p352]

S. Wandelt, M. Bux, and U. Leser. Trends in genome compression. *Current Bioinformatics*, 2014. [p356]

E. S. Wright. DECIPHER: harnessing local sequence context to improve protein multiple sequence alignment. *BMC Bioinformatics*, 16(1):1–14, Sept. 2015. [p352]

E. S. Wright and K. H. Vetsigian. DesignSignatures: a tool for designing primers that yields amplicons with distinct signatures. *Bioinformatics*, Jan. 2016. [p352]

E. S. Wright, L. S. Yilmaz, A. M. Corcoran, H. E. Okten, and D. R. Noguera. Automated design of probes for rRNA-targeted fluorescence in situ hybridization reveals the advantages of using dual probes for accurate identification. *Applied and environmental microbiology*, 80(16):5124–5133, July 2014a. [p352]

E. S. Wright, L. S. Yilmaz, S. Ram, J. M. Gasser, G. W. Harrington, and D. R. Noguera. Exploiting extension bias in polymerase chain reaction to improve primer specificity in ensembles of nearly identical DNA templates. *Environmental Microbiology*, 16(5):1354–1365, May 2014b. [p352]

G. Xie, R. DeMarco, R. Blevins, and Y. Wang. Storing biological sequence databases in relational form. *Bioinformatics*, 16(3):288–289, Mar. 2000. [p353]

Erik S. Wright
University of Wisconsin - Madison
330 N Orchard St, Madison WI 53715 USA
eswright@wisc.edu