Using the SRAdb Package to Query the Sequence Read Archive

Jack Zhu*and Sean Davis†

Genetics Branch, Center for Cancer Research,
National Cancer Institute,
National Institutes of Health

January 24, 2013

1 Introduction

High throughput sequencing technologies have very rapidly become standard tools in biology. The data that these machines generate are large, extremely rich. As such, the Sequence Read Archives (SRA) have been set up at NCBI in the United States, EMBL in Europe, and DDBJ in Japan to capture these data in public repositories in much the same spirit as MIAME-compliant microarray databases like NCBI GEO and EBI ArrayExpress.

Accessing data in SRA requires finding it first. This R package provides a convenient and powerful framework to do just that. In addition, SRAdb features functionality to determine availability of sequence files and to download files of interest.

SRA does not currently store aligned reads or any other processed data that might rely on alignment to a reference genome. However, NCBI GEO does often contain aligned reads for sequencing experiments and the SRAdb package can help to provide links to these data as well. In combination with the GEOmetadb and GEOquery packages, these data are also, then, accessible.

2 Getting Started

Since SRA is a continuously growing repository, the SRAdb SQLite file is updated regularly. The first step, then, is to get the SRAdb SQLite file from the online location. The download and uncompress steps are done automatically with a single command, getSRAdbFile.

*zhujack@mail.nih.gov
†sdavis2@mail.nih.gov

```
> library(SRAdb)
> sqlfile <- getSRAdbFile()
```

The default storage location is in the current working directory and the default filename is "SRAmetadb.sqlite"; it is best to leave the name unchanged unless there is a pressing reason to change it. Since this SQLite file is of key importance in SRAdb, it is perhaps of some interest to know some details about the file itself.

```
> file.info('SRAmetadb.sqlite')
```

```
                        size isdir mode
SRAmetadb.sqlite 3315819520 FALSE  644
                                 mtime
SRAmetadb.sqlite 2013-01-24 21:11:35
                                 ctime
SRAmetadb.sqlite 2013-01-24 21:11:35
                                 atime uid gid
SRAmetadb.sqlite 2013-01-24 21:11:35 691 692
                     uname  grname
SRAmetadb.sqlite biocbuild compbio
```

Then, create a connection for later queries. The standard DBI functionality as implemented in RSQLite function `dbConnect` makes the connection to the database. The `dbDisconnect` function disconnects the connection.

```
> sra_con <- dbConnect(SQLite(),sqlfile)
```

For further details, at this time see `help('SRAdb-package')`.

3 Using the SRAdb package

3.1 Interacting with the database

The functionality covered in this section is covered in much more detail in the DBI and RSQLite package documentation. We cover enough here only to be useful. The `dbListTables` function lists all the tables in the SQLite database handled by the connection object `sra_con` created in the previous section.

```
> sra_tables <- dbListTables(sra_con)
> sra_tables
```

```
 [1] "col_desc"        "experiment"
 [3] "metaInfo"        "run"
 [5] "sample"          "sra"
 [7] "sra_ft"          "sra_ft_content"
 [9] "sra_ft_segdir"   "sra_ft_segments"
[11] "study"           "submission"
```

There is also the `dbListFields` function that can list database fields associated with a table.

```
> dbListFields(sra_con,'study')
```

```
 [1] "study_ID"             "study_alias"
 [3] "study_accession"      "study_title"
 [5] "study_type"           "study_abstract"
 [7] "broker_name"          "center_name"
 [9] "center_project_name"  "study_description"
[11] "related_studies"      "primary_study"
[13] "sra_link"             "study_url_link"
[15] "xref_link"            "study_entrez_link"
[17] "ddbj_link"            "ena_link"
[19] "study_attribute"      "submission_accession"
[21] "sradb_updated"
```

Sometimes it is useful to get the actual SQL schema associated with a table. As an example of doing this and using an *RSQLite* shortcut function, `sqliteQuickSQL`, we can get the table schema for the *study* table.

```
> sqliteQuickSQL(sra_con,'PRAGMA TABLE_INFO(study)')
```

```
   cid                 name type notnull
1    0             study_ID REAL       0
2    1           study_alias TEXT       0
3    2      study_accession TEXT       0
4    3           study_title TEXT       0
5    4            study_type TEXT       0
6    5       study_abstract TEXT       0
7    6           broker_name TEXT       0
8    7           center_name TEXT       0
9    8  center_project_name TEXT       0
10   9    study_description TEXT       0
11  10      related_studies TEXT       0
12  11        primary_study TEXT       0
13  12             sra_link TEXT       0
14  13       study_url_link TEXT       0
15  14            xref_link TEXT       0
16  15    study_entrez_link TEXT       0
17  16            ddbj_link TEXT       0
18  17             ena_link TEXT       0
19  18      study_attribute TEXT       0
20  19 submission_accession TEXT       0
```

```
21  20        sradb_updated TEXT       0
   dflt_value pk
1        <NA>  0
2        <NA>  0
3        <NA>  0
4        <NA>  0
5        <NA>  0
6        <NA>  0
7        <NA>  0
8        <NA>  0
9        <NA>  0
10       <NA>  0
11       <NA>  0
12       <NA>  0
13       <NA>  0
14       <NA>  0
15       <NA>  0
16       <NA>  0
17       <NA>  0
18       <NA>  0
19       <NA>  0
20       <NA>  0
21       <NA>  0
```

3.2 Writing SQL queries and getting results

Select 3 records from the *study* table and show the first 5 columns:

```
> rs <- dbGetQuery(sra_con,'select * from study limit 3')
> rs[, 1:3]

  study_ID                study_alias
1        1              Natto BEST195
2        2 Resequence B. subtilis 168
3        3   DLD1_normoxia_nucleosome
  study_accession
1       DRP000001
2       DRP000002
3       DRP000003
```

Get the SRA study accessions and titles from SRA study that study_type contains "Transcriptome". The "%" sign is used in combination with the "like" operator to do a "wildcard" search for the term "Transcriptome" with any number of characters after it.

```
> rs <- dbGetQuery(sra_con, paste( "select study_accession,
+         study_title from study where",
+        "study_description like 'Transcriptome%'",sep=" "))
> rs[1:3,]

  study_accession
1       ERP000233
2       ERP000350
3       ERP000527

1 Identification of the expression profile of Staphylococcus aureus grown in the presenc
2
3                                                          Transcriptome Analysis of the
```

Of course, we can combine programming and data access. A simple `sapply` example shows how to query each of the tables for number of records.

```
> getTableCounts <- function(tableName,conn) {
+   sql <- sprintf("select count(*) from %s",tableName)
+   return(dbGetQuery(conn,sql)[1,1])
+ }
> do.call(rbind,sapply(sra_tables[1:3],
+         getTableCounts, sra_con, simplify=FALSE))

             [,1]
col_desc      129
experiment 205854
metaInfo        2
```

3.3 Conversion of SRA entity types

Large-scale consumers of SRA data might want to convert SRA entity type from one to others, e.g. finding all experiment accessions (SRX, ERX or DRX) and run accessions (SRR, ERR or DRR) associated with 'SRP001007'. Function sraConvert does the conversion with a very fast mapping between entity types.

Covert 'SRP001007' to other possible types in the SRAmetadb.sqlite.

```
> conversion <- sraConvert(c('SRP001007','SRP000931'), sra_con= sra_con)
> conversion[1:3,]

      study submission    sample experiment
1 SRP000931  SRA009053 SRS003463  SRX006134
2 SRP000931  SRA009053 SRS003462  SRX006133
3 SRP000931  SRA009053 SRS003461  SRX006132
```

```
        run
1 SRR018268
2 SRR018267
3 SRR018266
```

Check what SRA types and how many entities in each type in the conversion.

```
> apply(conversion, 2, unique)

$study
[1] "SRP000931" "SRP001007"

$submission
[1] "SRA009053" "SRA009276"

$sample
 [1] "SRS003463" "SRS003462" "SRS003461"
 [4] "SRS003460" "SRS003453" "SRS003456"
 [7] "SRS003458" "SRS003454" "SRS003455"
[10] "SRS003457" "SRS003464" "SRS003459"
[13] "SRS004650"

$experiment
 [1] "SRX006134" "SRX006133" "SRX006132"
 [4] "SRX006131" "SRX006122" "SRX006125"
 [7] "SRX006127" "SRX006123" "SRX006129"
[10] "SRX006124" "SRX006126" "SRX006135"
[13] "SRX006128" "SRX006130" "SRX007396"

$run
 [1] "SRR018268" "SRR018267" "SRR018266"
 [4] "SRR018265" "SRR018256" "SRR018259"
 [7] "SRR018261" "SRR018257" "SRR018263"
[10] "SRR018258" "SRR018260" "SRR018269"
[13] "SRR018262" "SRR018264" "SRR020739"
[16] "SRR020740"
```

3.4 Full text search

Searching by regular table and field specific SQL commands can be very powerful and if you are familiar with SQL language and the table structure. If not, SQLite has a very handy module called Full text search (fts3), which allow users to do Google like search with terms and operators. The function getSRA does Full text search against all fields in a fts3

table with terms constructed with the Standard Query Syntax and Enhanced Query Syntax. Please see http://www.sqlite.org/fts3.html for detail.

Find all run and study combined records in which any given fields has 'breast' and 'cancer' words, including 'breast' and 'cancer' are not next to each other:

```
> rs <- getSRA (search_terms ='breast cancer',
+      out_types=c('run','study'),  sra_con=sra_con)
> dim(rs)

[1] 2852   23
```

If you only wants records containing exact phrase of 'breast cancer', in which 'breast' and 'cancer' have other characters between other than a space:

```
> rs <- getSRA (search_terms ='"breast cancer"',
+      out_types=c('run','study'),  sra_con=sra_con)
> dim(rs)

[1] 2493   23
```

Find all sample records containing words of either 'MCF7' or 'MCF-7':

```
> rs <- getSRA (search_terms ='MCF7 OR "MCF-7"',
+      out_types=c('sample'), sra_con=sra_con)
> dim(rs)

[1] 599  10
```

Find all submissions by GEO:

```
> rs <- getSRA (search_terms ='submission_center: GEO',
+      out_types=c('submission'), sra_con=sra_con)
> dim(rs)

[1] 2388    6
```

Find study records containing a word beginning with 'Carcino':

```
> rs <- getSRA (search_terms ='Carcino*',
+      out_types=c('study'), sra_con=sra_con)
> dim(rs)

[1] 154  12
```

3.5 Get sra or sra-lite data file information

List fastq file ftp or fasp addresses associated with "SRX000122":

```
> listSRAfile (in_acc = c("SRX000122"),
+      sra_con = sra_con, fileType = 'sra')
> listSRAfile (in_acc = c("SRX000122"),
+      sra_con = sra_con, fileType = 'sra', srcType='fasp')
```

The above function does not check file availability, size and date of the sra or sra-lite data files on the server, but the function getSRAinfo does this, which is good to know if you are preparing to download them:

```
> rs <- getSRAinfo (in_acc=c("SRX000122"),
+      sra_con=sra_con)
> rs[1:3,]
```

Next you might want to download sra or sra-lite data files from the ftp site. The getSRAfile function will download all available sra or sra-lite data files associated with "SRR000648" and "SRR000657" from NCBI SRA ftp site to a new folder in current directory:

```
> getSRAfile( in_acc = c("SRR000648","SRR000657"),
+      sra_con = sra_con, destDir = getwd(),
+          fileType = 'litesra' )
```

Then downloaded .sra or .lite.sra can be easily converted into fastq files using fastq-dump in SRA Toolkit (http://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?cmd=show&f=software&m=software&s):

```
> ## system ("fastq-dump SRR000648.lite.sra")
```

Or users can directly download fastq files from EBI using ftp protocol:

```
> getFASTQinfo( in_acc = c("SRR000648","SRR000657"), srcType = 'ftp' )
> getSRAfile( in_acc, sra_con, destDir = getwd(),
+      fileType = 'fastq', srcType = 'ftp' )
```

3.6 Download SRA data files using fasp protocol

Curretly both NCBI and EBI supports fasp downloading of SRA data files, which has several advantages over ftp protocol, including high-speed transfering large files over long distance. Please check EBI or NCBI web site or Aspera (http://www.asperasoft.com/) for details. SRAdb has indcluded two wraper functions for using ascp command line program to download files by fasp protocol, which is included in in Aspera Connect software. But, due to complexity of installaton of the software and options within it, the funcitons develpped here ask users to supply main ascp comands.

Download fastq files from EBI ftp siteusing fasp protocol:

```
> ## List  fasp addresses of SRA fastq files associated with "SRX000122"
> listSRAfile (in_acc = c("SRX000122"), sra_con = sra_con,
+      fileType = 'fastq', srcType='fasp')
> ## download fastq files using fasp protocol:
> ascpCMD <-  'ascp -QT -l 300m -i
+    /usr/local/aspera/connect/etc/asperaweb_id_dsa.putty'
> getSRAfile( in_acc = c("SRX000122"), sra_con,
+      fileType = 'fastq', srcType = 'fasp',  ascpCMD = ascpCMD )
```

Download .lite.sra files from NCBI ftp site using fasp protocol:

```
> ## List  fasp addresses of SRA litesra files associated with "SRX000122"
> listSRAfile (in_acc = c("SRX000122"),
+      sra_con = sra_con, fileType = 'litesra', srcType='fasp')
> ## download litesra files using fasp protocol:
> ascpCMD <- 'ascp -QT -l 300m -i
+     /usr/local/aspera/connect/etc/asperaweb_id_dsa.putty'
> ## common ascpCMD for a system with Mac OS X:
> #ascpCMD <- "'/Applications/Aspera Connect.app
> #/Contents/Resources/ascp' -QT -l 300m -i
> #'/Applications/Aspera Connect.app/Contents/
> #Resources/asperaweb_id_dsa.putty'"
> sraFiles <- ascpSRA( in_acc = c("SRX000122"),
+      sra_con, ascpCMD, fileType = 'litesra', destDir=getwd() )
```

4 Interactive views of sequence data

Working with sequence data is often best done interactively in a genome browser, a task not easily done from R itself. We have found the Integrative Genomics Viewer (IGV) a high-performance visualization tool for interactive exploration of large, integrated datasets, increasing usefully for visualizing sequence alignments. In SRAdb, functions `startIGV`, `load2IGV` and `load2newIGV` provide convenient functionality for R to interact with IGV. Note that for some OS, these functions might not work or work well.

Launch IGV with 2 GB maximum usable memory support:

```
> startIGV("mm")
```

IGV offers a remort control port that allows R to communicate with IGV. The current command set is fairly limited, but it does allow for some IGV operations to be performed in the R console. To utilize this functionality, be sure that IGV is set to allow communication via the "enable port" option in IGV preferences. To load BAM files to IGV and then manipulate the window:

```
> exampleBams = file.path(system.file('extdata',package='SRAdb'),
+   dir(system.file('extdata',package='SRAdb'),pattern='bam$'))
> sock <- IGVsocket()
> IGVgenome(sock, 'hg18')
> IGVload(sock, exampleBams)
> IGVgoto(sock, 'chr1:1-1000')
> IGVsnapshot(sock)
```

5 Graphic view of SRA entities

Due to the nature of SRA data and its design, sometimes it is hard to get a whole picture of the relationship between a set of SRA entities. Functions of `entityGraph` and `sraGraph` in this package generate graphNEL objects with edgemode='directed' from input data.frame or directly from search terms, and then the `plot` function can easily draw a graph.

Create a graphNEL object from SRA accessions, which are full text search results of terms 'primary thyroid cell line'

```
> acc <- getSRA (search_terms ='primary thyroid cell line',
+      out_types=c('sra'), sra_con=sra_con, acc_only=TRUE)
> g <- entityGraph(acc)
> attrs <- getDefaultAttrs(list(node=list(
+          fillcolor='lightblue', shape='ellipse')))
> plot(g, attrs= attrs)
```

Create a graphNEL object directly from full text search results of terms 'primary thyroid cell line'

```
> g <- sraGraph('primary thyroid cell line', sra_con)
> library(Rgraphviz)
> attrs <- getDefaultAttrs(list(node=list(
+         fillcolor='lightblue', shape='ellipse')))
> plot(g, attrs=attrs)
```

It's considered good practise to explicitely disconnect from the database once we are done with it:

```
> dbDisconnect(sra_con)
```

```
[1] TRUE
```

6 sessionInfo

```
> toLatex(sessionInfo())
```

- R version 2.15.2 (2012-10-26), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=C`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: DBI 0.2-5, RCurl 1.95-3, RSQLite 0.11.2, SRAdb 1.12.1, bitops 1.0-5, graph 1.36.1
- Loaded via a namespace (and not attached): Biobase 2.18.0, BiocGenerics 0.4.0, GEOquery 2.24.0, XML 3.95-0.1, stats4 2.15.2, tools 2.15.2