

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

April 30, 2018

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 currently store aligned reads or other processed data that relies on alignment to a reference genome. Please refer to the SRA handbook (<http://www.ncbi.nlm.nih.gov/books/NBK47537/>) for details. NCBI GEO also 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



Figure 1: A graphical representation (sometimes called an *Entity-Relationship Diagram*) of the relationships between the main tables in the SRAdB package.

```
> library(SRAdb)
> sqlfile <- 'SRAmetadb.sqlite'
> if(!file.exists('SRAmetadb.sqlite')) 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. Note: the above downloading and uncompressing steps could take quite a few moments due to file size, depending on your network bandwidth. If interested, it can be timed using the following commands:

```
> timeStart <- proc.time()
> sqlfile <- getSRADBFile()
> proc.time() - timeStart
```

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 36931305472 FALSE  644
              mtime
SRAmetadb.sqlite 2018-04-30 19:13:50
              ctime
SRAmetadb.sqlite 2018-04-30 19:13:50
              atime uid gid
SRAmetadb.sqlite 2018-04-30 19:13:50 1005 1005
              uname grname
SRAmetadb.sqlite biocbuild biocbuild
```

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(RSQLite(), 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. A simplified illustration of the relationship between the SRA main data types is shown in the Figure 1.

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

[1] "col_desc"          "experiment"
[3] "fastq"             "metaInfo"
[5] "run"               "sample"
[7] "sra"               "sra_ft"
[9] "sra_ft_content"    "sra_ft_segdir"
[11] "sra_ft_segments"   "study"
[13] "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. Here, we get the table schema for the *study* table:

```
> dbGetQuery(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

The table "col_desc" contains information of filed name, type, descritption and default values:

```
> colDesc <- colDescriptions(sra_con=sra_con)[1:5,]
> colDesc[, 1:4]
```

	col_desc_ID	table_name	field_name
1	1	submission	ID
2	2	submission	accession
3	3	submission	alias

```

4          4 submission submission_comment
5          5 submission                files
      type
1      int
2 varchar
3 varchar
4      text
5      text

```

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 study_accession
1          1    DRP000001    DRP000001
2          2    DRP000002    DRP000002
3          3    DRP000003    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      DRP002494
2      DRP002820
3      DRP002612

                                study_title
1      Allium fistulosum transcriptome sequencing
2 Transcriptome sequence of planarian Dugesia japonica
3      Bursaphelenchus xylophilus transcriptome

```

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[c(2,4,5,11,12)],
+      getTableCounts, sra_con, simplify=FALSE))
```

```
      [,1]
experiment 4134148
metaInfo    2
run         4737881
sra_ft_segments 326271
study       140009
```

Get some high-level statistics could be to helpful to get overall idea about what data are available in the SRA database. List all study types and number of studies contained for each of the type:

```
> rs <- dbGetQuery(sra_con, paste( "SELECT study_type AS StudyType,
+      count( * ) AS Number FROM `study` GROUP BY study_type order
+      by Number DESC ", sep=""))
> rs
```

	StudyType	Number
1	Whole Genome Sequencing	45964
2	Other	38488
3	<NA>	23243
4	Transcriptome Analysis	16147
5	Metagenomics	14415
6	Population Genomics	763
7	Epigenetics	607
8	Exome Sequencing	230
9	Cancer Genomics	110
10	Pooled Clone Sequencing	31
11	Synthetic Genomics	9
12	Transcriptome Sequencing	1
13	Whole Genome Sequencing	1

List all Instrument Models and number of experiments for each of the Instrument Models:

```
> rs <- dbGetQuery(sra_con, paste( "SELECT instrument_model AS
+      'Instrument Model', count( * ) AS Experiments FROM `experiment`
+      GROUP BY instrument_model order by Experiments DESC", sep=""))
> rs
```

	Instrument Model
1	Illumina HiSeq 2000
2	<NA>
3	Illumina MiSeq
4	Illumina HiSeq 2500
5	454 GS FLX Titanium
6	Illumina Genome Analyzer II
7	NextSeq 500
8	Illumina Genome Analyzer IIX
9	HiSeq X Ten
10	454 GS FLX
11	unspecified
12	Ion Torrent PGM
13	Illumina HiSeq 4000
14	454 GS Junior
15	Illumina Genome Analyzer
16	454 GS FLX+
17	Illumina HiSeq 1000
18	PacBio RS II
19	PacBio RS
20	AB SOLiD 4 System
21	Illumina HiSeq 1500
22	454 GS
23	Ion Torrent Proton
24	Illumina HiSeq 3000
25	AB 5500xl Genetic Analyzer
26	Complete Genomics
27	Helicos HeliScope
28	Illumina HiScanSQ
29	NextSeq 550
30	AB SOLiD System 3.0
31	AB 5500 Genetic Analyzer
32	AB 3730xL Genetic Analyzer
33	454 GS 20
34	MinION
35	AB SOLiD System
36	AB SOLiD System 2.0
37	Illumina MiniSeq
38	AB SOLiD 3 Plus System
39	BGISEQ-500
40	AB 3730 Genetic Analyzer
41	AB 5500xl-W Genetic Analysis System

42	AB SOLiD 4hq System
43	AB 3130 Genetic Analyzer
44	AB 3500 Genetic Analyzer
45	AB 3130xL Genetic Analyzer
46	Sequel
47	Ion S5
48	HiSeq X Five
49	Illumina NextSeq 500
50	AB 3500xL Genetic Analyzer
51	454 GS FLX
52	AB SOLiD PI System
53	Illumina Genome Analyzer IIX
54	AB 310 Genetic Analyzer
55	Illumina MiSeq

Experiments

1	1352089
2	789006
3	633708
4	572551
5	148477
6	112444
7	89496
8	63561
9	53236
10	49412
11	32933
12	31574
13	29909
14	23956
15	19075
16	16702
17	14584
18	14421
19	12056
20	11025
21	9758
22	8704
23	8420
24	6082
25	5138
26	4276
27	3937

28	3675
29	3259
30	2561
31	2111
32	1168
33	986
34	857
35	501
36	483
37	361
38	323
39	319
40	260
41	203
42	153
43	111
44	71
45	55
46	53
47	41
48	18
49	18
50	14
51	10
52	3
53	2
54	1
55	1

List all types of library strategies and number of runs for each of them:

```
> rs <- dbGetQuery(sra_con, paste( "SELECT library_strategy AS
+      'Library Strategy', count( * ) AS Runs FROM `experiment`
+      GROUP BY library_strategy order by Runs DESC", sep=""))
> rs
```

	Library Strategy	Runs
1	WGS	1092136
2	<NA>	789006
3	AMPLICON	690070
4	RNA-Seq	627436
5	OTHER	314123
6	WXS	237358
7	CLONE	89555

8	ChIP-Seq	82824
9	POOLCLONE	53738
10	Bisulfite-Seq	32335
11	SELEX	25875
12	miRNA-Seq	22405
13	WGA	21078
14	RAD-Seq	10290
15	Targeted-Capture	8427
16	ATAC-seq	6060
17	ncRNA-Seq	5360
18	EST	3596
19	RIP-Seq	3264
20	DNase-Hypersensitivity	2868
21	MeDIP-Seq	2442
22	MNase-Seq	2328
23	MRE-Seq	2073
24	FL-cDNA	2010
25	Tn-Seq	1886
26	WCS	1809
27	MBD-Seq	1690
28	CLONEEND	523
29	FAIRE-seq	477
30	Hi-C	323
31	CTS	282
32	other	214
33	Synthetic-Long-Read	195
34	FINISHING	40
35	ChIA-PET	30
36	VALIDATION	22

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" and "SRP000931". Function `sraConvert` does the conversion with a very fast mapping between entity types.

Covert "SRP001007" and "SRP000931" 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 SRS003462  SRX006133
```

```

2 SRP000931  SRA009053 SRS003463  SRX006134
3 SRP000931  SRA009053 SRS003457  SRX006126
      run
1 SRR018267
2 SRR018268
3 SRR018260

```

Check what SRA types and how many entities for each type:

```

> apply(conversion, 2, unique)

$study
[1] "SRP000931" "SRP001007"

$submission
[1] "SRA009053" "SRA009276"

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

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

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

```

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 )
> dim(rs)

[1] 40530    23

> rs <- getSRA( search_terms = "breast cancer",
+               out_types = c("submission", "study", "sample",
+                             "experiment", "run"), sra_con )
> # get counts for some information interested
> apply( rs[, c('run','sample','study_type','platform',
+               'instrument_model')], 2, function(x)
+       {length(unique(x))} )
```

run	sample
40530	27091
study_type	platform
9	10
instrument_model	
36	

```
>
```

If you only want SRA records containing exact phrase of "breast cancer", in which "breast" and "cancer" do not have other characters between other than a space:

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

```
[1] 28887    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 )
> dim(rs)
```

```
[1] 5122    10
```

Find all submissions by GEO:

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

[1] 23797      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] 1257     12
```

3.5 Download SRA data files

List ftp addresses of the fastq files associated with "SRX000122":

```
> rs = listSRAfile( c("SRX000122"), sra_con, fileType = 'sra' )
```

The above function does not check file availability, size and date of the sra 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 ( c("SRX000122"), sra_con, sraType = "sra" )
> rs[1:3,]
```

```
1 ftp://ftp-trace.ncbi.nlm.nih.gov/sra/sra-instant/reads/ByRun/sra/SRR/SRR000/SRR000648/
2 ftp://ftp-trace.ncbi.nlm.nih.gov/sra/sra-instant/reads/ByRun/sra/SRR/SRR000/SRR000649/
3 ftp://ftp-trace.ncbi.nlm.nih.gov/sra/sra-instant/reads/ByRun/sra/SRR/SRR000/SRR000650/
  experiment      study      sample      run
1  SRX000122 SRP000098 SRS000290 SRR000648
2  SRX000122 SRP000098 SRS000290 SRR000649
3  SRX000122 SRP000098 SRS000290 SRR000650
  size(KB)      date
1      281 Jan 19 2012
2    130940 Jan 19 2012
3      844 Jan 19 2012
```

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

```
> getSRAfile( c("SRR000648","SRR000657"), sra_con, fileType = 'sra' )
```

```

      run      study      sample experiment
1 SRR000648 SRP000098 SRS000290 SRX000122
2 SRR000657 SRP000098 SRS000290 SRX000122

```

```

1 ftp://ftp-trace.ncbi.nlm.nih.gov/sra/sra-instant/reads/ByRun/sra/SRR/SRR000/SRR000648/
2 ftp://ftp-trace.ncbi.nlm.nih.gov/sra/sra-instant/reads/ByRun/sra/SRR/SRR000/SRR000657/

```

Then downloaded sra data files can be easily converted into fastq files using fastq-dump in SRA Toolkit (<http://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=software>):

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

Or directly download fastq files from EBI using ftp protocol:

```

> getFASTQinfo( c("SRR000648","SRR000657"), sra_con, srcType = 'ftp' )
> getSRAfile( c("SRR000648","SRR000657"), sra_con, fileType = 'fastq' )

```

3.6 Download SRA data files using fasp protocol

Currently both NCBI and EBI supports fasp protocol for downloading SRA data files, which has several advantages over ftp protocol, including high-speed transferring large files over long distance. Please check EBI or NCBI web site or Aspera (<http://www.asperasoft.com/>) for details. SRADB has included two wrapper functions for using ascp command line program (fasp protocol) to download SRA data files from either the NCBI or EBI, which is included in Aspera Connect software. But, due to complexity of installation of the software and options within it, the functions developed here ask users to supply main ascp commands.

Download fastq files from EBI ftp site using fasp protocol:

```

> ## List fasp addresses for associated fastq files:
> listSRAfile ( c("SRX000122"), sra_con, fileType = 'fastq', srcType='fasp')
> ## get fasp addresses for associated fastq files:
> getFASTQinfo( c("SRX000122"), sra_con, srcType = 'fasp' )
> ## download fastq files using fasp protocol:
> # the following ascpCMD needs to be constructed according custom
> # system configuration
> # common ascp installation in a Linux system:
> ascpCMD <- 'ascp -QT -l 300m -i
+ /usr/local/aspera/connect/etc/asperaweb_id_dsa.putty'
> ## common ascpCMD for a Mac OS X system:
> # ascpCMD <- "'/Applications/Aspera Connect.app/Contents/
> # Resources/ascp' -QT -l 300m -i '/Applications/
> # Aspera Connect.app/Contents/Resources/asperaweb_id_dsa.putty'"

```

```
>
> getSRAfile( c("SRX000122"), sra_con, fileType = 'fastq',
+           srcType = 'fasp', ascpCMD = ascpCMD )
```

Download sra files from NCBI using fasp protocol:

```
> ## List fasp addresses of sra files associated with "SRX000122"
> listSRAfile( c("SRX000122"), sra_con, fileType = 'sra', srcType='fasp')
> ## download sra files using fasp protocol
> getSRAfile( c("SRX000122"), sra_con, fileType = 'sra',
+           srcType = 'fasp', ascpCMD = ascpCMD )
```

The downloading message will show significant faster downloading speed than the ftp protocol:

```
' SRR000658.sra 100Completed: 159492K bytes transferred in 5 seconds (249247K bits/sec),
in 1 file. ... '
```

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 remote 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)
```

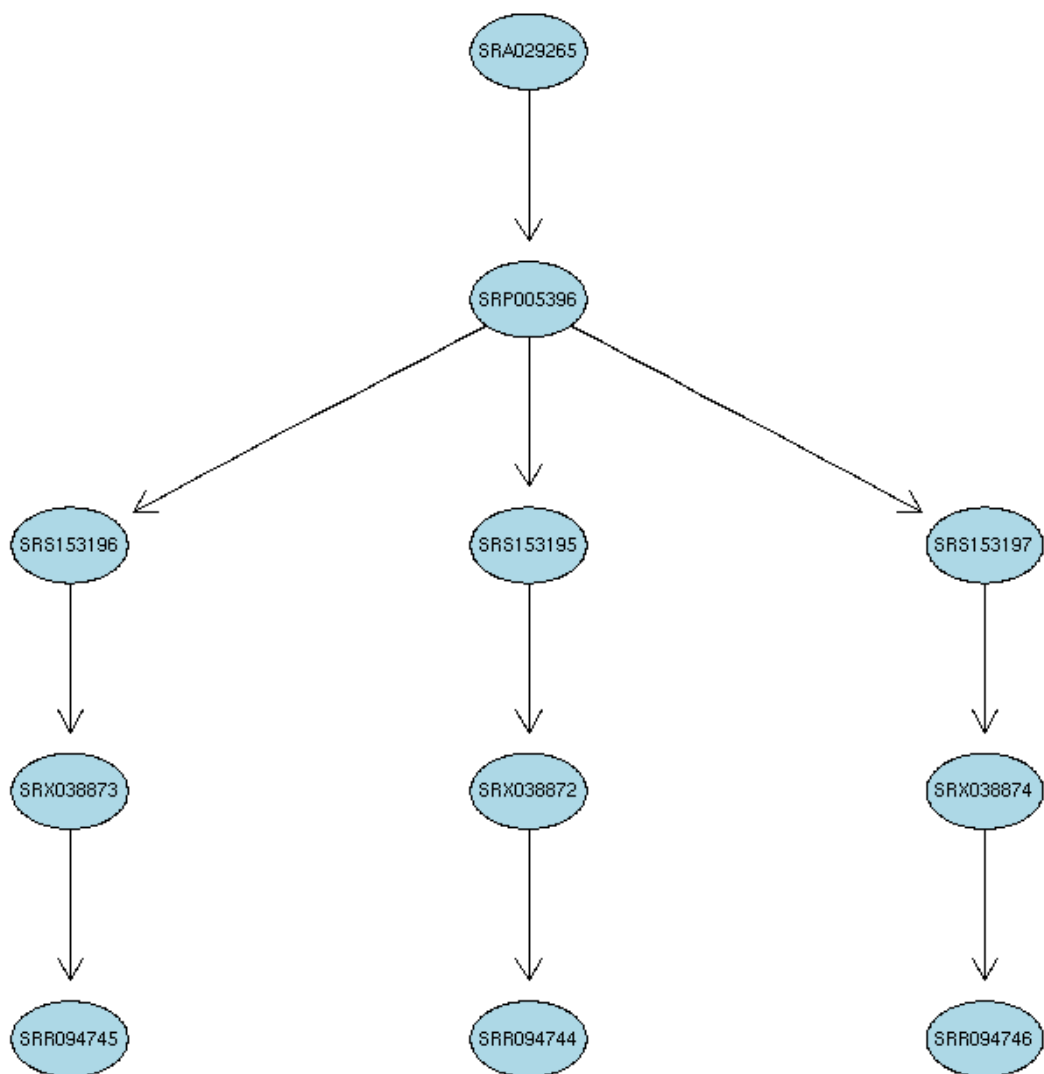



Figure 2: A graphical representation of the relationships between the SRA entities.

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 diagram.

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

```
> library(SRAdb)
> library(Rgraphviz)
> g <- sraGraph('primary thyroid cell line', sra_con)
> attrs <- getDefaultAttrs(list(node=list(
+   fillcolor='lightblue', shape='ellipse'))))
> plot(g, attrs=attrs)
> ## similiar search as the above, returned much larger data.frame and graph is too cl
> g <- sraGraph('Ewing Sarcoma', sra_con)
> plot(g)
>
```

Please see the Figure 2 for an example diagram.

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

```
> dbDisconnect(sra_con)
```

6 Example use case

This sesection will use the functionalities in the `SRAdb` package to explore data from the 1000 genomes project. Mainly,

1. Get some statistics of meta data and data files from the 1000 genomes project using the `SRAdb`
2. Download data files
3. Load bam files into the IGV from R
4. Create some snapshots programmatically from R

```
> library(SRAdb)
> setwd('1000g')
> if( ! file.exists('SRAmetadb.sqlite') ) {
+   sqlfile <- getSRAdbFile()
+ } else {
+   sqlfile <- 'SRAmetadb.sqlite'
+ }
> sra_con <- dbConnect(SQLite(),sqlfile)
> ## get all related accessions
```

```

> rs <- getSRA( search_terms = '"1000 Genomes Project"',
+             sra_con=sra_con, acc_only=TRUE)
> dim(rs)
> head(rs)
> ## get counts for each data types
> apply( rs, 2, function(x) {length(unique(x))} )

```

After you decided what data from the 1000 Genomes, you would like to download data files from the SRA. But, it might be helpful to know file size before downloading them:

```

> runs <- tail(rs$run)
> fs <- getSRAinfo( runs, sra_con, sraType = "sra" )

```

Now you can download the files through ftp protocol:

```

> getSRAfile( runs, sra_con, fileType='sra', srcType = "ftp" )

```

Or, you can download them through fasp protocol:

```

> ascpCMD <- "'/Applications/Aspera Connect.app/Contents/Resources/ascp' -QT -l 300m -"
> sra_files = getSRAfile( runs, sra_con, fileType='sra', srcType = "fasp", ascpCMD =

```

Next you might want to convert the downloaded sra files into fastq files:

```

> for( fq in basename(sra_files$fasp) ) {
+   system ("fastq-dump SRR000648.lite.sra")
+ }

```

... to be completed.

7 sessionInfo

- R version 3.5.0 (2018-04-23), x86_64-pc-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=en_US.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C
- Running under: Ubuntu 16.04.4 LTS
- Matrix products: default
- BLAS: /home/biocbuild/bbs-3.7-bioc/R/lib/libRblas.so
- LAPACK: /home/biocbuild/bbs-3.7-bioc/R/lib/libRlapack.so

- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, utils
- Other packages: BiocGenerics 0.26.0, RCurl 1.95-4.10, RSQLite 2.1.0, SRADB 1.42.0, bitops 1.0-6, graph 1.58.0
- Loaded via a namespace (and not attached): Biobase 2.40.0, DBI 0.8, GEOquery 2.48.0, R6 2.2.2, Rcpp 0.12.16, assertthat 0.2.0, bindr 0.1.1, bindrcpp 0.2.2, bit 1.1-12, bit64 0.9-7, blob 1.1.1, compiler 3.5.0, digest 0.6.15, dplyr 0.7.4, glue 1.2.0, hms 0.4.2, limma 3.36.0, magrittr 1.5, memoise 1.1.0, pillar 1.2.2, pkgconfig 2.0.1, purrr 0.2.4, readr 1.1.1, rlang 0.2.0, stats4 3.5.0, tibble 1.4.2, tidyr 0.8.0, tools 3.5.0, xml2 1.2.0