Annotation Data Package to Flat Files

Seth Falcon

January 17, 2006

1 Overview

This vignette documents the process of converting a part of the *hgu95av2* Affymetrix annotation package into flat files. The flat files are formatted for easy import into SQLite.

We will extract data for accession number, GO annotation, and PubMed reference and write the data to plain-text delimited files (so called "flat-files") that can be used to import the data into a SQLite database.

The techniques demonstrated are of general use when using R to convert arbitrary data into a format suitable for database import.

2 General strategy

In R, an `environment` is a built-in hash table data structure that maps strings, often called keys, to values which can be any R object. Like other annotation packages, the *hgu95av2* package contains a number of R environments that hold the annotation data.

Our general strategy will be to convert data stored in one or more environments to a `data.frame` which we will then write to disk in a delimited plain-text format. Each data frame corresponds to a single delimited text file. At import time, each text file will correspond to a single table in the database.

Environments in the annotation package that map each key to a single identifier, such as the `hgu95av2ACCNUM` environment, have a direct interpretation as a data frame and are easy to convert. For environments that map each key to a vector of identifiers, such as the `hgu95av2PMID` environment, we will need to do some data manipulation in R to "unroll" the data into tabular format suitable for a data frame.

Exercise 1
Explore the environments provided by the hgu95av2 package. Find out how many keys are in each environment. Determine which environments contain simple mappings and which will require more involved data manipulation in order to convert to tabular format. Hint: use the help system (try `help(package="hgu95av2")`*), along with the functions* `ls`*,* `get`*, and* `length`*.*

3 Utility functions

We define a utility function `dumpVects` to help with the conversion of R data structures to flat files appropriate for importing into a SQLite database. The `dumpVects` function takes any number of vectors of the same length, converts them into a `data.frame` and then uses `write.table` to write the data frame to a text file formatted for import into SQLite.

Because we want to be able to pass any number of vectors to the `dumpVects` function, we use R's ... argument specifier that allows a variable number of arguments to be specified.

```
> makeDf <- function(...) {
+     df <- list(...)
+     if (length(df) < 1)
+         stop("Invalid args")
+     lens <- sapply(df, length)
+     if (length(unique(lens)) > 1)
+         stop("Invalid args: all must be vectors of the same length")
+     attributes(df) <- list(class = "data.frame", row.names = seq(length = lens[1]),
+         names = paste("V", 1:length(df), sep = ""))
+     df
+ }
> dumpVects <- function(fname, ...) {
+     df <- makeDf(...)
+     write.table(df, row.names = FALSE, col.names = FALSE, sep = "|",
+         quote = FALSE, file = fname)
+ }
```

Exercise 2
Create a three variables containing vectors of the same length and use them as arguments to the `dumpVects` function in order to generate a delimited text file. Look at the text file you created to understand the output format.

4 Data conversion

4.1 Accession number

The `hgu95av2ACCNUM` environment maps Affymetrix probe ids to GenBank accession number. Each Affy id maps to a single GenBank accession number which makes conversion to the tabular format easy.

```
> library("hgu95av2")
> acc <- as.list(hgu95av2ACCNUM)
> dumpVects(fname = "hgu95av2-acc.txt", names(acc), unlist(acc))
```

4.2 PubMed identifier

The `hgu95av2PMID` environment gives the PubMed id for each publication that references a given Affymetrix probe id. The environment maps each Affymetrix probe id to a vector of PubMed identifiers. Since each Affymetrix id can map to many PubMed ids, we will have to unroll the data before converting to a data frame. After unrolling, the data will consist of pairs of Affymetrix ids and PubMed ids. A given Affymetrix id can appear many times, once for PubMed id with which is has an association.

To unroll the data, we use the lengths of the PubMed id vectors to repeat the given Affy id the appropriate number of times. Here is a simple example of the *unroll procedure*:

```
> dat <- list(a = 1:4, b = 10:11, c = 111)
> dat

$a
[1] 1 2 3 4

$b
[1] 10 11

$c
[1] 111
```

In our example data, `a`, `b`, and `c` represent Affymetrix ids and the values associated with each represent PubMed ids.

```
> lens <- sapply(dat, length)
> keys <- rep(names(dat), lens)
> vals <- unlist(dat)
> data.frame(key = I(keys), value = I(vals), row.names = 1:length(vals))

  key value
1   a     1
2   a     2
3   a     3
4   a     4
5   b    10
6   b    11
7   c   111
```

The Affymetrix ids (`names(dat)`) have each been repeated according to the number of PubMed ids associated with it (`lens[i]`). The data frame constructed has a row for each Affymetrix/PubMed id pair.

Exercise 3
Use the unroll procedure and `dumpVects` *to convert the* `hgu95av2PMID` *environment to a delimited text file named* `hgu95av2-pmid.txt`.

4.3 GO ontology lookup table

This lookup table maps the Gene Ontology (GO) code names to their descriptive names. This lookup table is not actually a part of the *hgu95av2* annotation package, but it will make our data self-documenting to include it.

```
> codes <- c("BP", "CC", "MF")
> desc <- c("Biological Process", "Cell Cycle", "Molecular Function")
> dumpVects(fname = "hgu95av2-goOntName.txt", codes, desc)
```

4.4 GO identifier

The `hgu95av2GO` environment maps each Affymetrix probe id to a list of lists describing the GO identifiers associated with the probe id. We will represent this data in two tables. One table, `hgu95av2-go.txt`, will map Affy ids to GO id and evidence codes. The second table, `hgu95av2-goId2Ontol.txt`, will map GO ids to the appropriate GO ontology code to which the id belongs (BP, CC, MF).

Some Affy ids do not have any GO annotations and a missing value indicator, NA, is given as the value in the environment. These need special treatment during the unroll procedure.

```
> go <- as.list(hgu95av2GO)
> affyId <- names(go)
> affyId <- rep(affyId, sapply(go, length))
> goId <- unlist(sapply(go, function(x) {
+     if (length(x) == 1 && is.na(x))
+         return(x)
+     else return(names(x))
+ }))
> evi <- unlist(sapply(go, function(x) {
+     if (length(x) == 1 && is.na(x))
+         return(x)
+     sapply(x, function(y) y$Evidence)
+ }))
> dumpVects(fname = "hgu95av2-go.txt", affyId, goId, evi)
```

Exercise 4
Create `hgu95av2-goId2Ontol.txt`*, a table mapping GO id to ontology code. Take care to deal with missing values (NA's) and repeated GO ids.*

Finally, we create a lookup table mapping GO evidence codes to their descriptions.

```
> codes <- c("IMP", "IGI", "IPI", "ISS", "IDA", "IEP", "IEA", "TAS",
+     "NAS", "ND", "IC")
> descs <- c("inferred from mutant phenotype", "inferred from genetic interaction",
+     "inferred from physical interaction", "inferred from sequence similarity",
```

```
+     "inferred from direct assay", "inferred from expression pattern",
+     "inferred from electronic annotation", "traceable author statement",
+     "non-traceable author statement", "no biological data available",
+     "inferred by curator")
> dumpVects(fname = "hgu95av2-goEvidence.txt", codes, descs)
```

Exercise 5
Improve the handling of missing values so that the value written to the text files is determined by a variable NASTRING.

5 Importing flat files into a SQLite database

SQLite's command line tool has an `.import` command for importing data into a single database table. The table must exist in the database prior to calling `.import`.

Here we import the text files created above into a SQLite database. This requires that you have the SQLite command line tool, `sqlite3`, installed on your system and available in your `PATH`.

```
> library("RBioinf")
> schemaFile <- file.path("schema", "affy-annotation-schema.sql")
> schemaFile <- system.file(schemaFile, package = "RBioinf")
> if (schemaFile == "") stop("the database schema file was not found.")
> sqlite <- system.file("sqlite/bin/sqlite3", package = "RSQLite")
> if (identical(sqlite, "")) sqlite <- "sqlite3"
> dbName <- "hgu95av2-sqlite.db"
> if (file.exists(dbName)) file.remove(dbName)
> importCmd <- paste(sqlite, dbName, "<", schemaFile)
> res <- system(importCmd)
> if (res != 0) stop("SQLite DB import failed.  This is the command that failed:\n",
+     importCmd)
```