Archiving data for use with *ontoTools*

VJ Carey `<stvjc@channing.harvard.edu>`

October 18, 2010

Contents

1 Introduction

Effective use of *ontoTools* depends upon a number of potentially laborious computations involving large scale metadata resources. Using Bioconductor's conventions, the *org.Hs.eg.db* package and the *GO* package, one can create 'object-ontology complexes' (OOCs) (or annotated corpora) for use in a variety of investigations.

In general, the *ontoTools* maintainer will attempt to supply the most important complexes with the package, to secure uniformity of results across diverse applications. However, users may need to create their own OOCs or derived concept probability scores. This vignette indicates how this can be done.

As noted, in realistic applications, the computations are laborious. Hence we will protect the reader from intensive computation by conditioning out the slowest computations. If you want to run this vignette directly, modify the variable `DontRun` to have value `FALSE` in the next chunk.

```
> DontRun <- TRUE
```

2 Nomenclature details

At this point, a standard nomenclature is lacking. But the following conventions may be of some use. The data subdirectory of package ontoTools will be guaranteed to hold the following objects, where `[x.y]` evaluates to a Bioconductor release tag.

```
goMFgraph.[x.y].rda    -- graph::graphNEL representing goMF DAG
goMFamat.[x.y].rda     -- namedSparse representing accessibility matrix
                          of goMFgraph
LL2GOMFooMap.[x.y].rda -- namedSparse representing map from LocusLink
                          to GO MF, using org.Hs.eg.db
LL2GOMFcp.[x.y].rda    -- vector of concept probabilities for LL2GOMF
```

3 Illustration: working with LocusLink and GO molecular function

To begin, we attach the current *org.Hs.eg.db* package and extract all the LocusLink tags.

```
> library(org.Hs.eg.db)
> lltags <- ls(org.Hs.egGO)
```

We see that there are 44811 tags for human loci:

```
> print(length(lltags))
```

```
[1] 44811
```

We now obtain the GO annotations for these tags. This is accomplished by saving annotations in a list indexed by loci, and then unlisting. We will store the restricted (molecular function annotation) mapping in an environment `hllgoEnv`.

```
> kvmap <- list()
> hllgoEnv <- new.env(hash = TRUE)
> library(GO.db)
```

We want to confine attention to molecular function (MF) terms.

```
> GOtags <- ls(GOTERM)
> library(Biobase)
> library(annotate)
> GOlabs <- mget(GOtags, GOTERM)
> GOMFtags <- GOtags[sapply(GOlabs, Ontology) == "MF"]
```

Now we iterate over loci, checking for presence of tag annotations in the MF ontology before saving to the `kvmap` list.

```
> if (!(DontRun)) {
+     cat(length(lltags))
+     for (i in 1:length(lltags)) {
+         if (i%%200 == 0)
+             cat(i)
+         tmp <- get(lltags[i], org.Hs.egGO)
+         tmp = gsub("@.*$", "", tmp)
+         tmp <- tmp[tmp %in% GOMFtags]
+         if (length(tmp) > 0) {
+             kvmap[[lltags[i]]] <- tmp
+             assign(lltags[i], tmp, env = hllgoEnv)
+         }
+     }
+ }
```

The resulting map has 44811 elements. These define the rows of the OOC map matrix (mapping objects to terms).

We now get the unique GO target tags. These define the columns of the OOC map.

```
> if (!(DontRun)) {
+     print(length(kvmap))
+     gotargs <- sort(unique(unlist(kvmap)))
+     llused <- names(kvmap)
+     print(length(gotargs))
+ }
```

Now we use the *ontoTools* utility that creates a named sparse matrix out of an object-term key-value environment. The `otkvEnv2namedSparse` function is quite slow for the 10000 by 2000 application with late 2003 LL and GO.

```
> library(ontoTools)
> if (!DontRun) {
+     LL2GOMFooMap.1.18 <- otkvEnv2namedSparse(llused, gotargs,
+         hllgoEnv)
+     save(LL2GOMFooMap.1.18, file = "LL2GOMFooMap1.18.rda", compress = TRUE)
+     save.image()
+ }
> if (DontRun) data(LL2GOMFooMap.1.18)
```

4 The GO ontology

In *ontoTools*, an ontology is a lightly annotated DAG. The package includes an function `buildGOgraph`, which works by default on the environment GO::GOMFPARENTS.

```
> if (!DontRun) {
+     goMFgraph.1.18 <- buildGOgraph()
+ } else data(goMFgraph.1.18)
> save.image()
```

This is then installed in an `ontoTools::ontology` object via the following steps

```
> if (!DontRun) {
+     go1.18DAG <- new("rootedDAG", root = "GO:0003674", DAG = goMFgraph.1.18)
+     GOMF1.18 <- new("ontology", name = "GOMF", version = "bioc 2.0",
+         rDAG = go1.18DAG)
+ }
> if (!DontRun) {
+     goMFamat.1.18 <- accessMat(GOMF1.18)
+ } else {
+     data(goMFamat.1.18)
+ }
> save.image()
```

Finally, we make the formal OOC instance:

```
> if (!DontRun) LL2GOMFooc1.18 <- new("OOC", ontology = GOMF1.18,
+     OOmap = LL2GOMFooMap.1.18)
> save.image()
```

5 Concept probabilities in LL2GOMF

Concept probabilities are computed on the basis of an OOC. At present this is extremely slow, and the calculation should be refined so that only terms that are actually used in the OOC are tested.

```
> if (!DontRun) LL2GOMFcp.1.18 <- conceptProbs(ooc = LL2GOMFooc1.18,
+     acc = goMFamat.1.18)
> save.image()
```