How To Use GOstats and Category to do Hypergeometric testing with unsupported model organisms

M. Carlson

May 19, 2021

1 Introduction

This vignette is meant as an extension of what already exists in the GOstatsHyperG.pdf vignette. It is intended to explain how a user can run hypergeometric testing on GO terms or KEGG terms when the organism in question is not supported by an annotation package. The 1st thing for a user to determine then is whether or not their organism is supported by an organism package through AnnotationForge. In order to do that, they need only to call the available.dbschemas() method from AnnotationForge.

```
> library("AnnotationForge")
> available.dbschemas()
```

```
 [1] "ARABIDOPSISCHIP_DB" "BOVINECHIP_DB"      "BOVINE_DB"
 [4] "CANINECHIP_DB"      "CANINE_DB"          "CHICKENCHIP_DB"
 [7] "CHICKEN_DB"         "ECOLICHIP_DB"       "ECOLI_DB"
[10] "FLYCHIP_DB"         "FLY_DB"             "GO_DB"
[13] "HUMANCHIP_DB"       "HUMANCROSSCHIP_DB"  "HUMAN_DB"
[16] "INPARANOIDDROME_DB" "INPARANOIDHOMSA_DB" "INPARANOIDMUSMU_DB"
[19] "INPARANOIDRATNO_DB" "INPARANOIDSACCE_DB" "KEGG_DB"
[22] "MALARIA_DB"         "MOUSECHIP_DB"       "MOUSE_DB"
[25] "PFAM_DB"            "PIGCHIP_DB"         "PIG_DB"
[28] "RATCHIP_DB"         "RAT_DB"             "WORMCHIP_DB"
[31] "WORM_DB"            "YEASTCHIP_DB"       "YEAST_DB"
[34] "ZEBRAFISHCHIP_DB"   "ZEBRAFISH_DB"
```

If the organism that you are using is listed here, then your organism is supported. If not, then you will need to find a source or GO (org KEGG) to gene mappings. One source for GO to gene mappings is the blast2GO project. But you might also find such mappings at Ensembl or NCBI. If your organism is not a typical model organism, then the GO terms you will find are probably going to be predictions based on sequence similarity measures instead of direct measurements. This is something that you might want to bear in mind when you draw conclusions later.

In preparation for our subsequent demonstrations, lets get some data to work with by borrowing from an organism package. We will assume that you will use something like `read.table`

to load your own annotation packages into a data.frame object. The starting object needs to be a data.frame with the GO Id's in the 1st col, the evidence codes in the 2nd column and the gene Id's in the 3rd.

```
> library("org.Hs.eg.db")
> frame = toTable(org.Hs.egGO)
> goframeData = data.frame(frame$go_id, frame$Evidence, frame$gene_id)
> head(goframeData)
```

```
  frame.go_id frame.Evidence frame.gene_id
1  GO:0002576            TAS             1
2  GO:0008150             ND             1
3  GO:0043312            TAS             1
4  GO:0001869            IDA             2
5  GO:0002576            TAS             2
6  GO:0007597            TAS             2
```

1.1 Preparing GO to gene mappings

When using GO mappings, it is important to consider the data structure of the GO ontologies. The terms are organized into a directed acyclic graph. The structure of the graph creates implications about the mappings of less specific terms to genes that are mapped to more specific terms. The Category and GOstats packages normally deal with this kind of complexity by using a special GO2All mapping in the annotation packages. You won't have one of those, so instead you will have to make one. AnnotationDbi provides some simple tools to represent your GO term to gene mappings so that this process will be easy. First you need to put your data into a GOFrame object. Then the simple act of casting this object to a GOAllFrame object will tap into the GO.db package and populate this object with the implicated GO2All mappings for you.

```
> goFrame=GOFrame(goframeData,organism="Homo sapiens")
> goAllFrame=GOAllFrame(goFrame)
```

In an effort to standardize the way that we pass this kind of custom information around, we have chosen to support geneSetCollection objects from GSEABase package. You can generate one of these objects in the following way:

```
> library("GSEABase")
> gsc <- GeneSetCollection(goAllFrame, setType = GOCollection())
```

1.2 Setting up the parameter object

Now we can make a parameter object for GOstats by using a special constructor function. For the sake of demonstration, I will just use all the EGs as the universe and grab some arbitrarily to be the geneIds tested. For your case, you need to make sure that the gene IDs you use are unique and that they are the same type for both the universe, the geneIds and the IDs that are part of your geneSetCollection.

```
> library("GOstats")
> universe = Lkeys(org.Hs.egGO)
> genes = universe[1:500]
> params <- GSEAGOHyperGParams(name="My Custom GSEA based annot Params",
+                              geneSetCollection=gsc,
+                              geneIds = genes,
+                              universeGeneIds = universe,
+                              ontology = "MF",
+                              pvalueCutoff = 0.05,
+                              conditional = FALSE,
+                              testDirection = "over")
```

And finally we can call `hyperGTest` in the same way we always have before.

```
> Over <- hyperGTest(params)
> head(summary(Over))
```

```
      GOMFID       Pvalue OddsRatio   ExpCount Count Size
1 GO:0036094 1.736644e-23  3.173684  55.891040   134 2446
2 GO:0003824 1.834033e-22  2.644634 129.102361   224 5650
3 GO:0019829 1.021891e-21 37.372598   1.005399    20   44
4 GO:0042625 3.120695e-21 34.493927   1.051099    20   46
5 GO:0043168 3.375418e-21  3.033294  54.245842   127 2374
6 GO:0042802 9.100814e-21  3.197440  44.009053   111 1926
                                                     Term
1                                   small molecule binding
2                                       catalytic activity
3 ATPase-coupled cation transmembrane transporter activity
4    ATPase-coupled ion transmembrane transporter activity
5                                            anion binding
6                                 identical protein binding
```

1.3 Preparing KEGG to gene mappings

This is much the same as what you did with the GO mappings except for two important simplifications. First of all you will no longer need to track evidence codes, so the object you start with only needs to hold KEGG Ids and gene IDs. Seconly, since KEGG is not a directed graph, there is no need for a KEGG to All mapping. Once again I will borrow some data to use as an example. Notice that we have to put the KEGG Ids in the left hand column of our initial two column data.frame.

```
> frame = toTable(org.Hs.egPATH)
> keggframeData = data.frame(frame$path_id, frame$gene_id)
> head(keggframeData)
```

```
  frame.path_id frame.gene_id
1         04610             2
```

```
2         00232             9
3         00983             9
4         01100             9
5         00232            10
6         00983            10
```

```
> keggFrame=KEGGFrame(keggframeData,organism="Homo sapiens")
```

The rest of the process should be very similar.

```
> gsc <- GeneSetCollection(keggFrame, setType = KEGGCollection())
> universe = Lkeys(org.Hs.egGO)
> genes = universe[1:500]
> kparams <- GSEAKEGGHyperGParams(name="My Custom GSEA based annot Params",
+                                 geneSetCollection=gsc,
+                                 geneIds = genes,
+                                 universeGeneIds = universe,
+                                 pvalueCutoff = 0.05,
+                                 testDirection = "over")
> kOver <- hyperGTest(params)
> head(summary(kOver))
```

```
      GOMFID       Pvalue OddsRatio   ExpCount Count Size
1 GO:0036094 1.736644e-23  3.173684  55.891040   134 2446
2 GO:0003824 1.834033e-22  2.644634 129.102361   224 5650
3 GO:0019829 1.021891e-21 37.372598   1.005399    20   44
4 GO:0042625 3.120695e-21 34.493927   1.051099    20   46
5 GO:0043168 3.375418e-21  3.033294  54.245842   127 2374
6 GO:0042802 9.100814e-21  3.197440  44.009053   111 1926
                                                    Term
1                                 small molecule binding
2                                    catalytic activity
3 ATPase-coupled cation transmembrane transporter activity
4    ATPase-coupled ion transmembrane transporter activity
5                                          anion binding
6                                 identical protein binding
```

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

- R version 4.1.0 (2021-05-18), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_GB`, `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 20.04.2 LTS`

- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.13-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.13-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.54.0, AnnotationForge 1.34.0, Biobase 2.52.0, BiocGenerics 0.38.0, Category 2.58.0, GO.db 3.13.0, GOstats 2.58.0, GSEABase 1.54.0, IRanges 2.26.0, Matrix 1.3-3, S4Vectors 0.30.0, XML 3.99-0.6, annotate 1.70.0, graph 1.70.0, org.Hs.eg.db 3.13.0
- Loaded via a namespace (and not attached): Biostrings 2.60.0, DBI 1.1.1, GenomeInfoDb 1.28.0, GenomeInfoDbData 1.2.6, KEGGREST 1.32.0, R6 2.5.0, RBGL 1.68.0, RCurl 1.98-1.3, RSQLite 2.2.7, Rcpp 1.0.6, Rgraphviz 2.36.0, XVector 0.32.0, bit 4.0.4, bit64 4.0.5, bitops 1.0-7, blob 1.2.1, cachem 1.0.5, compiler 4.1.0, crayon 1.4.1, curl 4.3.1, fastmap 1.1.0, genefilter 1.74.0, grid 4.1.0, httr 1.4.2, lattice 0.20-44, memoise 2.0.0, pkgconfig 2.0.3, png 0.1-7, rlang 0.4.11, rstudioapi 0.13, splines 4.1.0, survival 3.2-11, tools 4.1.0, vctrs 0.3.8, xtable 1.8-4, zlibbioc 1.38.0

```
>
```