Textual Description of annaffy

Colin A. Smith

April 14, 2025

Introduction

`annaffy` is part of the Bioconductor project. It is designed to help interface between Affymetrix analysis results and web-based databases. It provides classes and functions for accessing those resources both interactively and through statically generated HTML pages.

The core functionality of `annaffy` depends on annotation contained in Bioconductor data packages. The data packages are created by the `SQLForge` code inside of another package called `AnnotationDbi`. It gathers annotation data from many diverse sources and makes the information easily processed by R. Preconstructed packages for most Affymetrix chips are available on the Bioconductor web site.

1 Loading Annotation Data

`annaffy` represents each type of annotation data as a different class. Currently implemented classes include:

aafSymbol gene symbol

aafDescription gene description/name

aafFunction gene functional description

aafChromosome genomic chromosome

aafChromLoc location on the chromosome (in base pairs)

aafGenBank GenBank accession number

aafLocusLink LocusLink ids (almost never more than one)

aafCytoband mapped cytoband location

aafUniGene UniGene cluster ids (almost never more than one)

aafPubMed PubMed ids

aafGO Gene Ontology identifiers, names, types, and evidence codes

aafPathway KEGG pathway identifiers and names

For each class, there is a constructor function with the same name. It takes as arguments a vector of Affymetrix probe ids as well as the chip name. The chip name corresponds to the name of the data package that contains the annotation. If the data package for the chip is not already loaded, the constructor will attempt to load it. The constructor returns a list of the corresponding objects populated with annotation data. (`NA` values in the annotation package are mapped to empty objects.)

```
> library("annaffy")
```

(For the purpose of demonstration, we will use the `hgu95av2.db` metadata package and probe ids from the `aafExpr` dataset.)

```
> data(aafExpr)
> probeids <- featureNames(aafExpr)
> symbols <- aafSymbol(probeids, "hgu95av2.db")
> symbols[[54]]
```

```
[1] "ARVCF"
attr(,"class")
[1] "aafSymbol"
```

```
> symbols[55:57]
```

```
An object of class "aafList"
[[1]]
[1] "MRPS14"
attr(,"class")
[1] "aafSymbol"

[[2]]
[1] "TDRD3"
attr(,"class")
[1] "aafSymbol"

[[3]]
character(0)
attr(,"class")
[1] "aafSymbol"
```

All annotation constructors return their results as `aafList` objects, which act like normal lists but have special behavior when used with certain methods. One such method is `getText()`, which returns a simple textual representation of most `annaffy` objects. Note the differing ways `annaffy` handles missing annotation data.

```
> getText(symbols[54:57])

[1] "ARVCF"  "MRPS14" "TDRD3"  ""
```

Other annotation constructors return more complex data structures:

```
> gos <- aafGO(probeids, "hgu95av2.db")
> gos[[3]]

An object of class "aafGO"
[[1]]
An object of class "aafGOItem"
@id   "GO:0005163"
@name "nerve growth factor receptor binding"
@type "Molecular Function"
@evid "IBA"

[[2]]
An object of class "aafGOItem"
@id   "GO:0005515"
@name "protein binding"
@type "Molecular Function"
@evid "IPI"

[[3]]
An object of class "aafGOItem"
@id   "GO:0005576"
@name "extracellular region"
@type "Cellular Component"
@evid "TAS"

[[4]]
An object of class "aafGOItem"
@id   "GO:0005615"
@name "extracellular space"
@type "Cellular Component"
@evid "IBA"
```

```
[[5]]
An object of class "aafGOItem"
@id   "GO:0005737"
@name "cytoplasm"
@type "Cellular Component"
@evid "ISS"

[[6]]
An object of class "aafGOItem"
@id   "GO:0005788"
@name "endoplasmic reticulum lumen"
@type "Cellular Component"
@evid "TAS"

[[7]]
An object of class "aafGOItem"
@id   "GO:0007169"
@name "cell surface receptor protein tyrosine kinase signaling pathway"
@type "Biological Process"
@evid "IBA"

[[8]]
An object of class "aafGOItem"
@id   "GO:0007399"
@name "nervous system development"
@type "Biological Process"
@evid "TAS"

[[9]]
An object of class "aafGOItem"
@id   "GO:0007411"
@name "axon guidance"
@type "Biological Process"
@evid "TAS"

[[10]]
An object of class "aafGOItem"
@id   "GO:0007416"
@name "synapse assembly"
@type "Biological Process"
@evid "IDA"
```

```
[[11]]
An object of class "aafGOItem"
@id   "GO:0008021"
@name "synaptic vesicle"
@type "Cellular Component"
@evid "IBA"

[[12]]
An object of class "aafGOItem"
@id   "GO:0008083"
@name "growth factor activity"
@type "Molecular Function"
@evid "IBA"

[[13]]
An object of class "aafGOItem"
@id   "GO:0010832"
@name "negative regulation of myotube differentiation"
@type "Biological Process"
@evid "ISS"

[[14]]
An object of class "aafGOItem"
@id   "GO:0010976"
@name "positive regulation of neuron projection development"
@type "Biological Process"
@evid "ISS"

[[15]]
An object of class "aafGOItem"
@id   "GO:0021675"
@name "nerve development"
@type "Biological Process"
@evid "IBA"

[[16]]
An object of class "aafGOItem"
@id   "GO:0030424"
@name "axon"
@type "Cellular Component"
@evid "IBA"
```

```
[[17]]
An object of class "aafGOItem"
@id   "GO:0030425"
@name "dendrite"
@type "Cellular Component"
@evid "IBA"

[[18]]
An object of class "aafGOItem"
@id   "GO:0031547"
@name "brain-derived neurotrophic factor receptor signaling pathway"
@type "Biological Process"
@evid "TAS"

[[19]]
An object of class "aafGOItem"
@id   "GO:0031550"
@name "positive regulation of brain-derived neurotrophic factor receptor signaling path
@type "Biological Process"
@evid "TAS"

[[20]]
An object of class "aafGOItem"
@id   "GO:0038180"
@name "nerve growth factor signaling pathway"
@type "Biological Process"
@evid "IBA"

[[21]]
An object of class "aafGOItem"
@id   "GO:0043524"
@name "negative regulation of neuron apoptotic process"
@type "Biological Process"
@evid "IBA"

[[22]]
An object of class "aafGOItem"
@id   "GO:0048471"
@name "perinuclear region of cytoplasm"
@type "Cellular Component"
@evid "ISS"
```

```
[[23]]
An object of class "aafGOItem"
@id   "GO:0048668"
@name "collateral sprouting"
@type "Biological Process"
@evid "IDA"

[[24]]
An object of class "aafGOItem"
@id   "GO:0048672"
@name "positive regulation of collateral sprouting"
@type "Biological Process"
@evid "IDA"

[[25]]
An object of class "aafGOItem"
@id   "GO:0048812"
@name "neuron projection morphogenesis"
@type "Biological Process"
@evid "IBA"

[[26]]
An object of class "aafGOItem"
@id   "GO:0050804"
@name "modulation of chemical synaptic transmission"
@type "Biological Process"
@evid "IBA"

[[27]]
An object of class "aafGOItem"
@id   "GO:0051965"
@name "positive regulation of synapse assembly"
@type "Biological Process"
@evid "IDA"

[[28]]
An object of class "aafGOItem"
@id   "GO:1900122"
@name "positive regulation of receptor binding"
@type "Biological Process"
@evid "IDA"
```

```
[[29]]
An object of class "aafGOItem"
@id   "GO:2000008"
@name "regulation of protein localization to cell surface"
@type "Biological Process"
@evid "TAS"

[[30]]
An object of class "aafGOItem"
@id   "GO:2001234"
@name "negative regulation of apoptotic signaling pathway"
@type "Biological Process"
@evid "ISS"
```

The gene ontology constructor, `aafGO()`, returns `aafLists` of `aafGO` objects, which are in turn lists of `aafGOItem` objects. Within each of those objects, there are four slots: id, name, type, and evidence code. The individual slots can be accessed with the `@` operator.

```
> gos[[3]][[1]]@name
```

```
[1] "nerve growth factor receptor binding"
```

If the reader is not already aware, `R` includes two subsetting operators, which can be the source of some confusion at first. Single brackets (`[]`) always return an object of the same type that they are used to subset. For example, using single brackets to subset an `aafList` will return another `aafList`, even if it only contains one item. On the other hand, double brackets (`[[]]`) return just a single item which is not enclosed in a list. Thus the above statement first picks out the third `aafGO` object, then the first `aafGOItem`, and finally the name slot.

2 Linking to Online Databases

One of the most important features of the `annaffy` package its ability to link to various public online databases. Most of the annotation classes in annaffy have a `getURL()` method which returns single or multiple URLs, depending on the object type.

The simplest annotation class which produces a URL is `aafGenBank`. Because Affymetrix chips are generally based off GenBank, all probes have a corresponding GenBank accession number, even those missing other annotation data. The GenBank database provides information about the expressed sequence that the Affymetrix chip detects. Additionally, it helps break down the functional parts of the sequence and provides information about the authors that initially sequenced the gene fragment. (See this URL here.)

```
> gbs <- aafGenBank(probeids, "hgu95av2.db")
> getURL(gbs[[1]])

[1] "http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=search&db=nucleotide&term=U41068
```

In most distributions of R, you can open URLs in your browser with the `browseURL()` function. Many other types of URLs are also possible. Entrez Gene (formerly LocusLink) is a very useful online database that links to many other data sources not referenced by Bioconductor. One worthy of note is OMIM, which provides relatively concise gene function and mutant phenotype information. (See this URL here.)

```
> lls <- aafLocusLink(probeids, "hgu95av2.db")
> getURL(lls[[2]])

[1] "http://www.ncbi.nlm.nih.gov/sites/entrez?Db=gene&Cmd=DetailsSearch&Term=2322"
```

If you are interested in exploring the area of the genome surrounding a probe, the `aafCytoband` provides a link to NCBI's online genome viewer. It includes adjacent genes and other genomic annotations. (See this URL here.)

```
> bands <- aafCytoband(probeids, "hgu95av2.db")
> getURL(bands[[2]])

[1] "http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?direct=on&idtype=gene&id=2322"
```

For primary literature information about a gene, use the `aafPubMed` class. It will provide a link to a list of abstracts on PubMed which describe the gene of interest. The list abstracts that Bioconductor provides are by no means complete and will sometimes only include the paper in which the gene was cloned. (See this URL here.)

```
> pmids <- aafPubMed(probeids, "hgu95av2.db")
> getURL(pmids[[2]])

[1] "http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?tool=bioconductor&cmd=Retrieve&db=PubMed
```

A number of interesting queries can be done with the gene ontology class. You can display the gene ontology family hierarchy for an entire probe id at once, including multiple GO ids. The usefulness of such a query may be dubious, but it is possible. See this URL here.

```
> getURL(gos[[1]])

[1] "http://amigo.geneontology.org/amigo/term/GO:0001501%0aGO:0001501%0aGO:0005515%0aGO
```

You can also show the family hierarchy for a single GO id. (See this URL here.)

```
> getURL(gos[[1]][[4]])

[1] "http://amigo.geneontology.org/amigo/term/GO:0005576"
```

The last link type of note is that for KEGG Pathway information. Most genes are not annotated with pathway data. However, for those that are, it is possible to retrieve schematics of the biochemical pathways a gene is involved in. (See this URL here. Look for the enzyme in question to be highlighted in red.)

```
> paths <- aafPathway(probeids, "hgu95av2.db")
> getURL(paths[[4]])

[1] "http://www.genome.ad.jp/kegg/pathway/hsa/hsa04640.html"
[2] "http://www.genome.ad.jp/kegg/pathway/hsa/hsa04662.html"
[3] "http://www.genome.ad.jp/kegg/pathway/hsa/hsa05340.html"
```

3 Building HTML Pages

In addition to using `annaffy` interactively through R, it may also be desirable to generate annotated reports summarizing your microarray analysis results. Such a report can be utilized by a scientist collaborator with no knowledge of either R or Bioconductor. Additionally, by having all the annotation and statistical data presented together on one page, connections between and generalizations about the data can be made in a more efficient manner.

The primary intent of the `annaffy` package is to produce such reports in HTML. Additionally, it can easily format the same report as tab-delimited text for import into a table, spreadsheet, or database. It supports nearly all the annotation data available through Bioconductor. Additionally, it has facilities for including and colorizing user data in an informative manner.

The rest of this tutorial will make use of an `ExpressionSet` generated for demonstration purposes. It contains anonymized data from a microarray experiment which used the Affymetrix hgu95av2 chip. There are eight total samples in the set, four control samples and four experimental samples. 250 expression measures were selected at random from the results, and another 250 probe ids were selected at random and assigned to those expression measures. The data therefore has no real biological significance, but can still fully show the capabilities of `annaffy`.

3.1 Limiting the Results

HTML reports generated by `annaffy` can grow to become quite large unless some measures are taken to limit the results. Multi-megabyte web pages are unwieldy and should thus be avoided. Doing a ranked statistical analysis is one way to limit results, and will

be shown here. We will rank the expression measures by putting their two-sample Welch t-statistics in order of decreasing absolute value.

The first step is to load the `multtest` package which will be used for the t-test. (It is also part of the Bioconductor project.)

```
> library(multtest)
```

The `mt.teststat()` function requires a vector that specifies which samples belong to the different observation classes. Using a few R tricks, that vector can be produced directly from the first covariate of `pData`.

```
> class <- as.integer(pData(aafExpr)$covar1) - 1
```

```
[1] 0 0 0 0 1 1 1 1
```

Using the class vector, we calculate the t-statistic for each of the probes. We then generate an index vector which can be used to order the probes themselves in increasing order. As a last step, we produce the vector of ranked probe ids. Latter annotation steps will only use the first 50 of those probes.

```
> teststat <- mt.teststat(exprs(aafExpr), class)
> index <- order(abs(teststat), decreasing = TRUE)
> probeids <- featureNames(aafExpr)[index]
```

3.2 Annotating the Probes

Once there is a list of probes, annotation is quite simple. The only decision that needs to be made is which classes of annotation to include in the table. Including all the annotation classes, which is the default, may not be a good idea. If the table grows too wide, its usefulness may decrease. To see which columns of data can be included, use the `aaf.handler()` function. When called with no arguments, it returns the annotation types `annaffy` can handle.

```
> aaf.handler()
```

```
 [1] "Probe"               "Symbol"              "Description"
 [4] "Chromosome"          "Chromosome Location" "GenBank"
 [7] "Gene"                "Cytoband"            "PubMed"
[10] "Gene Ontology"       "Pathway"
```

To help avoid typing errors, subset the vector instead of retyping each column name.

```
> anncols <- aaf.handler()[c(1:3,8:9,11:13)]
```

```
[1] "Probe"       "Symbol"      "Description" "Cytoband"    "PubMed"
[6] "Pathway"     NA            NA
```

This may be too many columns, but it is possible at a later stage to choose to either not show some of the columns or remove them altogether. Note that by using the `widget=TRUE` option in the next function, it is also possible select data columns with a graphical widget. See Figure 1.

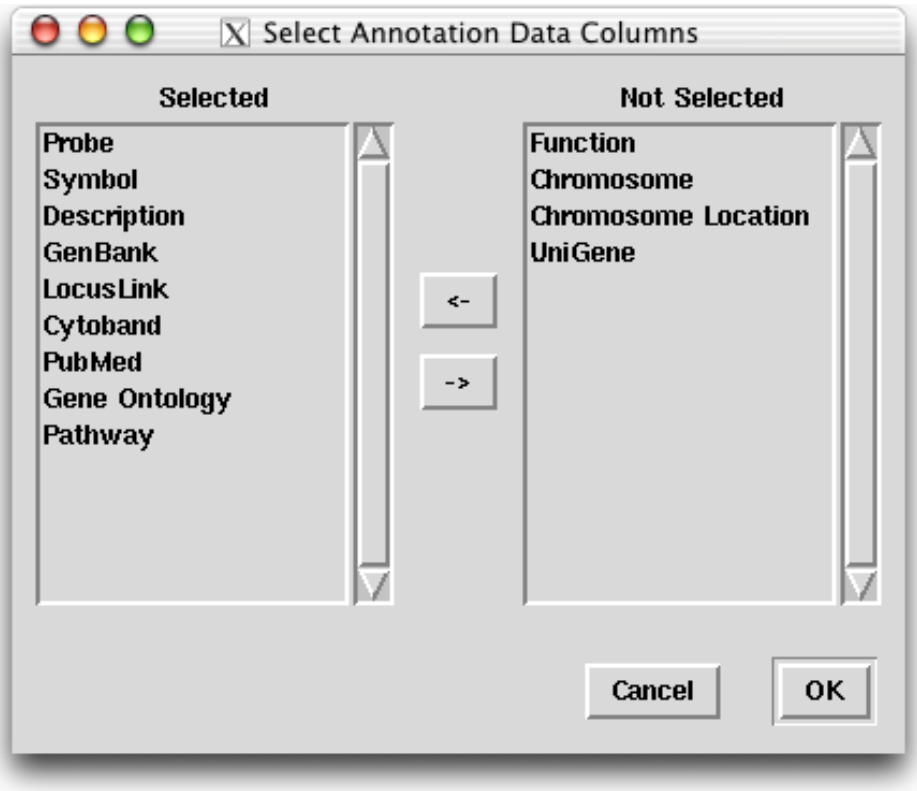

Figure 1: Graphical display for selecting annotation data columns.

Now we generate the annotation table with the `aafTableAnn()` function. Note that for this tutorial, `annaffy` is acting as its own data package. If you wish to annotate results for other chips, download the appropriate data package from the Bioconductor website.

```
> anntable <- aafTableAnn(probeids[1:50], "hgu95av2.db", anncols)
```

To see what has been produced so far, use the `saveHTML()` method to generate the HTML report. Using the optional argument `open=TRUE` will open the resulting file in your browser.

```
> saveHTML(anntable, "example1.html", title = "Example Table without Data")
```

See this page online here.

3.3 Adding Other Data

To add other data to the table, just use any of the other table constructors to generate your own table, and then merge the two. For instance, listing the t-test results along with the annotation data is quite useful. `annaffy` provides the option of colorizing signed data, making it easier to assimilate.

```
> testtable <- aafTable("t-statistic" = teststat[index[1:50]], signed = TRUE)
> table <- merge(anntable, testtable)
```

After HTML generation, a one line change to the style sheet header will change the colors used to show the positive and negative values. In fact, with a bit of CSS it is possible to heavily customize the appearance of the tables very quickly, even on a column by column basis.

`annaffy` also provides an easy way to include expression data in the table. It colorizes the cells with varrying intensities of green to show relative expression values. Additionally, because of the way `merge` works, it will always match probe id rows together, regardless of their order. This allows a quick "sanity check" on the other statistics produced, and can help decrease user error. (Check, for example, that the t-statistics and ranking seem reasonable given the expression data.)

```
> exprtable <- aafTableInt(aafExpr, probeids = probeids[1:50])
> table <- merge(table, exprtable)
> saveHTML(table, "example2.html", title = "Example Table with Data")
```

See this page online here.

Producing a tab-delimited text version uses the `saveText()` method. The text ouput also includes more digits of precision than HTML.

```
> saveText(table, "example2.txt")
```

4 Searching Metadata

Often a biologist will make hypotheses about changes in gene expression either before or after the microarray experiment. In order to facilitate the formulation and testing of such hypotheses, `annaffy` includes functions to search annotation metadata using various criteria. All search functions return character vectors of Affymetrix probe ids that can be used to subset data and annotation.

4.1 Text

The two currently implemented search functions are simple and easy to use. The first is a text search that matches against the textual representation of biological metadata. Recall that textual representations are extracted using the `getText()` method. For complex annotation structures, the textual representation can include a variety of information, including numeric identifiers and textual descriptions.

For the purposes of demonstration, we will use the hgu95av2.db annotation data package available through Bioconductor.

```
> library(hgu95av2.db)
> probeids <- ls(hgu95av2SYMBOL)
> gos <- aafGO(probeids[1:2], "hgu95av2.db")
> getText(gos)

[1] "GO:0000165: MAPK cascade, GO:0001784: phosphotyrosine residue binding, GO:0004674:
[2] "GO:0001525: angiogenesis, GO:0001570: vasculogenesis, GO:0001701: in utero embryon
```

The textual search is probably best applied to the Symbol, Description, and Pathway metadata types. (A specialized Gene Ontology search will be discussed later.) For instance, to find all the kinases on a chip, simply do a text search of Description for kinases.

```
> kinases <- aafSearchText("hgu95av2.db", "Description", "kinase")
> kinases[1:5]

[1] "1000_at"   "1001_at"   "1007_s_at" "1008_f_at" "1010_at"

> print(length(kinases))

[1] 770
```

One can search multiple metadata types with multiple queries all with a single function call. For instance, to find all genes with "ribosome" or "polymerase" in the Description or Pathway annotation, use the following function call.

```
> probes <- aafSearchText("hgu95av2.db", c("Description", "Pathway"),
+                         c("ribosome", "polymerase"))
> print(length(probes))

[1] 83
```

When doing searches of multiple annotation data types or multiple terms, by default the search returns all probe ids matching any of the search criteria. That can be altered by changing the logical operator from OR to AND using the `logic="AND"` argument. This is useful because `aafSearchText()` does not automatically tokenize a search query as Google and many other search engines do. For example, "DNA polymerase" finds all occurrences of that exact string. To find all probes whose description contains both "DNA" and "polymerase", use the following function call.

```
> probes <- aafSearchText("hgu95av2.db", "Description",
+                         c("DNA", "polymerase"), logic = "AND")
> print(length(probes))

[1] 16
```

Another useful application of the text search is to map a vector of GenBank accession numbers onto a vector of probe ids. This comes in handy if you wish to filter microarray data based on the results of a BLAST job.

```
> gbs <- c("AF035121", "AL021546", "AJ006123", "AL080082", "AI289489")
> aafSearchText("hgu95av2.db", "GenBank", gbs)

[1] "1954_at"    "32573_at"   "32955_at"   "34040_s_at" "35581_at"
[6] "38199_at"
```

Lastly, two points for power users. One, the text search is always case insensitive. Second, individual search terms are treated as Perl compatible regular expressions. This means that you should be cautious of special regular expression characters. See the Perl documentation[1] for further information about how to use regular expressions.

4.2 Gene Ontology

The second type of search available is a Gene Ontology search. It takes a vector of Gene Ontology identifiers and maps them onto a list of probe ids. Gene Ontology is a tree and you can include or exclude descendents with the `descendents` argument. The search also supports the `logic` argument. Because the Bioconductor metadata packages include pre-indexed Gene Ontology mappings, this search is very fast.

The input format for Gene Ontology ids is very flexible. You may use numeric or character vectors, either excluding or including the "GO:" prefix and leading zeros.

```
> aafSearchGO("hgu95av2.db", c("GO:0000002", "GO:0000008"))
```

[1] `http://perldoc.perl.org/perlre.html`

```
 [1] "1287_at"    "41146_at"   "32822_at"   "34988_at"   "38885_at"
 [6] "1665_s_at"  "36879_at"   "33286_at"   "1187_at"    "1188_g_at"
[11] "41099_at"   "41747_s_at" "37181_at"   "39745_at"   "1014_at"
[16] "34314_at"   "38846_at"   "39086_g_at" "1028_at"    "41004_at"
[21] "1939_at"    "1974_s_at"  "31618_at"   "36541_at"   "40781_at"
[26] "39643_at"   "34804_at"
```

```
> aafSearchGO("hgu95av2.db", c("2", "8"))
```

```
 [1] "1287_at"    "41146_at"   "32822_at"   "34988_at"   "38885_at"
 [6] "1665_s_at"  "36879_at"   "33286_at"   "1187_at"    "1188_g_at"
[11] "41099_at"   "41747_s_at" "37181_at"   "39745_at"   "1014_at"
[16] "34314_at"   "38846_at"   "39086_g_at" "1028_at"    "41004_at"
[21] "1939_at"    "1974_s_at"  "31618_at"   "36541_at"   "40781_at"
[26] "39643_at"   "34804_at"
```

```
> aafSearchGO("hgu95av2.db", c(2, 8))
```

```
 [1] "1287_at"    "41146_at"   "32822_at"   "34988_at"   "38885_at"
 [6] "1665_s_at"  "36879_at"   "33286_at"   "1187_at"    "1188_g_at"
[11] "41099_at"   "41747_s_at" "37181_at"   "39745_at"   "1014_at"
[16] "34314_at"   "38846_at"   "39086_g_at" "1028_at"    "41004_at"
[21] "1939_at"    "1974_s_at"  "31618_at"   "36541_at"   "40781_at"
[26] "39643_at"   "34804_at"
```

A good source for finding relevant Gene Ontology identifiers is the AmiGO website[2], operated by the Gene Ontology Consortium.

[2] http://www.godatabase.org/