Visualizing and Distances Using GO

R. Gentleman

June 23, 2016

Introduction

The basic characteristics of the GO (**?**) data are described in **?** and the interested reader is referred there for more details. In this paper we assume that readers are familiar with the basic DAG structure of GO and with the mappings of genes to GO terms that are provide by GOA (**?**). We examine some of the different visualization problems and data analytic problems that can be addressed using these data. To make some of the concepts concrete and to provide extensive, but related examples we will make use of a microarray data set (**?**).

We begin with brief descriptions and reminders of some of the important concepts, as well as of our example data. Once that has been done we will then consider several different bioinformatic tasks that are of interest and show how these tasks can be carried out using software from the Bioconductor Project.

The results reported in this document are based on version 3.2.3 of the *hgu95av2.db* data package and on 3.3.0 of the *GO.db* package. If you have different versions your results will probably be different.

0.1 Graphs

The relationships between different GO terms, within a specific ontology are represented in the form of a directed acyclic graph. The leaves of this graph represent the most specific terms and their are edges from a specific term (child) to all less specific terms (each is a parent). The *induced* GO graph is the graph that obtains from taking a set of GO terms and finding all parents of those terms, and so on until the root node has been obtained. Given any set of genes one may use the mappings provided by GOA (**?**) to find the most specific set of GO terms associated with those genes and hence, from those, to the induced GO graph. Thus we can speak of the induced GO graph either from the perspective of selected genes or from the perspective of selected terms.

We note that there is a form of algebra possible on these induced GO graphs. If G_1 and G_2 are two different GO graphs from the same ontology then we can form their union, $G_1 \cup G_2$ which is the union of the nodes in the two graphs together with the

union of the edge sets. We note that this is a bit different than the usual mathematical definition of the union of two graphs. Other set operations such as intersection and complement are also easy to define and implement and they will play a role in our subsequent discussions.

0.2 Some problems

One should be aware that our state of biological knowledge is constantly changing and that the terms, their relationships and the genes mapped to them are constantly changing and being updated. The meta-data used to prepare this report will likely be out of date in less than one year. Hence, one must try to ensure that the mappings being used are up to date (Bioconductor meta-data packages are built quarterly).

Another practical issue that needs some attention is consideration of the incompleteness of the data. Not all genes (or all transcripts) are assayed and not all genes are well enough understood to accurately annotate them at the existing GO nodes; to say nothing of the issues regarding the evolving nature of GO discussed above. For each problem the data analyst will need to consider what, if any, biases these issues might raise.

In all comparisons and gene list developments reported in the remainder of this paper we restrict attention to those genes which satisfy two conditions first they were included on the chip being used and second they have a GO annotation for the ontology of interest.

0.3 An Example

To demonstrate some of the tools that are included in the *GOstats* package we consider expression data from 79 samples from patients with acute lymphoblastic leukemia (ALL) that were investigated using HGU95AV2 Affymetrix GeneChip arrays (**?**). The data were normalized using quantile normalization and expression estimates were computed using RMA (**?**). We will consider the subset of these patients that have B-cell ALL and within those we will further restrict our interest to comparing three groups. Those with BCR/ABL (this is a translocation between chromosomes 9 and 22), those with ALL1/AF4 (this is a translocation between chromosomes 4 and 11) and those with no detected cytogenetic abnormalities (labeled NEG). There are 37 patients with BCR/ABL, 10 with ALL1/AF4 and 42 from the NEG group.

We will examine genes that are differentially expressed between these different phenotypes and then make use of GO and other meta-data resources to help understand the genes that were selected. The actual method of determining differentially expressed genes is in some sense irrelevant to the subsequent analysis using GO and readers can easily substitute their own favorite methods. We refer readers to **?** for a general discussion of some of the issues involved in gene filtering.

Our approach is fairly simplistic. First, we require that genes be expressed in a reasonable set of the samples. Since our smallest group, those with ALL1/AF4, has only

10 samples we will require that a gene be expressed in at least 7 samples to be deemed interesting. For our purposes we consider a gene to be expressed if its value is greater than 100. We also introduce a second selection criteria that is based on differences between the group means. We require that the difference between the smallest group mean and the largest be larger than 100 units. Applying both of these transformations yielded 771 probes that were differentially expressed.

Next restricting attention to just these 771 probes we use the `mt.maxT` function from the *multtest* package to compute adjusted p-values for F-test comparisons. We found 151 of these to have adjusted p-values less than 0.05. We will deem these to be the differentially expressed genes and will now turn our attention to using GO to gain a better understanding of the associated genes. Before going further though, we note that these 151 probes map to 130 distinct Entrez Gene identifiers and since our GO mappings are based on Entrez Gene we further reduce our attention to these. We kept the first (in index order) of each of the duplicated probes for any analyses or computations that require gene expression data and note that others may want to use different conventions.

1 Visualization

Using the genes selected in Section **??** we can construct the induced GO graph for any one of the three different Ontologies. In Figures **??**, **??** and **??** we present these three different graphs. Of course one would like to add some of the experimental data and information to these plots.

The nodes in these different graphs can be colored according to different criteria. In **?** they considered using color for the nodes. The studied differences in gene expression between heart and liver in mice. Their Figure 2 presents data that were assembled in roughly the following manner. Genes of interest were mapped to terms in the Biological Process arm of GO **?**. They then colored the nodes in this tree red if the associated genes were expressed in heart, green if they were expressed in liver and yellow if they were expressed in both. The figure itself is quite helpful and clearly displays the data informatively.

This is an interesting idea and we amplify it a bit here. In our example we have three groups and we might want to consider coloring the nodes differently. For each node in the induced graph there is one or more genes from our list of interesting genes annotated at that node (due to the way we constructed the graph). We could color the nodes according to the phenotype (ALL1/AF4, BCR/ABL, or NEG) that had the highest mean (if there is more than one gene then we might use the multiplicity of highest means). Or, possibly of more interest would be to categorize the genes (again we could use which phenotype had the highest mean) and render the nodes using pie charts.

The computations are quite straightforward. First we find the maximum mean for each gene. We see that BCR/ABL has the most maxima, followed by the ALL1/AF4 samples and finally the NEGs.

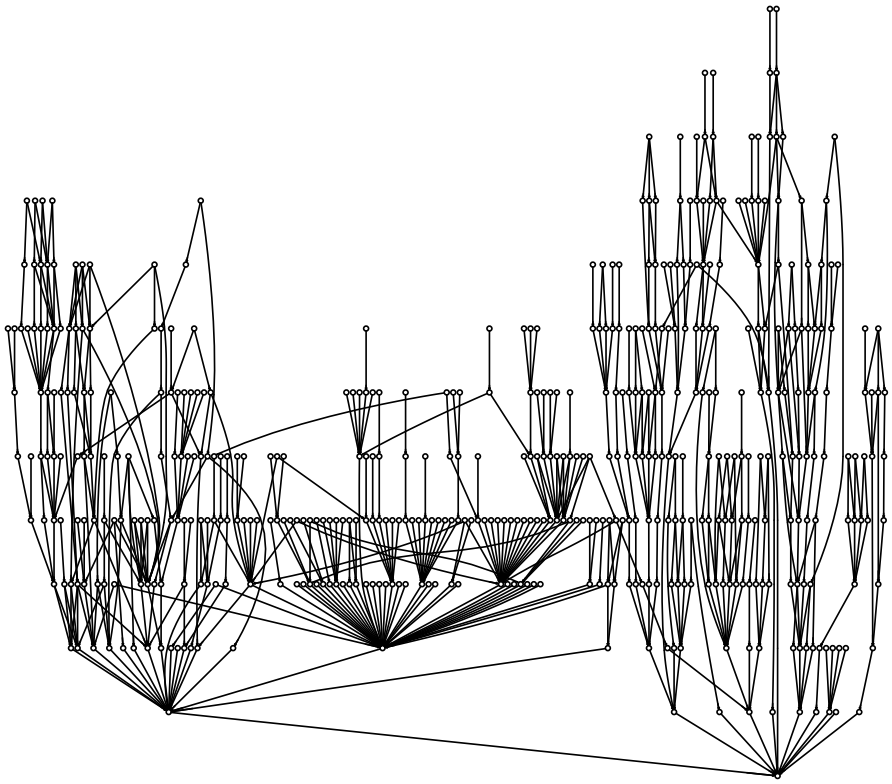

Figure 1: The induced MF GO graph for the selected genes.

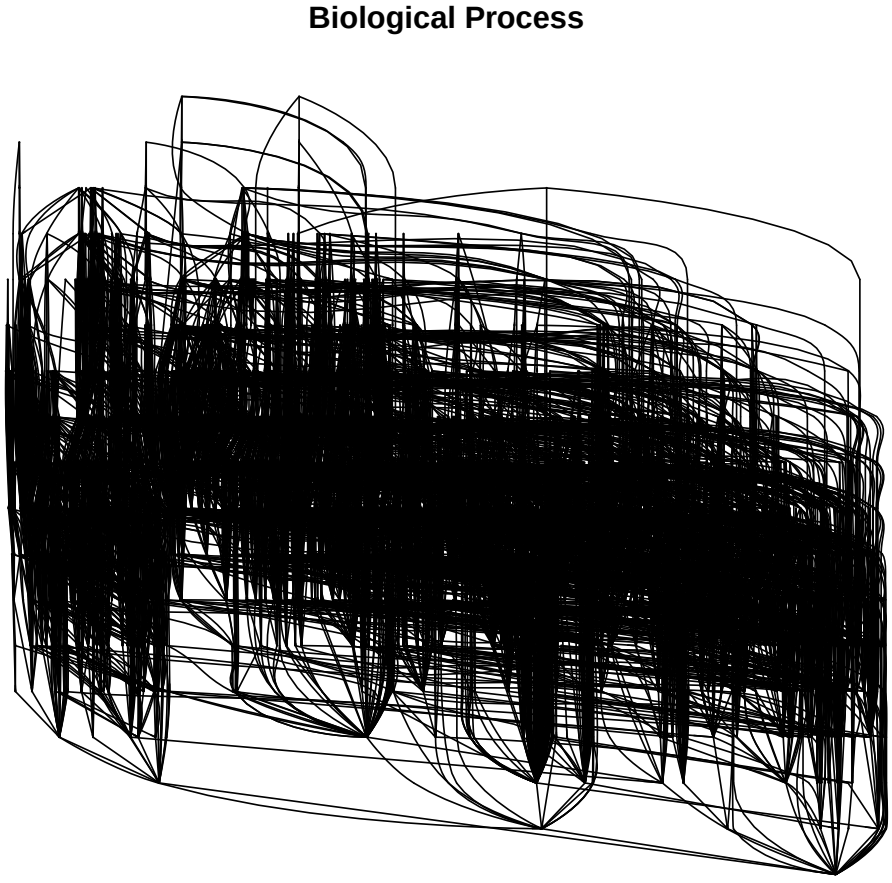

Figure 2: The induced BP GO graph for the selected genes.

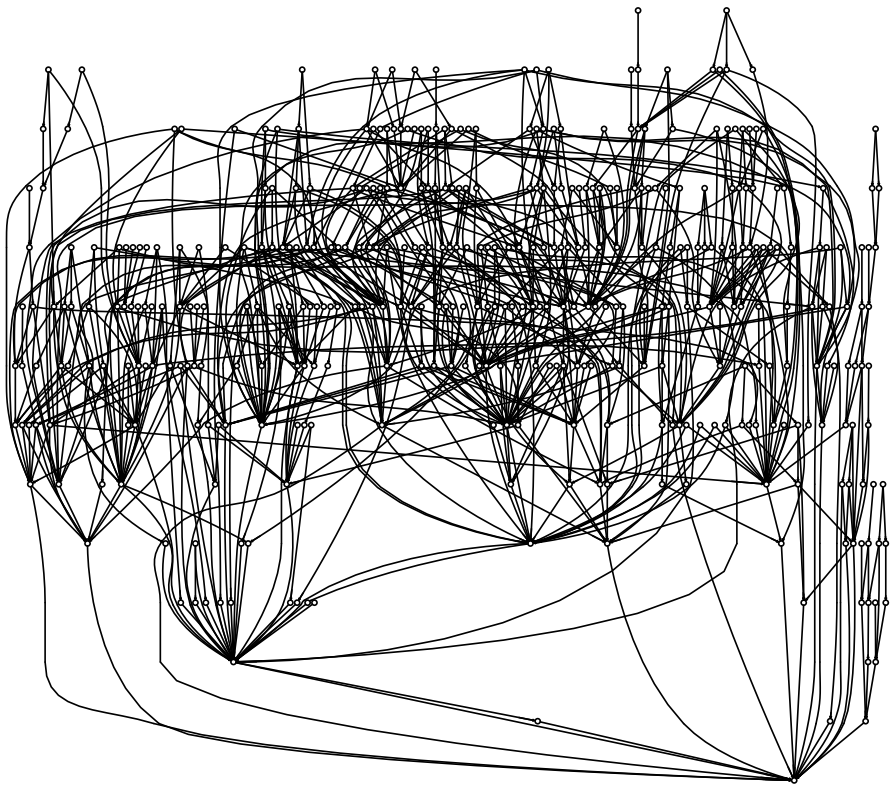

Figure 3: The induced CC GO graph for the selected genes.

```
> mns = apply(exprs(eS), 1, function(x) sapply(split(x, MB), mean))
> whismax = apply(mns, 2, function(x) match(max(x), x))
> maxNames = names(mns[,1])[whismax]
> names(maxNames) = names(whismax)
> table(maxNames)

maxNames
ALL1/AF4  BCR/ABL      NEG
      39       79       13
```

We next need to associate each GO term with the set of these genes and classify them according to which phenotype has the highest mean. Since the CC ontology has the fewest nodes in the induced graph we will use it for the examples here – the reader may want to adapt the code to deal with the other two ontologies.

```
>  CCnodes = nodes(gCC)
>  nodes2affy = mget(CCnodes, hgu95av2GO2ALLPROBES, ifnotfound=NA)
>  cts = sapply(nodes2affy, function(x) {
+        wh = names(whismax) %in% x
+        table(maxNames[wh])
+    })
>  all1cts = sapply(cts, function(x) x["ALL1/AF4"])
>  all1cts = ifelse(is.na(all1cts), 0, all1cts)
>  ##put nice names on
>  names(all1cts) = names(cts)
>  bcrcts = sapply(cts, function(x) x["BCR/ABL"])
>  bcrcts = ifelse(is.na(bcrcts), 0, bcrcts)
>  names(bcrcts) = names(cts)
>  negcts = sapply(cts, function(x) x["NEG"])
>  negcts = ifelse(is.na(negcts), 0, negcts)
>  names(negcts) = names(cts)
>  ctmat = cbind(all1cts, bcrcts, negcts)
```

Now that we have assembled the requisite counts we are now ready to layout and render the graph. The code in the following code chunk does this. We first do a layout to get the node positions and then write a function that will draw each node as a piechart and color the components the requested colors.

```
> if( require(Rgraphviz) ) {
+ opar = par(xpd = NA)
+ plotPieChart <- function(curPlot, counts, main) {
+     renderNode <- function(x) {
+         force(x)
```

```
+         y <- x*100+1
+         function(node, ur, attrs=list(), radConv=1) {
+             nodeCenter <- getNodeCenter(node)
+             pieGlyph(y, xpos=getX(nodeCenter),
+                      ypos=getY(nodeCenter),
+                      radius=getNodeRW(node),
+                      col=c("blue", "green", "red"))
+         }
+     }
+     drawing <- vector(mode="list", length=nrow(counts))
+     for (i in 1:length(drawing)) {
+         drawing[[i]] <- renderNode(counts[i,])
+     }
+     if( missing(main) )
+        main="Example Pie Chart Plot"
+
+     plot(curPlot, drawNode=drawing, main=main)
+
+     legend(x="bottomleft", legend=c("ALL1/AF4", "BCR/ABL", "NEG"),
+            fill=c("blue", "green", "red"))
+ }
+ plotPieChart(gCClo, ctmat)
+ par(opar)
+ }
```

We can see a number of interesting features in this plot. For example, long sequences of single colored nodes - as set of blue nodes on the right and another set of red nodes on the left. We might want to investigate these and see what cellular components they are related to.

We can make use of the `imageMap` function to provide tool tips for our graph.

```
>  getNodeBB = function(ingraph) {
+      xypos = getNodeXY(ingraph)
+      hts = getNodeHeight(ingraph)
+      wdsR = getNodeRW(ingraph)
+      wdsL = getNodeLW(ingraph)
+      llx = xypos$x - wdsL
+      lly = xypos$y - (hts/2)
+      urx = xypos$x + wdsR
+      ury = xypos$y + (hts/2)
+      cbind(llx, lly, urx, ury)
+  }
>  CCbb = getNodeBB(gCClo)
```

```
>  ##now we need to adjust the graphBB to the plot region
>  bbG = boundBox(gCClo)
>  llx = getX(botLeft(bbG))
>  lly = getY(botLeft(bbG))
>  urx = getX(upRight(bbG))
>  ury = getY(upRight(bbG))
> ##FIXME: work in progress here - we need to do a bit of mapping
> ##to get the right user coords on the jpg file...
> if (interactive()) {
+   jpeg("mygraph.jpg", quality=100, width=urx-llx, height=ury-lly)
+   opar = par(plt=c(0,1,0,1), xpd=NA)
+   plotPieChart(gCClo, ctmat)
+   par(opar)
+  dev.off()
+ }
> ## FIXME (wh 20.1.2005): better to use imageMap function for Ragraph objects
> ## in the package Rgraphviz instead.
> if(require("geneplotter")) {
+   con = openHtmlPage("example", "An Image Map")
+
+   imageMap(CCbb, con, tags=list(
+        TITLE = getNodeNames(gCClo),
+        HREF  = rep("", length(nodes(gCC)))),
+      imgname="mygraph.jpg")
+
+   closeHtmlPage(con)
+ }
>
```

2 GO induced Distances

Another idea that has seen some exploration is the notion of defining distances between two genes in terms of the similarity of their GO annotations. We explore some of the suggestions and provide some guidance for those who would like to explore these ideas.

It seems reasonable to treat the three ontologies separately. Even within an ontology a single gene might be annotated at several GO terms. Some mechanisms for dealing with this issue would be helpful. For each gene we could define the graph to be that which is induced by all GO terms to which the gene is mapped, within the ontology of interest. Then use the between graph distances as already described. One could define distance as some combination of the annotation specific distances. For example, if gene 1 is annotated at 2 terms and gene 2 is annotated at three terms there are 6 distances to be computed and some functional of these 6 distances will be reported.

There are two basic strategies that have been widely used to define distances, or similarities, between genes based on GO annotation. One, is to make use of the graph structure that is associated with each pair of genes. If the two graphs are very similar then the two genes are presumed to also be quite similar. The second general strategy is to use the information content in GO as a basis for assigning similarity. Readers should consult **?** for a general review of the area and for some suggestions. Code contributions that implement some of their methods would be most welcome.

Two methods based on graph similarity have been implemented in *GOstats* they are `simUI` and `simLP`. The first uses the number of nodes that the two graphs have in common divided by the number of nodes in the union of the two node sets. Hence these similarities are bounded between 0 and 1 with genes that are more similar having values near one.

For `simLP` the similarity measure is the depth of the longest shared path from the root node. Two genes that are both quite specific and similar should have long shared paths, while those that have less in common should have relatively short shared paths.

Of course neither is perfect and there are many cases where they seem to fail to uncover similarity and others where they suggest a similarity that does not exist. Further investigation and comparison with other methods is certainly warranted.

To compare different genes with respect to their GO annotation you may use `simLL` which first computes the induced GO graph for each of its two, user supplied, Entrez Gene IDs and then calls either `simLP` or `simUI`. Users can restrict the mappings from Entrez Gene IDs to GO IDs based on evidence codes.

Some further work is needed to correctly address two outstanding issues. First, in some cases an Entrez Gene identifier is mapped to several GO terms and in the current implementation these are merely combined into one large graph. However, there may be some benefits in considering the comparison of each induced graph separately. For example, suppose that we have a gene, G_1 that performs a specific molecular function and we want to know which other genes are reasonably similar to it. We might not care if those other genes also do something else and so in this case we would probably use each mapping separately.

The second issue that needs to be addressed at some time is to include functionality that would allow the user to construct a GO graph based only on *is-a* relationships or only on *partof* relationships.

In the next code chunk we take the first 10 genes selected and construct a set of similarity measures between all pairs of genes with respect to the BP ontology.

```
> igenes = unlist(mget(gN[1:10], hgu95av2ENTREZID))
> igenes = igenes[!is.na(igenes)]
> igenes = as.character(igenes)
> psets = mget(igenes, revmap(hgu95av2ENTREZID))
> psets = sapply(psets, function(x) x[1])
> GOs = mget(psets, hgu95av2GO, ifnotfound=NA)
> gBP = sapply(GOs, getOntology, "BP")
```

```
>  ##how many terms
>
>  sum(sapply(gBP, length))
```

```
[1] 156
```

```
>  ##drop those with no BP annotations
>  hasBP = sapply(gBP, function(x) length(x) > 0 && !is.na(x[1]))
>  gBP = gBP[hasBP]
>  ggs = lapply(igenes[hasBP], makeGOGraph, "BP", chip="hgu95av2.db")
> ## you could also do ggx = lapply(gBP, GOGraph, GOBPPARENTS)
> ## and should get the same thing
>
>  simatM1 = matrix(1, nr=sum(hasBP), nc=sum(hasBP))
>  for(i in 1:sum(hasBP))
+     for( j in 1:sum(hasBP) )
+         if( i== j ) next else
+           simatM1[i,j] = simUI(ggs[[i]], ggs[[j]])
>  library("RBGL")
>  simatM2 = matrix(1, nr=sum(hasBP), nc=sum(hasBP))
>  for(i in 1:sum(hasBP))
+     for( j in 1:sum(hasBP) )
+         if( i== j ) next else {
+           simatM2[i,j] = simLP(ggs[[i]], ggs[[j]])
+       }
>
```

You can plot these two measures against each other and compare them. We hope that the two measures are somewhat positively correlated.

3 Operations on GO Graphs

You should refer to the documentation for the *graph* and *RBGL* packages for details on how to manipulate *graph* instances. Here we show how to identify the leaves of a GO graph.

```
> gCC.leaves = leaves(gCC, "in")
> length(gCC.leaves)
```

```
[1] 124
```

```
> gCC.leaves[1:5]
```

```
[1] "GO:0070062" "GO:0008076" "GO:0030672" "GO:0070032" "GO:0070033"
```