

Gene set enrichment analysis with **topGO**

Adrian Alexa, Jörg Rahnenführer

January 26, 2007

<http://www.mpi-sb.mpg.de/~alexa>

1 Preprocessing

We analyse ALL gene expression data from [Chiaretti, S., *et al.*, 2004]. The dataset consists of 128 microarrays from different patients with ALL. First we load the libraries and the data:

```
> library(topGO)
> library(ALL)
> data(ALL)
```

When the **topGO** package is loaded three new environments **GOBPterm**, **GOMFTerm** and **GOMFTerm** are created and binded to the package environment. These environments are build based on the **GOTERM** environment from package **GO**. They are used for fast recovering of the information specific to each ontology. In order to access all GO groups that belong to a specific ontology, e.g. Biological Process (BP), one can type:

```
> BPterms <- ls(GOBPTerm)
> str(BPterms)

chr [1:11048] "GO:0000001" "GO:0000002" "GO:0000003" ...
```

Next we need to load the annotation data. The chip used for the experiment is HGU95aV2 Affymetrix.

```
> affyLib <- annotation(ALL)
> library(package = affyLib, character.only = TRUE)
```

Usually one needs to remove genes with low expression value and genes which might have very small variability across the samples. Package **genefilter** provides such tools.

```
> library(genefilter)
> f1 <- pOverA(0.25, log2(100))
> f2 <- function(x) (IQR(x) > 0.5)
> ff <- filterfun(f1, f2)
> eset <- ALL[genefilter(ALL, ff), ]
```

2 Creating a topGOdata object

The first step when using the **topGO** package is to create a **topGOdata** object. This object will contain all information necessary for the GO analysis, namely the gene list, the list of interesting genes, the scores of genes (if available) and the part of the GO ontology (the GO graph) which needs to be used in the analysis.

First, we need to define the set of genes that are to be annotated with GO terms. Usually, one starts with all genes present on the array. In our case we start with 2400 genes, genes that were not removed by the filtering.

```
> geneNames <- featureNames(eset)
> length(geneNames)
```

In the next step the user needs to define the list of interesting genes or to compute gene scores that quantify the significance of the genes. The **topGO** package deals with these two cases in a unified way. The only difference is the way the **topGOdata** object is build.

2.1 Predefined list of interesting genes

If the user has some a priori knowledge about a set of interesting genes, he can test the enrichment of GO terms with regard to this list of interesting genes. In this scenario, when only a list of interesting genes is provided, the user is restricted to the use of tests statistics that use only counts of genes.

To exemplify this we randomly select 100 genes and consider them as interesting genes.

```
> myInterestedGenes <- sample(geneNames, 100)
> geneList <- factor(as.integer(geneNames %in% myInterestedGenes))
> names(geneList) <- geneNames
> str(geneList)

Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
- attr(*, "names")= chr [1:2400] "1005_at" "1007_s_at" "1008_f_at" "1009_at" ...
```

The object `geneList` is a named factor that indicates which genes are interesting and which not. It is straightforward to compute such a named vector in the situation where a user has his own predefined list of interesting genes.

Next the `topGOdata` object is build. The user needs to specify the ontology of interest (BP, MF or CC) and an annotation function which maps genes/probe IDs to GO terms. The function `annFun.hgu` contained in the package is such an annotation function. As long as the user is using Affymetrix chips, this function does not need to be modified. In other cases the function can be easily modified to comply with the user's needs.

```
> GOdata <- new("topGOdata", ontology = "MF", allGenes = geneList,
+   annot = annFUN.hgu, affyLib = affyLib)
```

```
Building most specific GOs .....      ( 845 GO terms found. )

Build GO DAG topology .....          ( 1199 GO terms and 1440 relations. )

Annotating nodes .....                ( 2049 genes annotated to the GO terms. )
```

The initialisation of the `GOdata` object can take around one minute, depending on the number of annotated genes and on the chosen ontology (in this example we used MF as the ontology of interest). By typing `GOdata`, the user can see the values of some slots.

```
> GOdata
```

```
----- topGOdata object -----
```

Description:

-

Ontology:

- MF

2400 available genes (all genes from the array):

- symbol: 1005_at 1007_s_at 1008_f_at 1009_at 1020_s_at ...
- 100 significant genes.

2049 feasible genes (genes that can be used in the analysis):

- symbol: 1005_at 1007_s_at 1008_f_at 1009_at 1020_s_at ...
- 85 significant genes.

GO graph:

- a graph with directed edges
- number of nodes = 1199
- number of edges = 1440

```
----- topGOdata object -----
```

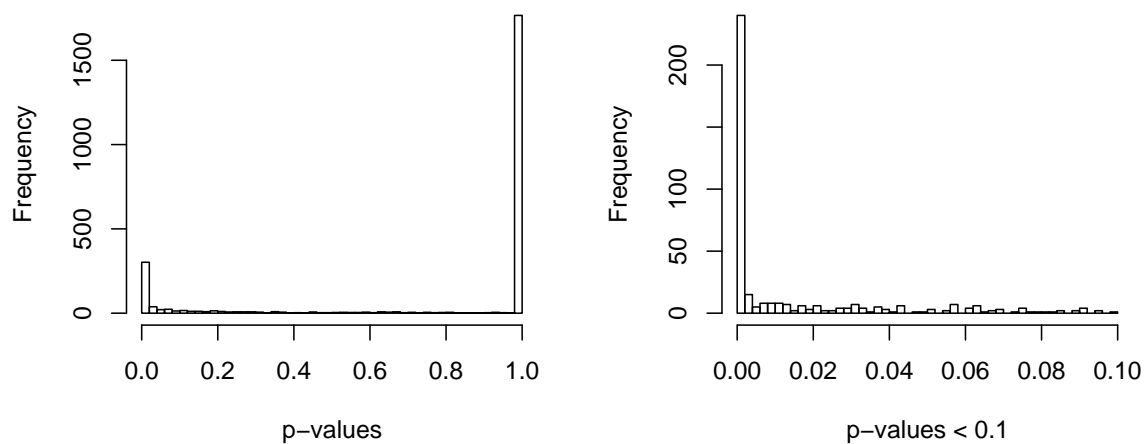


Figure 1: *The distribution of the gene's adjusted p -values.*

One important point here is that not all the genes that are provided by **geneList** can be annotated to the GO. This can be seen by comparing the number of all available genes (the genes present in **geneList**) with the number of feasible genes. It is straight forward to use only the feasible genes for the rest of the analysis, since for other genes no information is available.

The GO graph shows the number of nodes and edges of the specified GO ontology induced by the **geneList**. This graph contains only GO terms with at least one annotated feasible gene.

2.2 Using the genes score

In many cases the set of interesting genes can be computed based on a score assigned to all genes, for example based on the p -value returned by a study of differential expression. In this case, the **topGOdata** object can store the genes score and the rule specifying the list of interesting genes. However, the availability of genes scores allows the user to choose from a larger family of tests statistics to be used in the GO analysis.

A typical example is the study of the ALL dataset where we need to discriminate between ALL cells delivered from either B-cell or T-cell precursors. There are 95 B-cell ALL samples and 33 T-cell ALL samples in the dataset.

```
> y <- as.integer(sapply(eset$BT, function(x) return(substr(x,
+ 1, 1) == "T"))))
> str(y)
```

A two-sided t -test can be applied using the function **getPvalues**. By default the function computes FDR (false discovery rate) adjusted p -value in order to account for multiple testing. A different type of correction can be specified using the **correction** parameter. The distribution of the adjusted p -values is shown in Figure 1.

```
> geneList <- getPvalues(exprs(eset), classlabel = y, alternative = "greater")
> hist(geneList, br = 50)
```

Next, a function for specifying the list of interesting genes must be defined. This function needs to select genes based on their scores (in our case the adjusted p -values) and must return a logical vector specifying which gene is selected and which not. Also, this function must have one parameter, named **allScore** and must not depend on the names attribute of this parameter. For example, if we consider as interesting genes all genes with an adjusted p -value lower than 0.01, the function will look as follows:

```
> topDiffGenes <- function(allScore) {
+   return(allScore < 0.01)
+ }
```

```
> x <- topDiffGenes(geneList)
> sum(x)
```

With all these steps done, the user can now build the `topGOdata` object

```
> GOdata <- new("topGOdata", ontology = "BP", allGenes = geneList,
+   geneSel = topDiffGenes, description = "GO analysis of ALL data based on diff. expression.",
+   annot = annFUN.hgu, affyLib = affyLib)
```

```
Building most specific GOs .....      ( 927 GO terms found. )
Build GO DAG topology .....          ( 1824 GO terms and 2971 relations. )
Annotating nodes .....               ( 1999 genes annotated to the GO terms. )
```

Note that the only difference to the case in which we start with a predefined list of interesting genes is the use of the `geneSel` parameter. All further analysis depends only on this `GOdata` object.

3 Working with the `topGOdata` object

Once the `topGOdata` object is created the user can use various methods defined for this class to access the information encapsulated in the object.

The `description` slot contains information about the experiment. This information can be accessed or replaced using the method with the same name.

```
> description(GOdata)
> description(GOdata) <- paste(description(GOdata), "Object modified on:",
+   format(Sys.time(), "%d %b %Y"), sep = " ")
> description(GOdata)
```

Methods to obtain the list of genes that will be used in the further analysis or methods for obtaining all gene scores are exemplified below.

```
> a <- genes(GOdata)
> str(a)
> numGenes(GOdata)
```

Next we describe how to retrieve the score of a specified set of genes, e.g. a set of randomly selected genes. If the object was constructed using a list of interesting genes, then the factor vector that was provided at the building of the object will be returned.

```
> selGenes <- sample(a, 10)
> gs <- geneScore(GOdata, whichGenes = selGenes)
> print(gs)
```

If the user wants an unnamed vector or the score of all genes:

```
> gs <- geneScore(GOdata, whichGenes = selGenes, use.names = FALSE)
> print(gs)
> gs <- geneScore(GOdata, use.names = FALSE)
> str(gs)
```

The list of significant genes can be accessed using the method `sigGenes()`.

```
> sg <- sigGenes(GOdata)
> str(sg)
> numSigGenes(GOdata)
```

Another useful method is `updateGenes` which allows the user to update/change the list of genes (and their scores) from a `topGOdata` object. If one wants to update the list of genes by including only the feasible ones, one can type:

```
> .geneList <- geneScore(GOdata, use.names = TRUE)
> GOdata
> GOdata <- updateGenes(GOdata, .geneList, topDiffGenes)
> GOdata
```

There are also methods available for accessing information related to GO and its structure. First, we want to know which GO terms are available for analysis and to obtain all the genes annotated to a subset of these GO terms.

```
> graph(GOdata)
```

A graphNEL graph with directed edges

Number of Nodes = 1824

Number of Edges = 2971

```
> ug <- usedGO(GOdata)
> str(ug)
```

```
chr [1:1824] "GO:0000002" "GO:0000003" "GO:0000004" ...
```

Next, we select some random GO terms, count the number of annotated genes and obtain their annotation.

```
> sel.terms <- sample(usedGO(GOdata), 10)
> num.ann.genes <- countGenesInTerm(GOdata, sel.terms)
> num.ann.genes
> ann.genes <- genesInTerm(GOdata, sel.terms)
> str(ann.genes)
```

When the `sel.terms` parameter is missing all GO terms are used. The scores for all genes, possibly annotated with names of the genes, can be obtained using the method `scoresInTerm()`.

```
> ann.score <- scoresInTerm(GOdata, sel.terms)
> str(ann.score)
> ann.score <- scoresInTerm(GOdata, sel.terms, use.names = TRUE)
> str(ann.score)
```

Finally, some statistics for a set of GO terms are returned by the method `termStat`. As mentioned previously, if the `sel.terms` parameter is missing then the statistics for all available GO terms are returned.

```
> termStat(GOdata, sel.terms)
```

	Annotated	Significant	Expected
GO:0031341	2	2	0.23
GO:0051222	3	0	0.34
GO:0009141	26	2	2.95
GO:0006955	205	27	23.28
GO:0006810	367	33	41.68
GO:0045946	2	1	0.23
GO:0030111	5	0	0.57
GO:0015811	1	0	0.11
GO:0006677	1	0	0.11
GO:0031057	1	0	0.11

4 The GO analysis

We are now ready to start the GO analysis. The main function is `getSigGroups()` which takes two parameters. The first parameter is of class `topGOdata` and the second parameter is of class `groupStats`. The `topGO` package is designed to work with different test statistics and with multiple GO graph algorithms, see [Alexa, A., *et al.*, 2006].

There are three algorithms implemented in the package: `classic`, `elim` and `weight`. Also there are two types of test statistics which can be used, test statistics based on gene counts (like Fisher's exact test) and test statistics based on the genes scores (like Kolmogorov-Smirnov test). To distinguish between all the algorithms and to secure that all test statistics are only used with the appropriate algorithms, two classes are defined for each algorithm.

To better understand this principle consider the following example. Assume we decided to apply the `classic` algorithm. The two classes defined for this algorithm are `classicCount` and `classicScore`. If an object of this class is given as a parameter to `getSigGroups()` then the classic algorithm will be used. The `getSigGroups()` function can take a while, depending on the size of the graph (the ontology used), so be patient.

```
> test.stat <- new("classicCount", testStatistic = GOFisherTest,
+   name = "Fisher test")
> resultFis <- getSigGroups(GOdata, test.stat)
```

The algorithm is scoring 768 nontrivial nodes

According to this mechanism, one first defines a test statistic for the chosen algorithm, in this case `classic` and then runs the algorithm (see the second line). The slot `testStatistic` contains the test statistic function. In the above example `GOFisherTest` function which implements Fisher's exact test and is available in the `topGO` package was used. A user can define his own test statistic function and then apply it using the `classic` algorithm. (For example a function which computes the *Z* score can be implemented using as an example the `GOFisherTest` function.)

For the Kolmogorov-Smirnov (KS) test we have:

```
> test.stat <- new("classicScore", testStatistic = GOKSTest,
+   name = "KS tests")
> resultKS <- getSigGroups(GOdata, test.stat)
```

The algorithm is scoring 1824 nontrivial nodes

This time we used the class `classicScore`. This is done since the KS test needs scores of all genes and in this case the *representation* of a group of genes (GO term) is different.

The mechanism presented above for `classic` also hold for `elim` and `weight` with the only remark that for the `weight` algorithm no test based on gene scores is implemented. To run the `elim` algorithm with Fisher's exact test one needs to write:

```
> test.stat <- new("elimCount", testStatistic = GOFisherTest,
+   name = "Fisher test", cutOff = 0.01)
> resultElim <- getSigGroups(GOdata, test.stat)
```

The algorithm is scoring 768 nontrivial nodes

Parameters: cutOff = 0.01

Level 14:	1 nodes to be scored	(0 eliminated genes)
Level 13:	3 nodes to be scored	(0 eliminated genes)
Level 12:	11 nodes to be scored	(0 eliminated genes)
Level 11:	24 nodes to be scored	(4 eliminated genes)
Level 10:	51 nodes to be scored	(6 eliminated genes)

Level 9:	105 nodes to be scored	(28 eliminated genes)
Level 8:	129 nodes to be scored	(33 eliminated genes)
Level 7:	135 nodes to be scored	(44 eliminated genes)
Level 6:	112 nodes to be scored	(46 eliminated genes)
Level 5:	87 nodes to be scored	(46 eliminated genes)
Level 4:	63 nodes to be scored	(58 eliminated genes)
Level 3:	36 nodes to be scored	(58 eliminated genes)
Level 2:	10 nodes to be scored	(58 eliminated genes)
Level 1:	1 nodes to be scored	(58 eliminated genes)

Similarly, for the weight algorithm one types:

```
> test.stat <- new("weightCount", testStatistic = GOFisherTest,
+   name = "Fisher test", sigRatio = "ratio")
> resultWeight <- getSigGroups(GOdata, test.stat)
```

The algorithm is scoring 768 nontrivial nodes

Level 14:	1 nodes to be scored.
Level 13:	3 nodes to be scored.
Level 12:	11 nodes to be scored.
Level 11:	24 nodes to be scored.
Level 10:	51 nodes to be scored.
Level 9:	105 nodes to be scored.
Level 8:	129 nodes to be scored.
Level 7:	135 nodes to be scored.
Level 6:	112 nodes to be scored.
Level 5:	87 nodes to be scored.
Level 4:	63 nodes to be scored.
Level 3:	36 nodes to be scored.
Level 2:	10 nodes to be scored.
Level 1:	1 nodes to be scored.

Next we look at the results of the analysis. First we need to put all resulting p -values into a list. Then we can use the `genTable` function to generate a table with the results.

```
> l <- list(classic = score(resultFis), KS = score(resultKS),
+   elim = score(resultElim), weight = score(resultWeight))
> allRes <- genTable(GOdata, l, orderBy = "weight", ranksOf = "classic",
+   top = 20)
```

	GO.ID	Term	Annotated	Significant	Expected	classicRank	classic	KS	elim	weight
1	GO:0030217	T cell differentiation	8	7	0.91	3	1.6e-06	5.9e-05	1.6e-06	1.6e-06
2	GO:0050870	positive regulation of T cell activation	8	6	0.91	13	4.6e-05	0.00075	0.00513	4.6e-05
3	GO:0007204	elevation of cytosolic calcium ion conce...	6	5	0.68	15	9.9e-05	0.00172	0.21111	9.9e-05
4	GO:0050857	positive regulation of antigen receptor ...	4	4	0.45	20	0.00016	0.00114	0.00016	0.00016
5	GO:0030854	positive regulation of granulocyte diffe...	3	3	0.34	28	0.00145	0.00582	0.00145	0.00145
6	GO:0007200	G-protein signaling, coupled to IP3 seco...	7	4	0.79	37	0.00430	0.04771	0.00430	0.00430
7	GO:0006007	glucose catabolism	20	7	2.27	39	0.00469	0.03394	0.00469	0.00469
8	GO:0042110	T cell activation	14	11	1.59	1	8.6e-09	1.5e-06	0.00484	0.00484
9	GO:0007417	central nervous system development	12	5	1.36	49	0.00731	0.06296	0.00731	0.00731
10	GO:0007586	digestion	5	3	0.57	61	0.01214	0.04612	0.01214	0.01214
11	GO:0008543	fibroblast growth factor receptor signal...	5	3	0.57	62	0.01214	0.08579	0.01214	0.01214
12	GO:0001766	lipid raft polarization	2	2	0.23	65	0.01284	0.03808	0.01284	0.01284
13	GO:0001915	negative regulation of T cell mediated c...	2	2	0.23	66	0.01284	0.03388	0.01284	0.01284
14	GO:0001960	negative regulation of cytokine and chem...	2	2	0.23	67	0.01284	0.03388	0.01284	0.01284
15	GO:0006882	zinc ion homeostasis	2	2	0.23	68	0.01284	0.01846	0.01284	0.01284
16	GO:0007379	segment specification	2	2	0.23	69	0.01284	0.03712	0.01284	0.01284
17	GO:0016579	protein deubiquitination	2	2	0.23	70	0.01284	0.04180	0.01284	0.01284
18	GO:0045059	positive thymic T cell selection	2	2	0.23	71	0.01284	0.01869	0.01284	0.01284
19	GO:0050853	B cell receptor signaling pathway	2	2	0.23	72	0.01284	0.03388	0.01284	0.01284
20	GO:0007265	Ras protein signal transduction	10	4	1.14	92	0.01955	0.01575	0.01955	0.01955

Table 1: Significance of GO terms according to different tests.

`allRes` is a data.frame containing the top 20 GO terms identified by the `weight` algorithm (see `orderBy` parameter). This parameter allows the user decide which p -values should be used for ordering the GO terms. The table includes some statistics on the GO terms plus the p -values obtained from the other algorithms/test statistics. Table 1 shows the results.

We can take a look at the p -values computed by each algorithm, see Figure 2:

```
> par(mfrow = c(2, 2))
> for (nn in names(1)) {
+   p.val <- l[[nn]]
+   hist(p.val[p.val < 1], br = 50, xlab = "p values",
+        main = paste("Histogram for method:", nn, sep = " "))
+ }
```

Another insightful way of looking at the results of the analysis is to investigate how the significant GO terms are distributed over the GO graph. For each algorithm the subgraph induced by the most significant GO terms is plotted. In the plots, the *significant nodes* are represented as boxes. The plotted graph is the upper induced graph generated by these *significant nodes*.

```
> showSigOfNodes(GOdata, score(resultFis), firstTerms = 5,
+   useInfo = "all")
> showSigOfNodes(GOdata, score(resultWeight), firstTerms = 5,
+   useInfo = "def")
```

If we want to print the graphs to .pdf or .ps file, then we can use the following command:

```
> printGraph(GOdata, resultWeight, firstSigNodes = 5, fn.prefix = "tGO",
+   pdfSW = TRUE)
```

```
tGO_weightCount_5_def --- no of nodes: 66
```

To emphasise differences between two methods, one can type:

```
> printGraph(GOdata, resultWeight, firstSigNodes = 10,
+   resultFis, fn.prefix = "tGO", useInfo = "def")
```

```
tGO_weightCount_classicCount_10_def --- no of nodes: 95
```

```
> printGraph(GOdata, resultElim, firstSigNodes = 15, resultFis,
+   fn.prefix = "tGO", useInfo = "all")
```

```
tGO_elimCount_classicCount_15_all --- no of nodes: 119
```

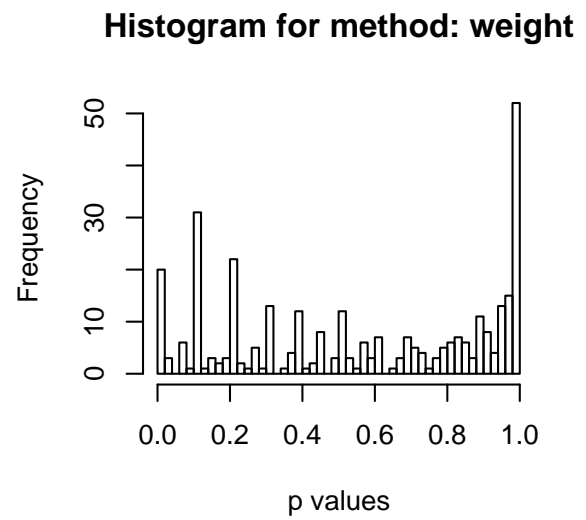
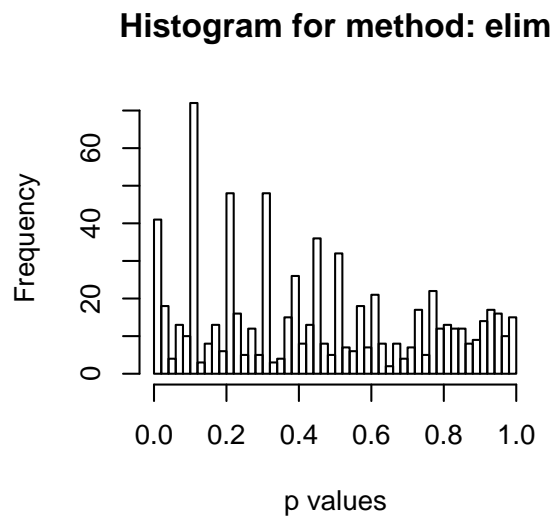
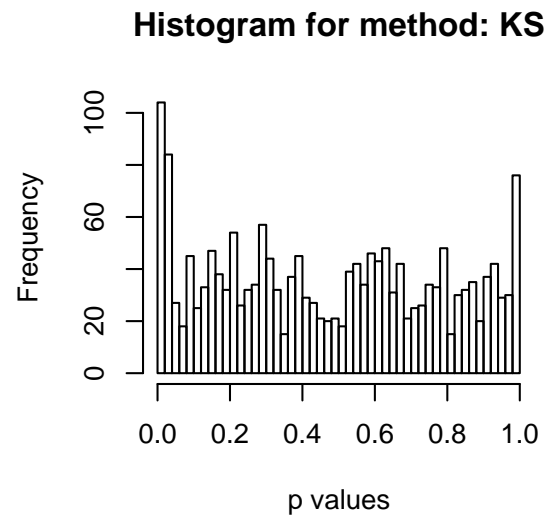
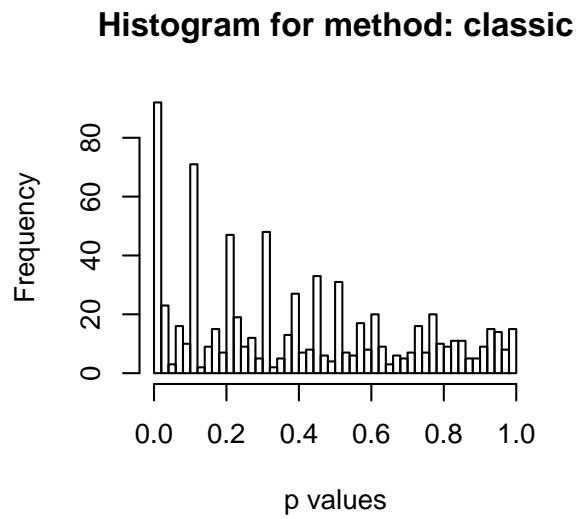



Figure 2: *The distribution of the p -values returned by each method.*

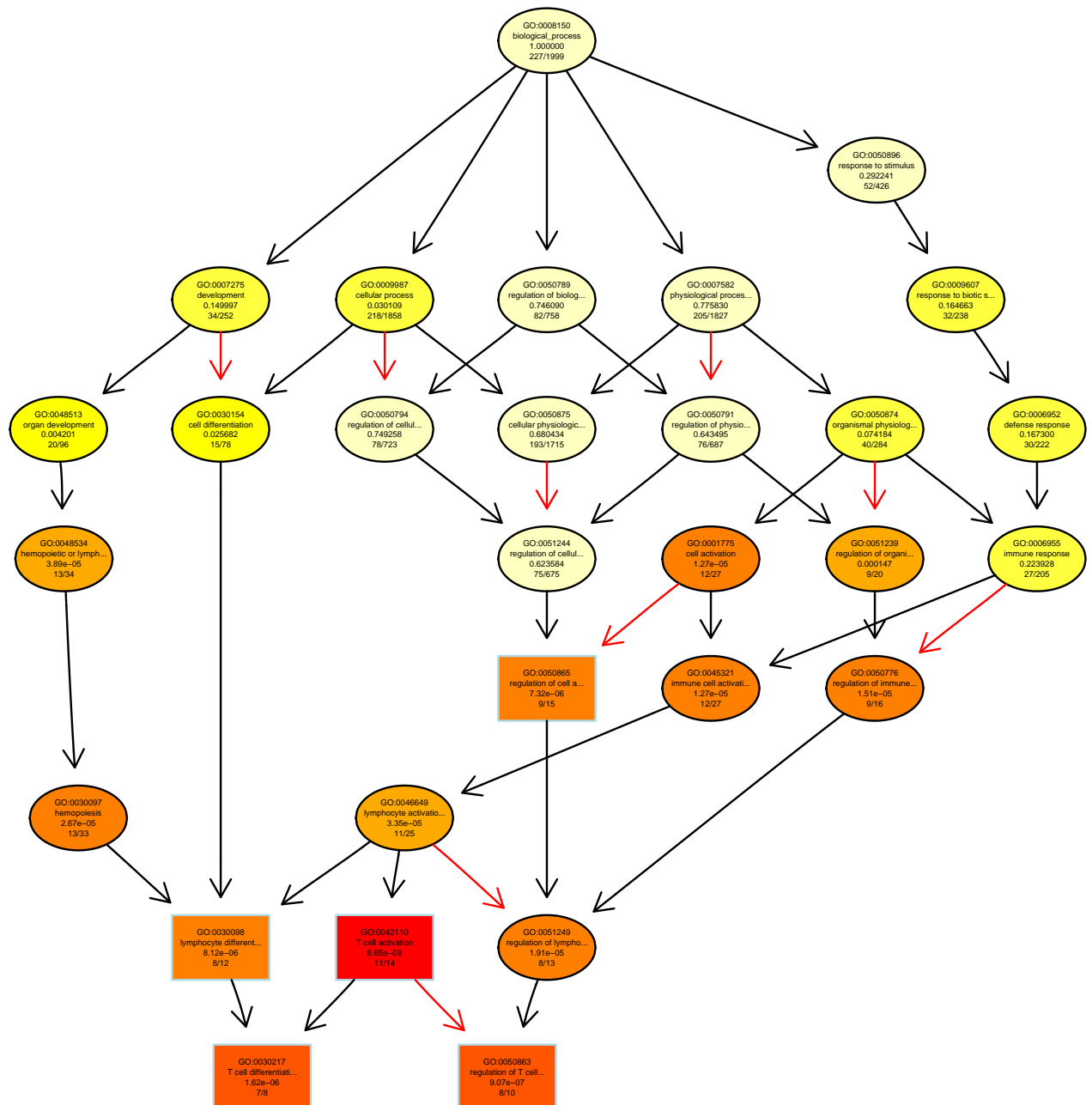


Figure 3: The subgraph induced by the top 5 GO terms identified by the classic algorithm for scoring GO terms for enrichment. Boxes indicate the 5 most significant terms. Box color represents the relative significance, ranging from dark red (most significant) to light yellow (least significant). Black arrows indicate *is-a* relationships and red arrows *part-of* relationships.

References

- [Alexa, A., *et al.*, 2006] Alexa, A., *et al.* (2006). Improved scoring of functional groups from gene expression data by decorrelating go graph structure. *Bioinformatics*, 22(13):1600–1607.
- [Chiaretti, S., *et al.*, 2004] Chiaretti, S., *et al.* (2004). Gene expression profile of adult T-cell acute lymphocytic leukemia identifies distinct subsets of patients with different response to therapy and survival. *Blood*, 103(7):2771–2778.