Gene set enrichment analysis with **topGO**

Adrian Alexa, Jörg Rahnenführer

October 3, 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 [1:13860] "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 .....        ( 897 GO terms found. )

Build GO DAG topology ..........        ( 1284 GO terms and 1531 relations. )

Annotating nodes ...............        ( 2104 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.

 2104 feasible genes (genes that can be used in the analysis):
   - symbol:  1005_at 1007_s_at 1008_f_at 1009_at 1020_s_at  ...
   - 87  significant genes.

 GO graph:
   - a graph with directed edges
   - number of nodes = 1284
   - number of edges = 1531

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

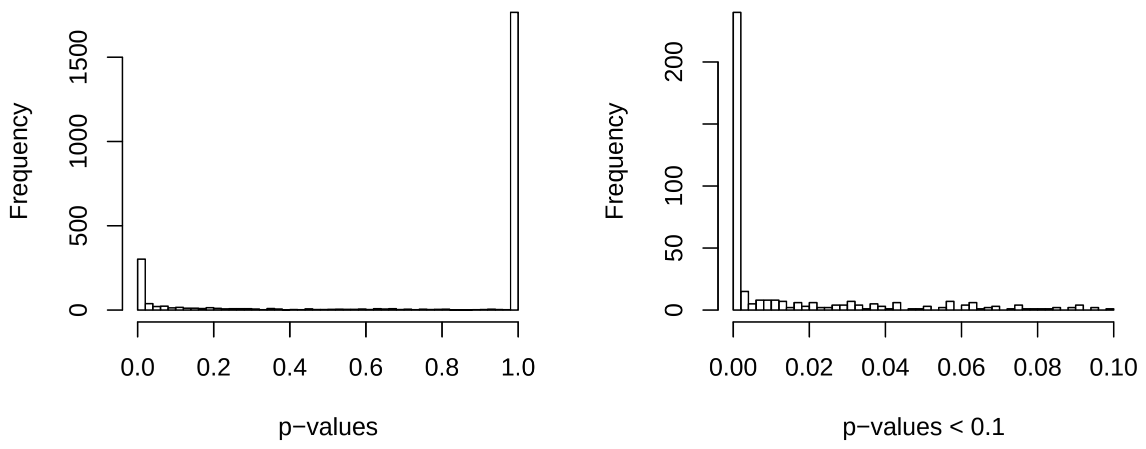

Figure 1: *The distribution of the gene's adjusted p-values.*

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 by 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 .....        ( 1298 GO terms found. )

Build GO DAG topology ..........        ( 2608 GO terms and 4459 relations. )

Annotating nodes ...............        ( 2054 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 = 2608
Number of Edges = 4459
```

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

```
 chr [1:2608] "GO:0000002" "GO:0000003" "GO:0000018" ...
```

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:0006415         3           0     0.35
GO:0019885         1           0     0.12
GO:0001764         3           0     0.35
GO:0009650         1           0     0.12
GO:0030048         3           0     0.35
GO:0044403         5           0     0.59
GO:0045604         2           0     0.23
GO:0006839        10           1     1.17
GO:0006163        37           5     4.34
GO:0009889        59           5     6.92
```

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()` than 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)

                         -- Classic Algorithm --

                 the algorithm is scoring 1046 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)

                         -- Classic Algorithm --

                 the algorithm is scoring 2608 nontrivial nodes
                 parameters:
                         test statistic:  KS tests
                         score order:  increasing
```

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)

                         -- Elim Algorithm --

                 the algorithm is scoring 1046 nontrivial nodes
                 parameters:
                         test statistic:  Fisher test
                         cutOff:  0.01
```

```
         Level 15:        1 nodes to be scored       (0 eliminated genes)

         Level 14:        5 nodes to be scored       (0 eliminated genes)

         Level 13:        12 nodes to be scored       (4 eliminated genes)

         Level 12:        20 nodes to be scored       (4 eliminated genes)

         Level 11:        31 nodes to be scored       (7 eliminated genes)

         Level 10:        55 nodes to be scored       (7 eliminated genes)

         Level 9:        100 nodes to be scored       (19 eliminated genes)

         Level 8:        159 nodes to be scored       (21 eliminated genes)

         Level 7:        178 nodes to be scored       (29 eliminated genes)

         Level 6:        189 nodes to be scored       (29 eliminated genes)

         Level 5:        138 nodes to be scored       (39 eliminated genes)

         Level 4:        90 nodes to be scored       (43 eliminated genes)

         Level 3:        51 nodes to be scored       (43 eliminated genes)

         Level 2:        16 nodes to be scored       (136 eliminated genes)

         Level 1:        1 nodes to be scored       (136 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)

                         -- Weight Algorithm --

                 The algorithm is scoring 1046 nontrivial nodes
                 parameters:
                         test statistic:  Fisher test : ratio

         Level 15:        1 nodes to be scored.

         Level 14:        5 nodes to be scored.

         Level 13:        12 nodes to be scored.

         Level 12:        20 nodes to be scored.

         Level 11:        31 nodes to be scored.

         Level 10:        55 nodes to be scored.

         Level 9:        100 nodes to be scored.

         Level 8:        159 nodes to be scored.

         Level 7:        178 nodes to be scored.

         Level 6:        189 nodes to be scored.
```

	GO.ID	Term	Annotated	Significant	Expected	Rank in classic	classic	KS	elim	weight
1	GO:0050870	positive regulation of T cell activation	13	9	1.53	3	1.7e-06	7.7e-05	0.00079	1.7e-06
2	GO:0050857	positive regulation of antigen receptor-...	4	4	0.47	10	0.00019	0.00120	0.00019	0.00019
3	GO:0051209	release of sequestered calcium ion into ...	4	4	0.47	11	0.00019	0.00120	0.00019	0.00019
4	GO:0030217	T cell differentiation	13	8	1.53	5	2.4e-05	0.00047	2.4e-05	0.00033
5	GO:0030854	positive regulation of granulocyte diffe...	3	3	0.35	25	0.00160	0.00611	0.00160	0.00160
6	GO:0007200	G-protein signaling, coupled to IP3 seco...	6	4	0.70	31	0.00229	0.02386	0.00229	0.00229
7	GO:0030101	natural killer cell activation	4	3	0.47	39	0.00583	0.03393	0.00583	0.00583
8	GO:0046661	male sex differentiation	4	3	0.47	40	0.00583	0.03063	0.00583	0.00583
9	GO:0045621	positive regulation of lymphocyte differ...	5	3	0.59	44	0.01332	0.07415	0.01332	0.01332
10	GO:0006007	glucose catabolic process	18	6	2.11	46	0.01342	0.12446	0.01342	0.01342
11	GO:0001553	luteinization	2	2	0.23	50	0.01372	0.03646	0.01372	0.01372
12	GO:0001766	lipid raft polarization	2	2	0.23	51	0.01372	0.03912	0.01372	0.01372
13	GO:0001915	negative regulation of T cell mediated c...	2	2	0.23	52	0.01372	0.03481	0.01372	0.01372
14	GO:0001960	negative regulation of cytokine and chem...	2	2	0.23	53	0.01372	0.03481	0.01372	0.01372
15	GO:0007379	segment specification	2	2	0.23	54	0.01372	0.03817	0.01372	0.01372
16	GO:0019987	negative regulation of anti-apoptosis	2	2	0.23	55	0.01372	0.02552	0.01372	0.01372
17	GO:0031573	intra-S DNA damage checkpoint	2	2	0.23	56	0.01372	0.03724	0.01372	0.01372
18	GO:0040018	positive regulation of body size	2	2	0.23	57	0.01372	0.03646	0.01372	0.01372
19	GO:0045059	positive thymic T cell selection	2	2	0.23	58	0.01372	0.01875	0.01372	0.01372
20	GO:0045086	positive regulation of interleukin-2 bio...	2	2	0.23	59	0.01372	0.03646	0.01372	0.01372

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

```
         Level 5:        138 nodes to be scored.

         Level 4:        90 nodes to be scored.

         Level 3:        51 nodes to be scored.

         Level 2:        16 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)
```

`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(l)) {
+     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")
```

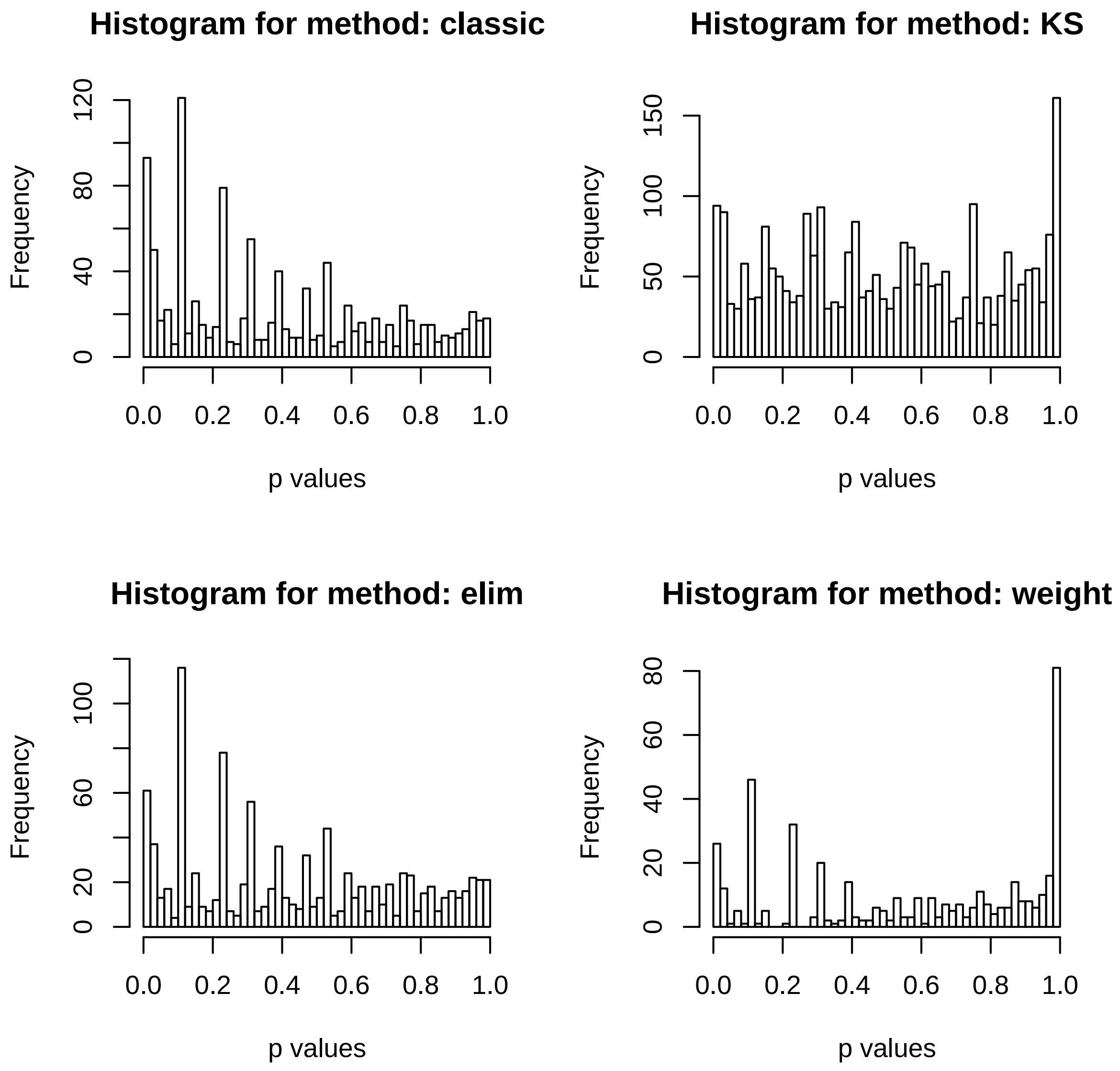

Figure 2: *The distribution of the p-values returned by each method.*

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:  87
```

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:  120

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

tGO_elimCount_classicCount_15_all  --- no of nodes:  134
```

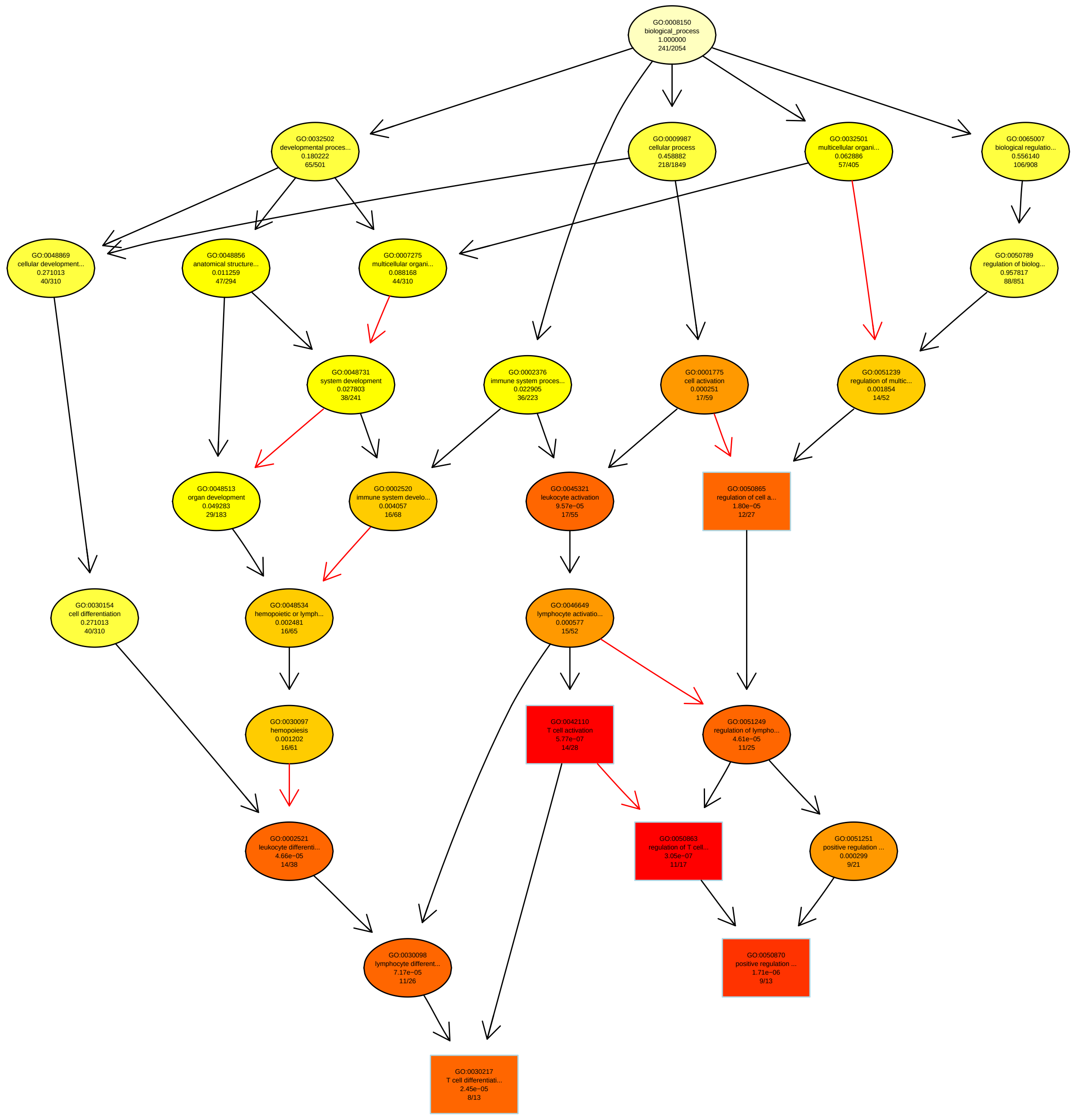

Figure 3: *The subgraph induced by the top 5 GO terms identified by the* classic *algorithm for scoring GO terms for enrichment. Boxes indicate the 5 most significant terms. Box color represents the relative significance, ranging from dark red (most significant) to light yellow (least significant). Black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

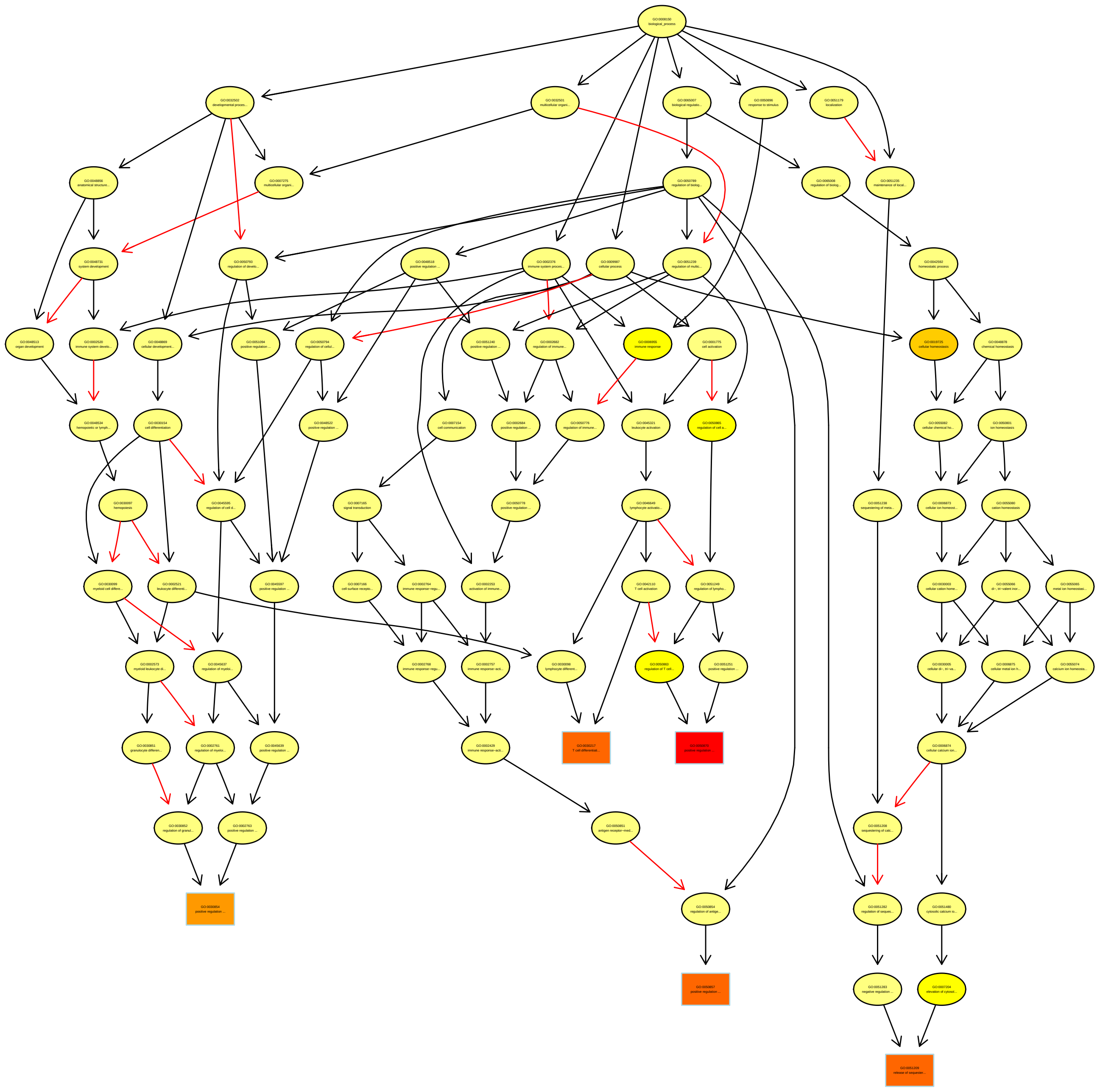

Figure 4: *The subgraph induced by the top 5 GO terms identified by the* weight *algorithm for scoring GO terms for enrichment. Boxes indicate the 5 most significant terms. Box color represents the relative significance, ranging from dark red (most significant) to light yellow (least significant). Black arrows indicate $is\text{-}a$ relationships and red arrows $part\text{-}of$ relationships.*

References

[Alexa, A., *et al.*, 2006] Alexa, A., *et al.* (2006). Improvined scoring of functional groups from gene expression data be 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.