Gene set enrichment analysis with **topGO**

Adrian Alexa, Jörg Rahnenführer

May 3, 2008
http://www.mpi-sb.mpg.de/∼alexa

Contents

1 Introduction

The result of a microarray experiment is a list of genes with corresponding expression profiles. Such a gene list is the starting point for an investigation of the biology manifested by the experimental data. Often, genes are ranked according to differential expression between disease groups or according to correlation of expression values with a phenotype measurement. The result of such an analysis is an ordered list of genes.

Methods that test for enrichment of GO terms have been proposed by [Draghici et al., 2003], [Zeeberg, B. R., *et al.*, 2003] and [Al-Shahrour, F., *et al.*, 2004]. A comparative study of commonly used tools for analyzing GO term enrichment was recently presented by [Khatri and Draghici, 2005]. None of the methods compared in this study integrates the knowledge encapsulated in the hierarchical structure of the GO database. Recently [Grossmann et al., 2006] discussed scoring enrichment of GO terms in a local sense. The over-representation of a GO term is quantified with respect to its direct less specific neighbors in the GO hierarchy. In the present paper we propose algorithms that identify over-represented terms in a more global sense, integrating the whole GO topology in the score.

Several other methods integrating the hierarchical structure of the GO have related but different goals. [Balasubramanian et al test the association between multiple sources of functional genomics data. Each data source is represented by a single graph with nodes representing genes and with edges representing functional links. For a group of genes a graph based on GO is obtained by placing edges between all gene pairs that are annotated to GO terms with a small distance in GO graph topology. The proposed tests statistically compare the occurrence of edges observed in different graphs. [Joslyn et al., 2004] define distances between nodes in the GO graph for ordering GO terms with respect to a group of genes. A GO term is considered more important if many genes in the group are annotated to GO terms close in graph topology. The resulting rank-ordered list of GO terms is then clustered in order to identify summarizing nodes for the characteristics of the gene group.

`topGO` package provides a set of classes and methods which allows the user to run the mentioned algorithms/methods. Moreover the user can use the current framework to develop and test new enrichment algorithms which make use of the GO structure.

This document gives an overview .....

2 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 [1:14598] "GO:0000001" "GO:0000002" "GO:0000003" ...
```

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

```
> affyLib <- paste(annotation(ALL), "db", sep = ".")
> 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), ]
```

3 Sample session

This section provides a quick start in performing an enrichment analysis. Details about the used functions will be presented in the following section.

Lets assume we want to test enrichment of GO terms with differentially expressed genes. The first step is to

3.1 Deciding on the test

get gene

3.2 Setting the annotations

3.3 Running an algorithm

3.4 Looking at the results

4 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.

4.1 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 2 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.db, affyLib = affyLib)

Building most specific GOs .....        ( 925 GO terms found. )

Build GO DAG topology ..........        ( 1330 GO terms and 1602 relations. )

Annotating nodes ...............        ( 2121 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.

 2121 feasible genes (genes that can be used in the analysis):
   - symbol:  1005_at 1007_s_at 1008_f_at 1009_at 1020_s_at  ...
   - 89  significant genes.

 GO graph (nodes with at least  0  genes):
   - a graph with directed edges
   - number of nodes = 1330
   - number of edges = 1602

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

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.

4.2 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.

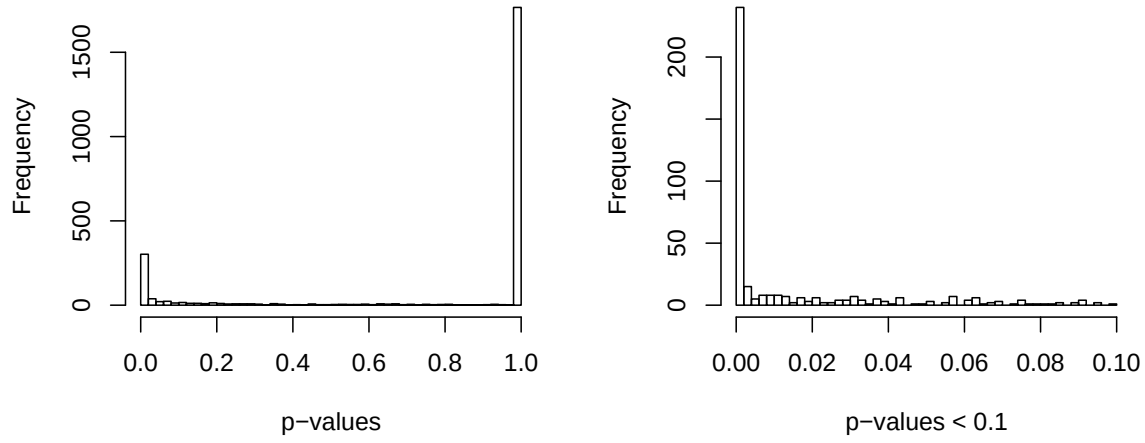

Figure 1: *The distribution of the gene's adjusted p-values.*

```
> 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.db, affyLib = affyLib)
```

```
Building most specific GOs .....        ( 1452 GO terms found. )

Build GO DAG topology ..........        ( 2913 GO terms and 5102 relations. )

Annotating nodes ...............        ( 2071 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.

5 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 = 2913
Number of Edges = 5102
```

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

 chr [1:2913] "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:0033365	1	0	0.12
GO:0045638	6	2	0.71
GO:0019883	2	0	0.24
GO:0035305	1	0	0.12
GO:0048545	6	1	0.71
GO:0007190	4	0	0.47
GO:0051341	4	0	0.47
GO:0000902	46	6	5.42
GO:0001541	2	1	0.24
GO:0009792	16	2	1.89

6 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].

6.1 High-level interface

xxxxxxxxxxxxx

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. b 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 1245 nontrivial nodes
                parameters:
                        test statistic:  Fisher test
```

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 2913 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 1245 nontrivial nodes
                parameters:
                        test statistic:  Fisher test
                        cutOff:  0.01

        Level 15:        2 nodes to be scored        (0 eliminated genes)

        Level 14:        7 nodes to be scored        (0 eliminated genes)

        Level 13:        18 nodes to be scored        (5 eliminated genes)

        Level 12:        27 nodes to be scored        (5 eliminated genes)

        Level 11:        42 nodes to be scored        (8 eliminated genes)

        Level 10:        77 nodes to be scored        (12 eliminated genes)

        Level 9:        132 nodes to be scored        (24 eliminated genes)
```

```
        Level 8:        178 nodes to be scored        (30 eliminated genes)

        Level 7:        195 nodes to be scored        (38 eliminated genes)

        Level 6:        218 nodes to be scored        (39 eliminated genes)

        Level 5:        172 nodes to be scored        (43 eliminated genes)

        Level 4:        106 nodes to be scored        (47 eliminated genes)

        Level 3:        52 nodes to be scored        (97 eliminated genes)

        Level 2:        18 nodes to be scored        (97 eliminated genes)

        Level 1:        1 nodes to be scored        (97 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 1245 nontrivial nodes
                parameters:
                        test statistic:  Fisher test : ratio

        Level 15:        2 nodes to be scored.

        Level 14:        7 nodes to be scored.

        Level 13:        18 nodes to be scored.

        Level 12:        27 nodes to be scored.

        Level 11:        42 nodes to be scored.

        Level 10:        77 nodes to be scored.

        Level 9:        132 nodes to be scored.

        Level 8:        178 nodes to be scored.

        Level 7:        195 nodes to be scored.

        Level 6:        218 nodes to be scored.

        Level 5:        172 nodes to be scored.

        Level 4:        106 nodes to be scored.

        Level 3:        52 nodes to be scored.

        Level 2:        18 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.

	GO.ID	Term	Annotated	Significant	Expected	Rank in classic	classic	KS	elim	weight
1	GO:0050870	positive regulation of T cell activation	15	9	1.77	3	1.0e-05	0.00040	0.00219	1e-05
2	GO:0050857	positive regulation of antigen receptor-...	4	4	0.47	13	0.00019	0.00121	0.00019	0.00019
3	GO:0030217	T cell differentiation	15	9	1.77	4	1.0e-05	0.00047	0.00066	0.00066
4	GO:0051209	release of sequestered calcium ion into ...	5	4	0.59	20	0.00086	0.00591	0.00086	0.00086
5	GO:0030854	positive regulation of granulocyte diffe...	3	3	0.35	27	0.00162	0.00613	0.00162	0.00162
6	GO:0007200	G-protein signaling, coupled to IP3 seco...	7	4	0.82	40	0.00493	0.05030	0.00493	0.00493
7	GO:0050863	regulation of T cell activation	19	12	2.24	2	1.3e-07	2.0e-05	0.00541	0.00541
8	GO:0030101	natural killer cell activation	4	3	0.47	42	0.00591	0.03415	0.00591	0.00591
9	GO:0040014	regulation of multicellular organism gro...	4	3	0.47	43	0.00591	0.02109	0.00591	0.00591
10	GO:0046661	male sex differentiation	4	3	0.47	44	0.00591	0.03071	0.00591	0.00591
11	GO:0007565	female pregnancy	12	5	1.41	53	0.00858	0.07702	0.00858	0.00858
12	GO:0045619	regulation of lymphocyte differentiation	9	5	1.06	30	0.00184	0.02383	0.00184	0.01324
13	GO:0008585	female gonad development	5	3	0.59	64	0.01348	0.07432	0.01348	0.01348
14	GO:0022602	ovulation cycle process	5	3	0.59	65	0.01348	0.07432	0.01348	0.01348
15	GO:0019218	regulation of steroid metabolic process	4	3	0.47	45	0.00591	0.00802	0.00591	0.01374
16	GO:0001766	membrane raft polarization	2	2	0.24	71	0.01383	0.03930	0.01383	0.01383
17	GO:0001915	negative regulation of T cell mediated c...	2	2	0.24	72	0.01383	0.03488	0.01383	0.01383
18	GO:0001960	negative regulation of cytokine and chem...	2	2	0.24	73	0.01383	0.03488	0.01383	0.01383
19	GO:0002378	immunoglobulin biosynthetic process	2	2	0.24	74	0.01383	0.03488	0.01383	0.01383
20	GO:0007379	segment specification	2	2	0.24	75	0.01383	0.03835	0.01383	0.01383

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

```
> allRes <- GenTable(GOdata, classic = resultFis, KS = resultKS,
+     elim = resultElim, weight = resultWeight, orderBy = "weight",
+     ranksOf = "classic", topNodes = 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.

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:  94
```

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:  109

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

tGO_elimCount_classicCount_15_all  --- no of nodes:  141
```

7 Session Information

The version number of R and packages loaded for generating the vignette were:

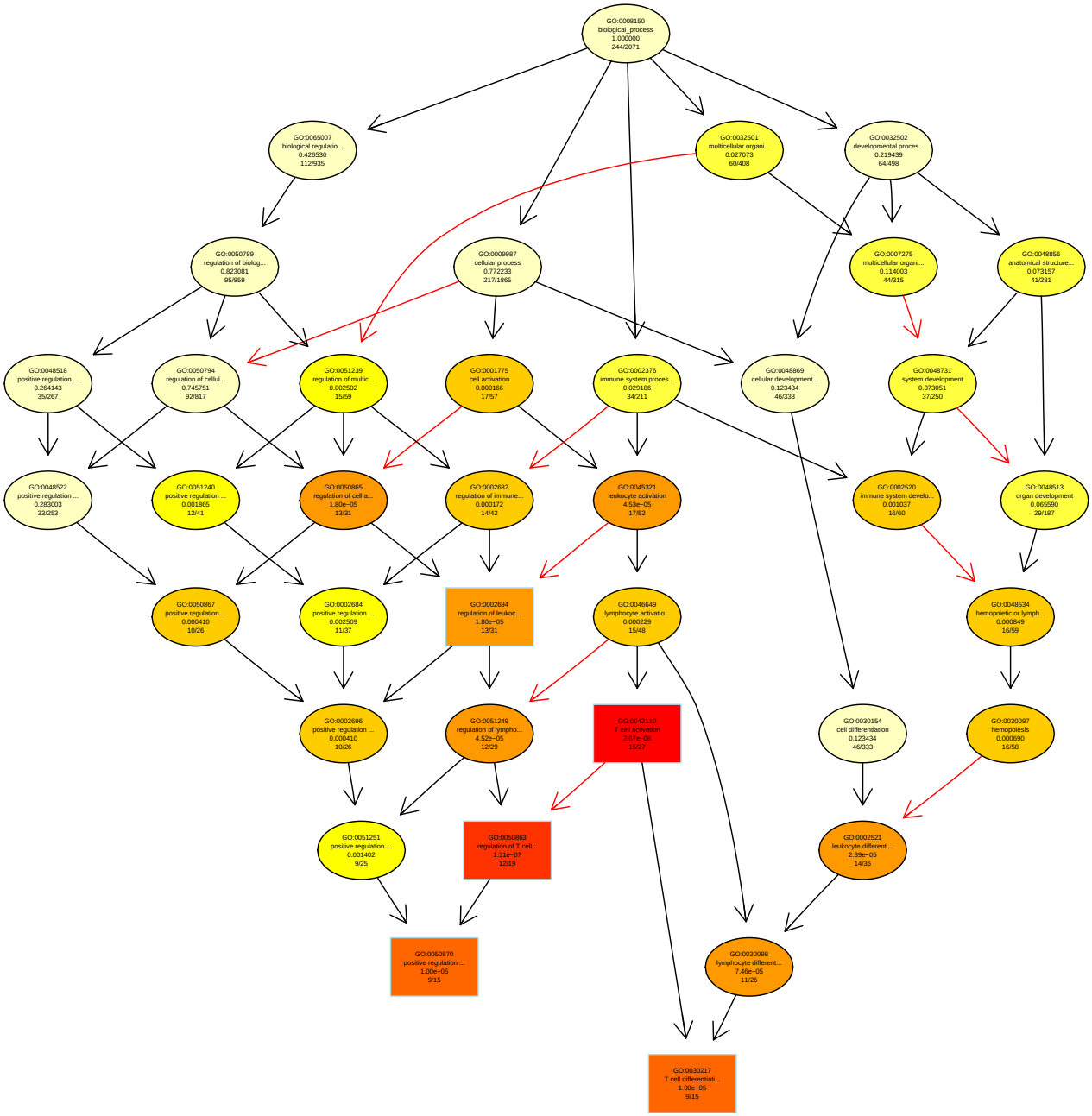

Figure 2: *The subgraph induced by the top 5 GO terms identified by the* classic *algorithm for scoring GO terms for enrichment. Boxes indicate the 5 most significant terms. Box color represents the relative significance, ranging from dark red (most significant) to light yellow (least significant). Black arrows indicate* is-a *relationships and red arrows* $part$-of *relationships.*

- R version 2.7.0 (2008-04-22), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=en_US;LC_NUMERIC=C;LC_TIME=en_US;LC_COLLATE=en_US;LC_MONETARY=C;LC_MESSAGES=en_US;L`
- Base packages: base, datasets, graphics, grDevices, grid, methods, splines, stats, tools, utils
- Other packages: ALL 1.4.3, AnnotationDbi 1.2.0, Biobase 2.0.0, DBI 0.2-4, genefilter 1.20.0, GO.db 2.2.0, graph 1.18.0, hgu95av2.db 2.2.0, multtest 1.20.0, Rgraphviz 1.18.0, RSQLite 0.6-8, SparseM 0.77, survival 2.34-1, topGO 1.8.1, xtable 1.5-2
- Loaded via a namespace (and not attached): annotate 1.18.0, cluster 1.11.10

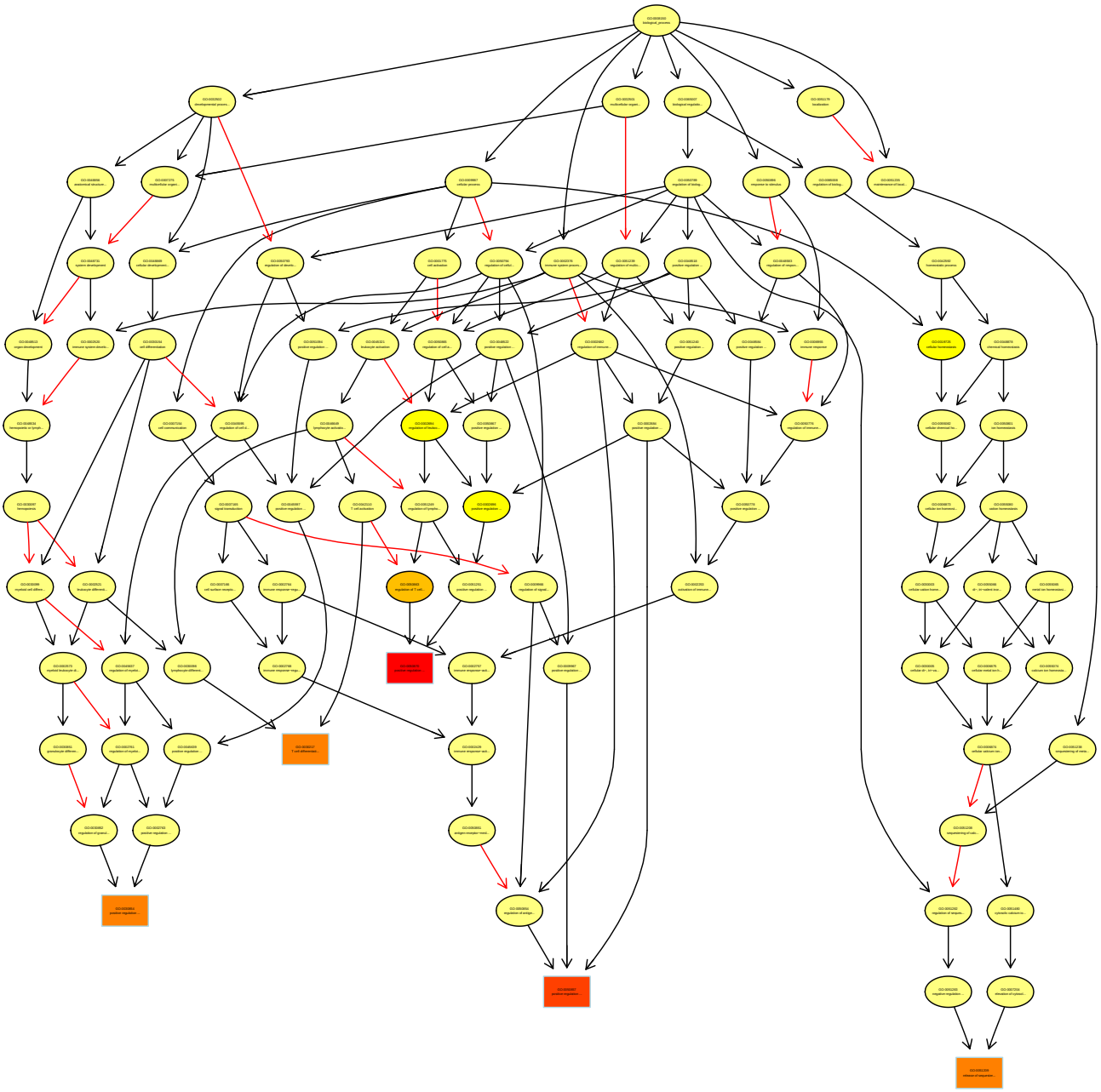

Figure 3: *The subgraph induced by the top 5 GO terms identified by the* weight *algorithm for scoring GO terms for enrichment. Boxes indicate the 5 most significant terms. Box color represents the relative significance, ranging from dark red (most significant) to light yellow (least significant). Black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

References

[Al-Shahrour, F., *et al.*, 2004] Al-Shahrour, F., *et al.* (2004). FatiGO: a web tool for finding significant associations of Gene Ontology terms with groups of genes. *Bioinformatics*, 20:578–580.

[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.

[Balasubramanian et al., 2004] Balasubramanian, R., LaFramboise, T., Scholtens, D., and Gentleman, R. (2004). A graph-theoretic approach to testing associations between disparate sources of functional genomics data. *Bioinformatics*, 20(18):3353–3362.

[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.

[Draghici et al., 2003] Draghici, S., Khatri, P., Martins, R. P., Ostermeier, C., and Krwetz, S. A. (2003). Global functional profiling of gene expression. *Genomics*, 81:98–104.

[Grossmann et al., 2006] Grossmann, S., Bauer, S., Robinson, P. N., and Vingron, M. (2006). An improved statistic for detecting over-represented Gene Ontology annotations in gene sets. In *Proc. 10th Ann. Int. Conf. on Res. in Comput. Biol. (RECOMB '06)*, Venice.

[Joslyn et al., 2004] Joslyn, C. A., Mniszewski, S. M., Fulmer, A., and Heaton, G. (2004). The Gene Ontology Categorizer. *Bioinformatics*, 20(Suppl. 1):i169–i177.

[Khatri and Draghici, 2005] Khatri, P. and Draghici, S. (2005). Ontological analysis of gene expression data: current tools, limitations, and open problems. *Bioinformatics*, 21(18):3587–3595.

[Zeeberg, B. R., *et al.*, 2003] Zeeberg, B. R., *et al.* (2003). GoMiner: a resource for biological interpretation of genomic and proteomic data. *Genome Biology*, 4(4):R28.