Integrating the GO graph structure in scoring the significance of Gene Ontology terms

by

Adrian Alexa

A dissertation submitted in partial satisfaction of the
requirements for the degree of

Master of Science

in

Computer Science

I N F O R M A T I K

Computational Biology and Applied Algorithmics Group
Max Planck Institute for Informatics
Saarbruecken, Germany

Supervised by:

Dr. Jörg Rahnenführer

Prof. Dr. Thomas Lengauer, Ph.D.

March, 2005

Abstract

The analysis of gene expression data is an important tool for understanding mechanisms of living systems. Microarray experiments provide large amounts of data, but it is difficult to understand biological processes or molecular functions from such data on their own.

Clustering methods based on expression data have been used to deal with this problem, but the biological relevance of the results is limited. New methods have been proposed in which annotations from the Gene Ontology (GO) database are integrated into the analysis in order to gain biological understanding. Gene classes obtained from GO are scored with respect to their significance in datasets from microarray experiments.

Current methods of this type do not incorporate the structure of the GO database when computing statistical quantities. In this thesis we developed methods that make use of this topology in order to improve the biological insight obtained from gene expression.

By ignoring the structure of the GO, the current methods do not account for the strong dependences between the GO terms. Four algorithms with different levels of complexity for decorrelating the GO terms are proposed. The algorithms are systematically evaluated on a real microarray dataset and on simulated data. The results obtained for both datasets show that our methods improve the inference.

Beside method development we also focus on the visualization of the results. More insight on the biological functions dependences can be obtained by inspecting how the significant GO terms are distributed in the GO graph.

I hereby declare that this thesis is entirely my own work except where otherwise indicated. I have used only the resources given in the list of references.

Adrian Alexa
March, 2005

Acknowledgements

First of all, I would like to acknowledge my supervisors Jörg Rahnenführer and Thomas Lengauer.

Jörg had an admirable patience, he never seemed to lose confidence in me, although I was not a perfect student, many times couldn't met my deadlines. We got many times involved into hours of discussions, out of which I always got some new ideas and energy for my work. He shared his experience with me, teaching me some of the research rules.

I am particularly thankful to Prof. Lengauer for introducing me to the field of Bioinformatics and for giving me the chance to make my master thesis in his research group.

I want to thank all members of AG3 group, for creating a very nice working environment. I had fruitful discussions with Tobias Sing, Oliver Sander, and Jochen Maydt on issues related to the R programming language, speeding up the process of my work. Besides working, we also spent some nice time together. Particularly, Andreas Steffan is guilty for showing me how wonderful the world of Jazz music is. A big thanks to all of you guys that helped me recover from my accident. Thanks for being there and supporting me through those difficult moments.

I would also like to thank the International Max Planck Research School for offering me a fellowship for my master studies. A special thanks to Kerstin Meyer Ross, the IMPRS coordinator, for her support.

Last but not least I would like to thank Georgiana Ifrim for helping me in writing down the thesis and for her continuous support.

Contents

List of Figures

List of Tables

List of Algorithms

The formalists are like a watchmaker who is absorbed in making his watches look pretty that he has forgotten their purpose of telling the time, and has therefore omitted to insert any works.

“Principles of Mathematics”
Bertrand Russell

Chapter 1

Introduction

Microarray technology gives us the possibility of measuring the abundance of mRNA expression for thousands of genes simultaneously. The challenge is to understand the biological processes or molecular functions from the large amount of data generated by these high-throughput experiments.

1.1 Problem statement

The result of a microarray experiment is a long list of genes together with the expression profiles that show the gene activity under particular conditions or at specific time points. This data is the starting point for an investigation of the underlying biology.

In typical studies the genes are analyzed one by one. One such study is to divide the samples into two groups like disease and healthy, and then to rank the genes according to the differential expression. When the expression profiles measure the activity of the gene at specific time points, genes can be sorted according to the correlation of the expression values with a phenotype measurement. For both studies the result is an ordered list of genes. The task for the researcher is to infer the underlying biology from the ordered list of genes.

In many cases the ordered list of genes is not sufficient and more biological knowledge needs to be included for a rigorous analysis. Grouping the genes based on their regulatory regions was one of the first steps in this direction. With the development of biological knowledge databases more information is available to augment the gene expression data. Biologically interesting sets of genes, genes that belong to a pathway or genes that are known to have the same biological function can now be compiled. Given the sets of genes representing some biological function the task is to find the relevant biological functions for the underling experiment. Gene set enrichment methods analyze the positions of members of a biological function in the ordered list of genes. The biological function is more important if its members are among the top genes in the ordered list.

New methods have been proposed in which annotations from the Gene Ontology (GO) database are integrated into the analysis. Gene sets obtained from GO are scored independently one of each other using different test statistics.

1.2 Motivation: What we need to improve

Current methods that test the enrichment of GO terms do not fully use the knowledge encapsulated in the hierarchical structure of the GO database. The complicated structure of the GO introduces strong dependences between the GO terms.

The statistical interpretation of the results obtained from such methods is unclear due to these dependences. Even when using multiple testing correction procedures the results are biased by the correlation between the GO terms. We believe that by de-correlating the GO terms the interpretation of the results can be improved and new insights can be obtained. Analyzing the GO graph structures, the local dependences between the GO terms can be identified and removed. In this thesis we discuss some strategies to achieve this goal.

Before going further we reformulate the problem in a more general framework. Figure 1.1 presents the problem that we want to solve.

Given:

- a directed acyclic graph (GO) and a set of *items* (genes) s.t.:
  1. each *node* in the graph contains some items
  2. the *parent* of a node contains *all* the items of its child
  3. a node can contain items that are *not found* in the children
- a *subset of items* that we call *significant items* (differentially expressed genes)

Goal:

- find the nodes from the graph (*biological functions*) that *best represent* the significant items w.r.t. some scoring function (*some test statistic*)

Figure 1.1: *Statement of the general problem*

Another important tool in such an analysis is the visualization of the results. By inspecting subgraphs that are in some way significant for the experiment under study, the researcher can gather more insights on the underlying biology. Good visualization tools of the GO graph are needed in this direction.

1.3 Outline

The thesis is structured as follows:

Chapter 2 focuses on the general concepts that we will use in this thesis. We organize it in three sections.

Gene expression data gives the necessary background in this area, emphasizing the main goals of the analysis and discussing what have been achieved so far and where is need for improvement.

Hypothesis testing briefly gives the mathematical background in hypothesis testing theory in the first part. Then, in the second part we demonstrate the importance of this theory for analyzing gene expression data.

Ontologies introduce the reader into the basics of ontologies and their use, especially in biology. A detailed description is given for the Gene Ontology.

Chapter 3 presents some of the current solutions to the problem of scoring the enrichment of members of a gene set. Some remarks are made on the drawbacks of these methods.

Chapter 4 introduces our ideas on how the hierarchical structure of the GO graph should be integrated in scoring gene sets. Four algorithms with different levels of complexity are proposed.

Chapter 5 gives few implementation details necessary for better understanding the experimental results. The **ALL** dataset on which we evaluate our methods is introduced.

We emphasize visualization issues, and the discussion of the results is also based on the visual inspection of the GO graph. For the experimental part we also give the results obtained by running the algorithms with simulated data. We discuss advantages and disadvantages of the algorithms based on the experimental results.

Chapter 6 is summarizing all ideas presented and our contribution in this thesis. Briefly, future directions are discussed.

Chapter 2

Technical basis

In this chapter we introduce some concepts that are relevant for the rest of this thesis. We start by introducing the biology necessary for understanding the content of this thesis, ways to obtain the necessary data using microarray experiments and the methods used for analyzing this data.

Then we give the mathematical background for hypothesis testing showing how and where this mathematical tool is used in gene expression data analysis. Briefly we point out the basis of multiple testing.

In the last section, we briefly introduce the basics of ontologies. Having these basics we describe Gene Ontology - what it is, how it is structured, how genes and gene products can be associated with ontology terms and how we can use this ontology in analyzing gene expression data.

2.1 Gene expression data

To understand the notion of gene expression and the source of the data that we want to analyze, we present some molecular biology basics, following a more extensive, general treatment (Setubal and Meidanis, 1997; Speed, 2003; Tobler, 2002; Dubitzky et al., 2003; Brown and Botstein, 19

2.1.1 DNA, genes and gene expression

DNA. Each cell from an organism contains some very long molecules of DNA (deoxyribonucleic acid) called chromosomes. For us, DNA is a one dimensional molecule composed of two complementary strands that are coiled around each other as a double helix. Each strand can be seen as a string containing only four letters: A, G, C and T (these are the basis that constitute the DNA backbone: **A**denine, **G**uanine, **C**ytosine and **T**hymine). The two strands contain complementary base sequences: A is the complement of T, and G is the complement of C. When two complementary bases (or sequences) are near one another they will form hydrogen bonds.

Genes. A gene is a contiguous stretch of variable length of DNA. Genes are important because they encode the information for synthesizing proteins. Proteins perform a variety of specialized functions in the cell. They are important in terms of biological function and one of the goals in biological science is to understand their structure and function.

Figure 2.1 gives a schematic view of the structure of a gene.

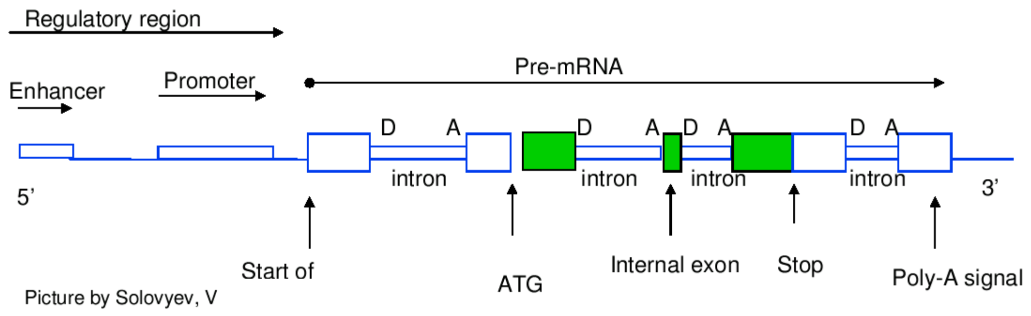


Figure 2.1: *Schematic representation of an eukaryotic gene*

Central dogma of molecular genetics. We are interested is the mechanism that transforms the information stored in a gene into a protein. This transformation is known as the *central dogma of molecular biology* and is shown in Figure 2.2.

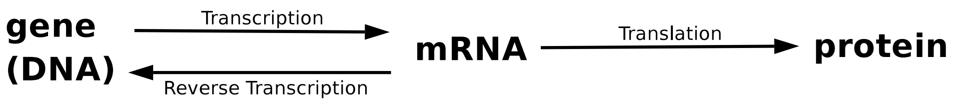


Figure 2.2: *Central dogma of molecular biology*

In short, the information from a particular gene is transferred from a strand of DNA into an intermediate molecule called mRNA (messenger ribonucleic acid) using part of a DNA strand as a template. This process is known as transcription. Thereafter the information contained in the mRNA is used to assemble proteins. This second step is known as translation.

Gene expression. The overall process consisting of transcription and thereafter translation is called gene expression. Thus, we can say that gene expression is the activity of a gene in a cell at a certain time stamp.

Gene expression is important for several reasons. First, knowing which proteins are being produced in a cell can help us distinguish between different tissues. Second, if we can measure which or how many genes are expressed in a cell at different time points, or different stress conditions, then comparing these measurements can tell us if a cell is healthy or not. Finding out the type of tumor of a tissue or diagnosis of a patient are obvious applications for gene expression.

Unfortunately, the current technology does not allow for measuring the real protein abundance, but it is possible to measure the amount of mRNA. It would be helpful if there was a one-to-one mapping from genes to mRNA to protein, but this simplistic view of gene expression does not hold because the process of building proteins is quite complicated. The complication arises mainly due to the gene structure, see Figure 2.1. A gene can be spliced

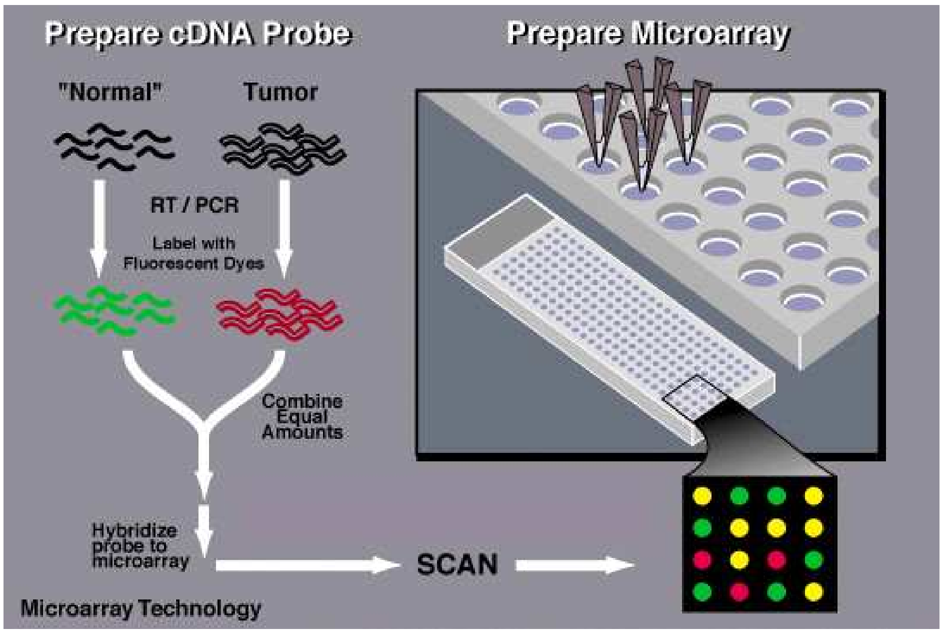


Figure 2.3: *Basic principle of microarray technology*

(the introns are cut and only the exons are translated) during the transcription phase. Also, the choice of the promoter for a gene can result in transcribing different proteins.

Even though there is no high correlation between mRNA abundance and protein abundance, it is believed that measuring the mRNA levels gives us valuable information about the protein levels.

The first step in analyzing gene expression abundance is to measure it. This is the topic of the next section.

2.1.2 Microarrays

Microarray technology gives us the possibility of measuring (quantifying) the abundance of mRNA expression for thousands of genes simultaneously. The most common microarrays types, are cDNA and oligonucleotide arrays. We briefly describe how cDNA microarrays work. The basic phases are shown in Figure 2.3.

Physically, a microarray is a small glass slide. At fixed locations called spots, single-stranded DNA molecules that represent genes of interest are placed. Each such spot measures the relative amount of transcribed mRNA. The procedure is as follows:

- **Probe preparation.** The solution that contains the single-stranded DNA is prepared.
- **Sample preparation.** Two mRNA samples, one reference sample (healthy tissue) and one from the tissue that we want to analyze (cancer tissue), are extracted from the patient. The mRNA is converted into more stable cDNA solution. Then the reference sample is labeled with green-dye reporter and the target sample with red-dye reporter, after which the solutions are mixed.

- **Hybridization.** The sample solution is put on the array. The idea behind this step is that complementary sequences will bind to each other. If in the target sample we have more cDNA for a specific gene, than it is expected that more probe DNA will bind to it. Thus, on the corresponding spot on the array, there will be more green labeled molecules.

- **Washing.** After hybridization, the solution that did not hybridize is removed.

- **Scanning.** The array is scanned with a scanner that generates two images, one for each color. The amount of green and red color is measured and stored for each gene.

After scanning, the chemical process is finished and now the raw data are ready for analysis. There are many issues that should be addressed in a microarray experiment. We will not discuss them here, but refer to (Yang and Speed, 2002; Edgar et al., 2002; Brazma, 2003).

One microarray experiment gives information about the mRNA levels of one patient. If we want to compare the expression levels in different patients, we need to repeat the experiment for each patient. Thus in the end, we will obtain a data matrix containing the raw data for each patient.

Given the current state of technology, microarray experiments are not perfect and the resulting raw data are very noisy. The first step in the analysis of the data is called low-level analysis and its aim is to reduce the noise introduced by the experimental process. This analysis includes image analysis, normalization, missing value handling, feature selection, etc. There are plenty of methods (Quackenbush, 2002; Park, 2003; Huber et al., 2002) that address these issues.

The low-level analysis transforms the raw data matrix into a new matrix, which is called gene expression data (GED) matrix, see Figure 2.4.

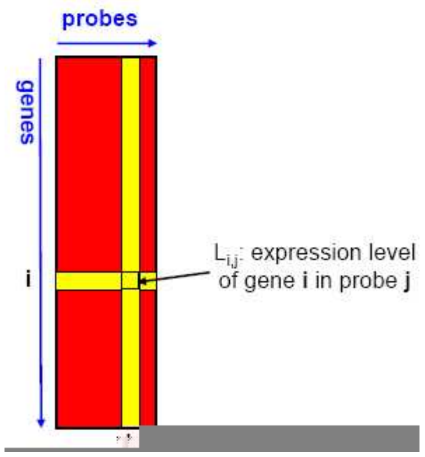


Figure 2.4: *Gene expression data matrix*

Usually the rows of the matrix represent the genes on the array and the columns represent the patients (or different states of a cell). The number of rows is quite large (in the order of 10^5) and the number of columns is small (10 to 100). The shape of the GED matrix influences the complexity of the analysis.

2.1.3 Analysis of GED

There are many biological questions that can be answered using gene expression data. Each question requires a specific type of analysis and the difficulty of the problem varies from one analysis to the other, see (Friedman and Kaminski, 2002; von Heydebreck et al., 2001).

Differentially expressed genes. Finding differentially expressed genes is sometimes considered a last stage in the low-level analysis. This study searches for genes (rows in the matrix) that exhibit different expression levels under different experimental conditions. For example, let us imagine a study in which we have patients with cancer and healthy patients (usually known as disease versus healthy study). We say that a gene is differentially expressed if there is a significant change in expression levels between the two types of patients. In Section 2.2 we will demonstrate how one can determine this difference in expression levels.

If such a study is required, than the output of the analysis will be a list of genes together with the scores giving the amount of differential expression.

Sample classification. One of the first successful studies was to group tumors into classes based on the expression levels (Golub et al., 1999). The classification of tumor types can be used as a diagnostic and therapeutic tool.

An important point to note is that, given the shape of the GED matrix, this problem is not difficult. We have few sample data points (50 to 100) in a very high dimensional space (10^5). Three types of analysis can be seen at this point.

- **Class prediction:** Here the classes are known (the columns of the matrix are labeled with the disease type) and a classifier is trained using this data. This is a typical supervised learning task, and there is a large variety of methods to achieve this (from simple linear discriminant analysis, classification trees to support vector machines and neural networks). After the classifier is trained new patients can be investigated for diagnosis.

- **Feature selection:** This can be thought of as a subtask of class prediction, but the result of such an analysis can be used to find out what genes are significant for discriminating between diseases. Thus the result can be used in feature experiments design, focusing the analysis on the significant genes.

- **Class discovery:** Here the aim is to find meaningful groups in the data, i.e. to recover known types of tumors or to find new ones. This is a typical unsupervised classification task, and as for the supervised case there are several methods that can be used.

For more details on learning methods we refer to (Hastie et al., 2001).

Gene classification. Another type of analysis is to cluster genes with respect to the samples' (patients') profiles. The ultimate goal of this analysis is to infer which biological processes are meaningful for our study.

Unlike in the previous case, here the problem is more difficult. Now we have plenty of sample points (genes), but the space in which they lie is considerably smaller (around 10 or 20 samples). Clustering in such a setup can be meaningless, giving no new biological insight for our study. Figure 2.5 shows the resulting clusters for a microarray study.

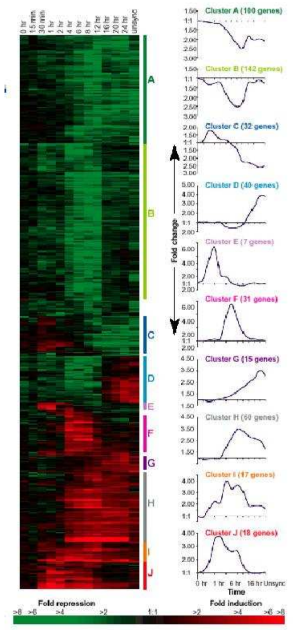


Figure 2.5: *Example of gene clustering.*

The questions that arise naturally after the clusters are obtained are: What meaning do the genes from the same cluster have? Do they have the same biological function?

It turns out that in general genes in the same cluster can have completely different biological meanings. It can happen that two genes have similar expression profiles but are not biologically related at all.

Biclustering. There are other types of analysis that can be carried out with gene expression data. One method that drew the attention of the community is biclustering (Cheng and Church, 2000).

A bicluster is defined as *a subset of genes that exhibit similar behavior across a subset of samples*. Thus biclusters are submatrices of the expression data matrix. Finding biclusters is not a trivial task and different methods were proposed (Tanay et al., 2002), but the problem is still considered open.

Augmenting the GED. Recently new ideas were proposed to address the problems discussed above. Since we are interested in understanding the mechanisms of a disease at the genetic level, more biological knowledge should be included in the study.

One of the first ideas was to analyze the gene expression data together with the regulatory regions of the genes. In this way the genes having similar functions should be clustered together.

The extension of this idea and the development of biological knowledge databases posed new analysis tasks. Biologically interesting groups of genes, genes that belong to a pathway or genes that are known to have the same biological function, are formed. The task is to analyze these gene sets using the gene expression data.

Some methods have been proposed already. For example scoring genes belonging to a pathway is discussed in (Rahnenführer et al., 2004). Sets of genes having the same biological function can be formed using Gene Ontology, see Section 2.3. Scoring these gene sets is discussed in (Draghici et al., 2003; Beissbarth and Speed, 2004; Al-Shahrour et al., 2004; Gentleman, 2004c). These tools will be described in more details in Chapter 3.

In this thesis we will discuss how the scoring of gene sets (GO terms) can be improved by incorporating more biological knowledge.

2.2 Hypothesis testing

Hypothesis testing is a part of statistical decision theory. Its aim is to determine whether there is enough statistical evidence to let us conclude that a hypothesis about a model, is supported by the data. The discussion in this section is based on (Lehmann, 1986; Casella and Berger, 2001; Westfall and Young, 1993).

Definitions. In statistics, a hypothesis is a statement about a population parameter. There are two complementary hypotheses in a testing problem - the *null hypothesis* and the *alternative hypothesis*. They are denoted by H_0 and H_1, respectively.

If our population is modeled by a random variable $\mathbf{X}$, whose distribution $F(X; \theta)$ depends on the parameter θ, then we can write the null and the alternative hypothesis as $H_0 : \theta \in \Theta_0$ and $H_1 : \theta \notin \Theta_0$, where Θ_0 is a subset of the parameter space Θ.

To illustrate this, suppose we want to decide if a treatment affected the heart rate for some patients. If we denote the average heart rate by θ and we know that without treatment the average heart rate equals θ_0, then we will test for $H_0 : \theta = \theta_0$, there is no change in patients heart rate after the treatment, versus $H_1 : \theta \neq \theta_0$, some change in heart rate appeared after the treatment was submitted.

As we stated above the purpose of hypothesis testing is to establish if experimental evidence supports the rejection of the null hypothesis. The experimental evidence is specified in terms of a test statistic $g(\mathbf{X}) = g(X_1, \ldots, X_n)$, where $X_1, \ldots, X_n$ are samples from a random variable $\mathbf{X}$ and g is a multivariate function. An example of a test statistic is $g(\mathbf{X}) = \bar{X}$, the sample mean. Based on the test statistic we decide if we reject H_0. The subset of the sample space for which the test rejects H_0 is called rejection region. The complement of the rejection region is called acceptance region.

Error types. There are two types of errors that appear in hypothesis testing. First, suppose that H_0 is true and the test rejects H_0, then we make a *Type I error* (sometimes referred to as false positive). On the other hand, if H_1 is true and the test statistic decides to accept H_0, then we make a *Type II error* (or false negative). Since we have a background distribution for the sample we can define these errors in terms of probabilities.

Let R be the rejection region for a test statistic. The Type I error is defined as:

$$\alpha = P(\mathbf{X} \in R \mid \theta), \qquad \text{for all } \theta \in \Theta_0.$$

Similarly a Type II error is defined as:

$$P(\mathbf{X} \notin R \mid \theta) = 1 - P(\mathbf{X} \in R \mid \theta), \qquad \text{for all } \theta \in \Theta_0^C.$$

If we define the function $\beta(\theta) = P(\mathbf{X} \in R \mid \theta)$, then we can express both types of errors with this function: If $\theta \in \Theta_0$, then $\beta(\theta)$ is the probability of a Type I error and if $\theta \notin \Theta_0$ then $1 - \beta(\theta)$ is the probability of a Type II error. The function $\beta(\theta)$ is called the power function of the test. The probability of a Type I error is also denoted by α and is called the significance level of the test.

Ideally $\beta(\theta) = 0$ when H_0 is true and $\beta(\theta) = 1$ when H_1 is true. This situation is unlikely to happen in practice and a compromise must be made. An important observation is that by setting a small value (close to 0) for the Type I error does not mean that we also obtain a large value (close to 1) for the Type II error.

In hypothesis testing we must decide what type of error we want to keep low. For example, if we are testing for a disease and we incorrectly decide that the patient has the disease, then the treatment will be useless for the patient. If on the other hand we fail to diagnose the disease of a patient, then the patient can die. Since the null hypothesis is the one that is stated, it is of common practice to control the Type I error probability of the test. This is done by setting the significance level to α. Then, we search for the test that minimize the Type II error. Given α and a test statistic we can define a rejection region. If the sample points lie in the rejection region then we reject H_0 in favor of H_1. For example, if we are interested in the alternative hypothesis, then by setting a small α we accept H_1 only if it is strongly supported by the data.

p-Values. The procedure described in the previous paragraph has the following drawback: We either accept H_0 or we reject H_0 based on the level α. We would like a measure the amount of statistical evidence with which the data is supporting the alternative hypothesis. This measure is given by the p-value of the test.

Mathematically, a p-value $p(\mathbf{X})$ is a test statistic such that small values of $p(\mathbf{X})$ give evidence that H_1 is true. It can be defined as

$$p(\mathbf{x}) = \sup_{H_0} P(g(\mathbf{X}) \geq g(\mathbf{x}) \mid H_0).$$

where $g(\mathbf{X})$ is a test statistic such that a large value of g gives evidence that H_1 is true ($\mathbf{x}$ is the observed sample). In other words, the p-value of a test is the probability of observing a test statistic at least as extreme as the one computed from the sample, given that H_0 is true.

Another way of defining a p-value is to condition on a sufficient statistic. Roughly, a sufficient statistic for a parameter θ is a statistic that captures all the information on θ that is contained in the sample. See (Casella and Berger, 2001) for more details on sufficient statistics. If $S(\mathbf{X})$ is a sufficient statistic for the model $f(\mathbf{x} \mid \theta)$ under H_0 and $g(\mathbf{X})$ is a test statistic such that a large value of g give evidence that H_1 is true, then a valid p-value is

$$p(\mathbf{x}) = P(g(\mathbf{X}) \geq g(\mathbf{x}) \mid S(\mathbf{x}) = s).$$

The p-value of a test provides valuable information since it gives the result of a test on a more continuous scale. If a specific test level α is set than comparing $p(\mathbf{x})$ with α tells if the H_0 has to be rejected. Even more, the smaller the p-value, the more statistical evidence exists to support the alternative hypothesis.

2.2.1 Statistical tests

In this section we briefly describe two test statistic that we will use in the rest of the thesis.

t-test. Let $X_1, \ldots, X_n$ be a random sample from a normal distribution $N(\mu_X, \sigma_X^2)$ with mean μ_X and variance σ_X^2. Let $Y_1, \ldots, Y_m$ be a random sample from a normal distribution $N(\mu_Y, \sigma_Y^2)$ (both samples are independent one of each other). The case of sequential sampling usually arises in applications involving random variables that are observed under different experimental conditions (e.g. microarrays studies). We are interested in testing

$$H_0 : \mu_X = \mu_Y \quad \text{versus} \quad H_1 : \mu_X \neq \mu_Y.$$

Let us further assume that $\sigma_X = \sigma_Y = \sigma$, but unknown (without this assumption the problem is considerably harder). In other words we are interested in testing the hypothesis that the random variables $\mathbf{X}$ and $\mathbf{Y}$ came from the same population.

We define the random variable $\bar{\mathbf{w}} = \bar{\mathbf{X}} - \bar{\mathbf{Y}}$. Under the null hypothesis the mean and the variance can be calculated as

$$\mu_w = 0 \quad \text{and} \quad \sigma_w^2 = \frac{\sigma_X^2}{n} + \frac{\sigma_Y^2}{m}.$$

Thereafter, we define the test statistic

$$\mathbf{T} = \frac{\bar{\mathbf{w}}}{\hat{\sigma_w^2}}.$$

Since we do not know σ we use the pooled variance estimate $\hat{\sigma_w^2}$ of the whole sample. An unbiased estimate for σ^2 is given by

$$S^2 = \frac{1}{n+m-2} \left(\sum_{i=1}^{n} (X_i - \bar{X})^2 + \sum_{i=1}^{m} (Y_i - \bar{Y})^2 \right).$$

Thus we obtain the statistic

$$\mathbf{T} = \frac{\bar{\mathbf{X}} - \bar{\mathbf{Y}}}{\sqrt{S^2 \left(\frac{1}{n} + \frac{1}{m}\right)}}.$$

Under the null hypothesis, $\mathbf{T}$ has a *Student's t*-distribution with $n+m-2$ degrees of freedom. This test is called *two-sample t test.*

To obtain a p-value for this test we must compute

$$p(\mathbf{x}, \mathbf{y}) = 2P\left(t_{n+m-2} \geq \frac{|\bar{X} - \bar{Y}|}{\sqrt{S^2 \left(\frac{1}{n} + \frac{1}{m}\right)}} \right) = 2F_{t,n+m-2}\left(-\frac{|\bar{X} - \bar{Y}|}{\sqrt{S^2 \left(\frac{1}{n} + \frac{1}{m}\right)}} \right)$$

where $F_{t,n+m-2}$ is Student's t the distribution t_{n+m-2}. with $n+m-2$ degrees of freedom.

We use the t-test for determining the differentially expressed genes in a microarray study, see Section 2.1.2. For example, suppose we have two genes for which the expression levels are given in Table 2.1.

There are 15 measurements, 10 from patients that have cancer and 5 from healthy patients. We want to determine which genes are affected by the disease. The assumption here is that the expression level of a gene is normally distributed. If a gene expression is not affected by

Tumor	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N
$gene_1$	1.54	1.68	1.10	0.98	2.01	1.45	1.33	1.67	1.87	1.51	-0.16	-1.13	-0.89	-0.99	-1.10
$gene_2$	0.13	-0.17	0.01	-0.55	0.76	0.14	0.02	-0.10	0.03	0.07	0.20	0.34	-0.23	0.12	-0.22

Table 2.1: *Numerical expression data matrix.*

the disease then we expect the same value for all measurements. Since the measurements are not very precise, some noise is introduced, which is modeled by a normal random variable.

Therefore we can apply a t-test to see if the expression levels of tumor versus healthy are coming from the same distribution. For $gene_1$ we obtain a p-value of 1.190e-05, thus we can safely accept the alternative hypothesis: The samples came from different populations and the gene is differentially expressed. On the other hand, for $gene_2$ we obtain a p-value of 0.95, and thus the data support the null hypothesis: The samples came form the same population, thus $gene_2$ is not differentially expressed.

Fisher's exact test Let $X \sim binomial(m, p_X)$ and $Y \sim binomial(n, p_Y)$ be two independent observations from two binomial distributions with fixed m and n. We want to test the hypothesis

$$H_0 : p_X = p_Y \quad \text{versus} \quad H_1 : p_X \neq p_Y.$$

Under the null hypothesis ($p = p_X = p_Y$) the joint density of (X, Y) is

$$f(x, y \mid H_0) = \binom{m}{x}\binom{n}{y} p^{x+y}(1-p)^{m+n-(x+y)}.$$

We see that all the information is summarized in the random variable $X+Y$. So, $T = X+Y$ is a sufficient statistic under H_0. Knowing T, we can use the test statistic $g(X, Y \mid T = t) = X$, which gives evidence for rejecting H_0 if X attains an extreme values. A small value for X will result in a large value for $Y = T - X$ and vice-versa. It can be easily shown that the conditional density of X knowing $T = t$ is $hypergeometric(m+n, m, t)$. The probability for obtaining the value x, $0 \leq x \leq t$ is given by

$$P(x \mid T = t) = \frac{\binom{m}{x}\binom{n}{t-x}}{\binom{m+n}{t}}. \tag{2.1}$$

Thus, we can define a p-value for this test as

$$p(\mathbf{x}, \mathbf{y}) = P(\mathbf{x} \geq x \mid \mathbf{x} + \mathbf{y} = t) = \sum_{k=x}^{\min(m,t)} \frac{\binom{m}{k}\binom{n}{t-k}}{\binom{m+n}{t}}.$$

The test defined by this p-value is called Fisher's exact test.

We further describe how this test is used for deciding whether two characteristics (groups) $\mathcal{A}$ and $\mathcal{B}$ are independent in a population sample. First, a sample is taken from the population. Then a contingency table is formed, see Table 2.2.

Here S denotes the sample size, X is the number of sample points that are in both groups $\mathcal{A}$ and $\mathcal{B}$ and X' the number of sample points from groups $\bar{\mathcal{A}}$ and $\mathcal{B}$. Similarly Y and Y'

	$\mathcal{A}$	$\bar{\mathcal{A}}$	
$\mathcal{B}$	X	X'	M
$\bar{\mathcal{B}}$	Y	Y'	N
	T	T'	S

Table 2.2: *Contingency table*

denote the number of sample points from groups $\mathcal{A}$ and $\bar{\mathcal{B}}$, respectively $\bar{\mathcal{A}}$ and $\bar{\mathcal{B}}$. M and N are the marginals for group $\mathcal{B}$ and $\bar{\mathcal{B}}$. T and T' are the marginals for groups $\mathcal{A}$ and $\bar{\mathcal{A}}$.

Since we are sampling at random from a fixed size population the joint distribution of X, X', Y, and Y' is multinomial

$$P(X = x, X' = x', Y = y, Y' = y') = \frac{S!}{x!x'!y!y'!} p_{\mathcal{A}\mathcal{B}}^{x} p_{\bar{\mathcal{A}}\mathcal{B}}^{x'} p_{\mathcal{A}\bar{\mathcal{B}}}^{y} p_{\bar{\mathcal{A}}\bar{\mathcal{B}}}^{y'}.$$

where $p_{\mathcal{A}\mathcal{B}}$ is the probability of a sample point to be in both group $\mathcal{A}$ and $\mathcal{B}$ and so on. It can be shown that testing for independence between $\mathcal{A}$ and $\mathcal{B}$ is equivalent to testing if the ratio $\rho = \frac{(p_{\bar{\mathcal{A}}\mathcal{B}} p_{\mathcal{A}\bar{\mathcal{B}}})}{(p_{\mathcal{A}\mathcal{B}} p_{\bar{\mathcal{A}}\bar{\mathcal{B}}})}$ is equal to 1, see Section 4.6 from (Lehmann, 1986) for a complete derivation.

To carry out this test, we condition on the marginals $X + X' = m$ and $X + Y = t$. Under the hypothesis of $\rho = 1$ we obtain the hypergeometric distribution

$$P(X = x \mid X + X' = m, X + Y = t) = \frac{\binom{m}{x}\binom{n}{t-x}}{\binom{m+n}{t}}.$$

This is the same distribution as in the previous case, see equation (2.1), in which we tested for equality of two binomial probabilities.

In other words testing the independence of two groups in a contingency table conditioned on the marginal probabilities is the same as computing a Fisher's exact test. We will use Fisher's exact test for scoring groups of genes for enrichment as defined in Section 2.3.1.

2.2.2 Multiple testing

In the previous paragraph we discuss testing a single hypothesis for a data set. In many situations multiple questions are posed for an experiment or the same question is investigated in similar experiments. Thus, multiple hypothesis must be tested and the result of all tests should be included in the final result of the analysis, see Chapter 1 from (Westfall and Young, 1993).

The problem of determining if a gene is differentially expressed in a microarray experiment (see Section 2.1.3) fits naturally in the multiple hypothesis testing framework; the measurements of thousands of genes simultaneously generate large multiplicity problems and the aim of the analysis is to quantify in some statistical manner the differential expression of each gene.

We will use this problem to motivate and to introduce the basics of multiple testing. First, let $X = (x_{ij})_{m \times n}$ be the gene expression data matrix with m genes and n samples (patients or cell states). For each column j there is a response variable y_j. For example a binary variable discriminating between tumor or healthy cells. The columns of the matrix, together with the response variables are thought as random variables $(\mathbf{x}_j, y_j)$, for $j = 1, \ldots, n$ from a population of interest ($\mathbf{x}_j$ denote the random variable for the j^{th} column of matrix X). Let X_i denote the expression level of gene i and let Y be the response vector. For each gene i we form the null hypothesis

$$H_{0i}: \text{ there is no association between } X_i \text{ and } Y.$$

We saw in Section 2.2.1 that in the case of a binary response Y, that a appropriate test statistic is a two sided t-test. Based on the p-value computed for each gene i we accept or reject the hypothesis H_{0i}. In the case of testing a single hypothesis we control the Type I error. For example, we reject the null hypothesis if the p-value is smaller than $\alpha = 0.05$. To show why we cannot use the usual significance level α in the presence of multiple hypothesis we consider the following hypothetical experiment: The matrix X is randomly generated. All entries are iid. drown from a standard normal distribution. If p_i denotes the p-value of gene i, then it is easy to show that

$$p_{min} = P(\min_i p_i \leq \alpha) = 1 - (1-\alpha)^m.$$

p_{min} is the probability of rejecting at least one null hypothesis from m hypotheses at a specified level α. For $\alpha = 0.05$ and $m = 10$ we will reject at least one null hypothesis in 40% of the cases. For $m = 100$ the probability increases to 0.995. Under the null hypothesis we expect $m \times \alpha$ hypothesis to be rejected. In the case of microarray experiments with $m \sim 10.000$ tests, the result of the analysis will give us almost 500 differentially expressed genes even under the null hypothesis of no true differentially expressed genes.

Type I error rates. Due to the multiple testing problem it is necessary to define new types of error rates. There are several extensions of Type I error rate for multiple testing procedures, see (Dudoit et al., 2003b; Westfall and Young, 1993) for a detailed list. One such error rate is the Family-Wise Error rate (FWE).

Let H_0 denote the multiple test null hypothesis: $H_{0i_1}, \ldots, H_{0i_k}$ are true, for some $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, m\}$. This null hypothesis depends on the joint distribution of the test statistics T_i for all genes and is also called the *null distribution* of the test. Then, the FWE is defined as

$$\text{FWE} = P\{\text{reject at least one } H_{0i} \mid H_0\}.$$

Similar as in the single hypothesis testing scenario, a multiple testing procedure is controlling the FWE at a level α if FWE $\leq \alpha$.

Another used error rate is the False Discovery Rate (FDR) introduced by (Benjamini and Hochberg, 1
Let R denote the number of rejected hypotheses and let V denote the number of true null hypotheses that are rejected, the Type I errors. Then, the FDR is the expected proportion of Type I errors among the rejected hypotheses, that is

$$\text{FDR} = \begin{cases} E\left(\frac{V}{R}\right) & \text{if } R > 0, \\ 0 & \text{if } R = 0. \end{cases}$$

Adjusted p-values. Given an error rate, for example the FWE, one can define rejection regions or p-values for deciding on accepting or rejecting the null hypotheses. p-values are more useful, since they do not need the test level to be determined in advance. As in the single hypothesis case, a p-value (adjusted p-value) is defined for each test as

$$\tilde{p_i} = \inf\{\alpha \mid H_{0i} \text{ is rejected at FWE} = \alpha\}.$$

Note that the adjusted p-value depends on the null distribution.

Having computed the adjusted p-value, we reject H_{0i} at a FWE α, if $\tilde{p_i} \leq \alpha$. In our example, we say that gene i is differentially expressed. There are several methods for adjusting p-values. One of the simplest (and also the most conservative) method is the Bonferroni adjustment. If p_i denotes the unadjusted p-value then

$$\tilde{p_i} = \min\{mp_i, 1\}.$$

This is equivalent to rejecting H_{0i} at a FWE α, if $p_i \leq \alpha/m$. Thus we will reject a null hypothesis only if there is very strong evidence in the data. Other methods are not so conservative, see (Dudoit et al., 2003b; Ge et al., 2003) who compare different p-value adjustment methods for microarray data analysis.

Using adjusted p-values is appropiate when multiple questions are investigated simultaneously. However the raw p-values should also be considered as a measure of evidence; when computing adjusted p-values several assumptions, e.g. independence between the test statistics, are made depending on the method used. Thus, taking into account both raw and adjusted p-values the results of the analysis will be more accurate.

2.3 Ontologies

In this section we introduce some generalities about ontologies and discuss the use of ontologies in bioinformatics, focusing on Gene Ontology (**GO**). For further details on ontologies and their use in bioinformatics we refer to (Gruber, 1993; Kremer, 2001; Staab and Studer, 2004; Jones and Paton, 1999; Ashburner et al., 2000).

In in Artificial Intelligence (AI) ontologies are seen as formal models of shared understanding within a domain. They are the basis of Knowledge Management, aiming to bring a consensus in the way a specific domain is described.

There have been many attempts to define the term *ontology*. It was first introduced in philosophy, where an ontology is a systematic account of existence. Thus the meaning in AI is that *what exists is exactly that which can be represented*.

Definition. The Stanford Knowledge Systems Lab provides the following definition:

"An ontology is an explicit specification of some topic. For our purposes, it is a formal and declarative representation which includes the vocabulary (or names) for referring to the terms in that subject area and the logical statements that

> *describe what the terms are, how they are related to each other, and how they can or cannot be related to each other. Ontologies therefore provide a vocabulary for representing and communicating knowledge about some topic and a set of relationships that hold among the terms in that vocabulary."*

More formally an ontology is defined in (Gruber, 1993) as:

> *"A formal explicit specification of a shared conceptualization for a domain of interest."*

Here by conceptualization we understand a simplified view of the word that we wish to represent for some purpose.

The *Universe of Discourse* is the set of objects (*entities*) that can be formally represented for some domain. This set of objects together with the relations between the objects make up the representational *vocabulary*. Based on this vocabulary we can define an ontology by a set of representational terms. In such an ontology, definitions associate the names of entities in the universe of discourse (classes, relations, functions) with human-readable text describing what the names are meant to denote, and formal axioms that constrain the interpretation and well-formed use of these terms.

Hierarchical structure. Ontologies can be represented graphically by a semantic net or a conceptual graph, entities representing the nodes and the edges being the relations between them. In many cases the graph structure is as simple as a tree, but in the case of complex ontologies this is not the case. Since there are at least two types of relations between entities, namely *'is a member of'* and *'is a subset of'* plus other domain specific relations, undirected loops can be formed in the graph. We are particularly interested in the graph structure and we will exploit its properties throughout this thesis.

Bio-ontologies. Ontologies arise naturally in biology (Jones and Paton, 1999), since:

- *hierarchical structures of concepts* are of primal interest in the organization of knowledge in the *biological sciences*. As an example we have the classification of living entities into kingdoms, phyla, classes, orders, families, genera, species, etc.
- biological systems are generally *very complex*, especially at the molecular level. Here the hierarchical relations are complex and likely to exhibit any problems.
- scientific areas are generally more *formal* than non-theoretical ones. It is hoped, that analyzing biological theories will solve many of the problems that exist in non-technical, less formal areas.

Thus ontologies are needed in bioinformatics, giving the necessary structures and tools for working with the biological knowledge. In the last years more and more ontologies for bioinformatics appeared: *Sequence Ontology Project* (ontology describing features on a nucleotide or protein sequence), *Edinburgh Mouse Atlas Project* (a digital atlas of mouse development), *TAMBIS Ontology* (ontology of molecular biological and bioinformatics concepts), *BioCyc* (collection of Pathway/Genome Databases), etc. A good resource for the currently available ontologies for biological domains is Open Biological Ontologies[1].

[1]http://obo.sourceforge.net/

We are interested in the Gene Ontology (Ashburner et al., 2000; Consortium, 2001), the most widely accepted description of genes and gene-products for eukaryotes.

2.3.1 Gene Ontology

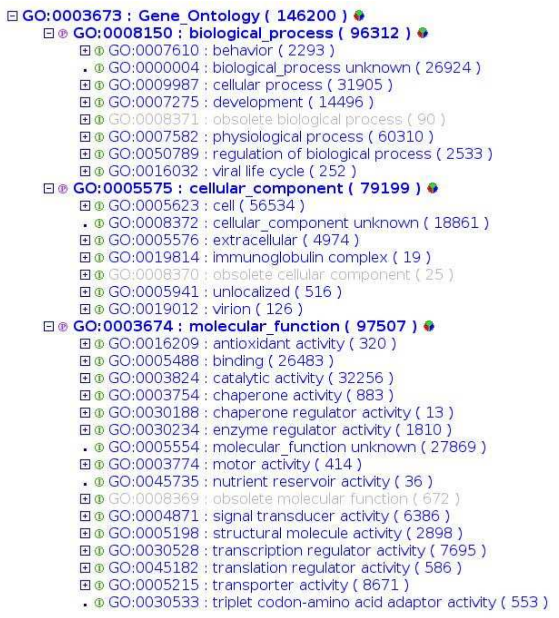
The Gene Ontology project started from the fact that a large fraction of gene products (a physical entity: a protein, or a functional RNA such as alpha-globin) that have a role in the core biological process are found in all eukaryotes. Thus the biological knowledge of these gene products can be transfered from one organism, for which experiments are easier, to other less tractable organisms. The goal of the GO project was to produce an ontology that describes the roles of genes and gene products in any organism.

The GO started in 1998 as a joint project between three model organism databases: FlyBase (Drosophila), Saccharomyces Genome Database (SGD), Mouse Genome Informatics (MGI). Later on the GO turned out to be widely accepted by the bioinformatic community and more organisms were added to it, including human. A detailed list of these can be found at the GO Web resource[2].

The ontologies. There are three ontologies that describe attributes of gene products in their independent biological areas (kingdoms): molecular function, biological process and cellular component.

A snapshot from the AmiGO browser, which shows the three different ontologies, is shown in Figure 2.6.

- **Molecular Function (MF)** describes the biochemical activity of a gene product. It describes activities and not the entities performing the actions (e.g. ligand, enzyme, kinase activity, etc).
- **Biological Process (BP)** describes activities that may require more than one step and need assemblies of gene products for achieving these activities. Activities involve physical or chemical transformations. Examples of BP are cell death, translation, pyramidine metabolism, etc.
- **Cellular Component (CC)** describes the location in the cell where a gene product is active. Examples of CC include ribosome, nuclear inner membrane.



```
⊟ GO:0003673 : Gene_Ontology ( 146200 )
    ⊟ GO:0008150 : biological_process ( 96312 )
        ⊞ GO:0007610 : behavior ( 2293 )
        . GO:0000004 : biological_process unknown ( 26924 )
        ⊞ GO:0009987 : cellular process ( 31905 )
        ⊞ GO:0007275 : development ( 14496 )
        ⊞ GO:0008371 : obsolete biological process ( 90 )
        ⊞ GO:0007582 : physiological process ( 60310 )
        ⊞ GO:0050789 : regulation of biological process ( 2533 )
        ⊞ GO:0016032 : viral life cycle ( 252 )
    ⊟ GO:0005575 : cellular_component ( 79199 )
        ⊞ GO:0005623 : cell ( 56534 )
        . GO:0008372 : cellular_component unknown ( 18861 )
        ⊞ GO:0005576 : extracellular ( 4974 )
        ⊞ GO:0019814 : immunoglobulin complex ( 19 )
        ⊞ GO:0008370 : obsolete cellular component ( 25 )
        ⊞ GO:0005941 : unlocalized ( 516 )
        ⊞ GO:0019012 : virion ( 126 )
    ⊟ GO:0003674 : molecular_function ( 97507 )
        ⊞ GO:0016209 : antioxidant activity ( 320 )
        ⊞ GO:0005488 : binding ( 26483 )
        ⊞ GO:0003824 : catalytic activity ( 32256 )
        ⊞ GO:0003754 : chaperone activity ( 883 )
        ⊞ GO:0030188 : chaperone regulator activity ( 13 )
        ⊞ GO:0030234 : enzyme regulator activity ( 1810 )
        . GO:0005554 : molecular_function unknown ( 27869 )
        ⊞ GO:0003774 : motor activity ( 414 )
        . GO:0045735 : nutrient reservoir activity ( 36 )
        ⊞ GO:0008369 : obsolete molecular function ( 672 )
        ⊞ GO:0004871 : signal transducer activity ( 6386 )
        ⊞ GO:0005198 : structural molecule activity ( 2898 )
        ⊞ GO:0030528 : transcription regulator activity ( 7695 )
        ⊞ GO:0045182 : translation regulator activity ( 586 )
        ⊞ GO:0005215 : transporter activity ( 8671 )
        . GO:0030533 : triplet codon-amino acid adaptor activity ( 553 )
```

Figure 2.6: *AmiGO viewer: listed are the three independent ontologies*

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

A gene product can be present in any of the three ontologies, implying that the relation between a gene product and the ontologies is a *one to many* relationship. This is in accordance with our biological knowledge (Ashburner et al., 2000) that a protein can

> *function in several processes, contain domains that carry out diverse molecular functions, and participate in multiple alternative interactions with other proteins, organelles or locations in the cell.*

The current status of the existing ontologies (January 2005) is shown in Table 2.3:

Ontology	**No. of Terms**
Molecular Function	7370
Biological Processes	8808
Cellular Components	1420

Table 2.3: *Current status of the GO terms for each ontology.*

From now on we will use *GO terms* to refer to the objects (entities) of the three ontologies. Also, we will use as a synonym *GO node*. Definitions for GO terms are taken from the Oxford Dictionary of Molecular Biology(Smith, 1997), or other sources like SWISS-PROT(Bairoch and Apweiler, 2000). For each term the source of the definition is stored and can be accessed. More details about this can be found in (Consortium, 2001).

The DAG structure. Similar to other existing ontologies, GO has a hierarchical structure that forms a *directed acyclic graph* (DAG). For such a graph we can use the notions of *child* and *parent*. The abuse of notations here is that a child can have multiple parents.

The child-parent relation between terms (nodes) can be of two types:

is a: type means that the child is an instance of the parent, e.g. a mitotic chromosome is an instance of a chromosome,

part of: type says that a child is a component of the parent, e.g. the telomere is a component of a chromosome.

Child terms can have different types of relations (*is a* or *part of*) with their different parents. This structure can be seen in Figure 2.7 for a small part of the GO ontology.

Each GO term in the ontology has a unique identifier. For example the root of the BP ontology has the identifier GO:0008150. From Figure 2.6 we see that each GO identifier is linked to a definition. There is almost no relation between the term identifiers, even if all nodes have at least one type of relation with other nodes. One reason is that the ontologies change with time and new terms can be introduced between a parent and a child.

Since in the literature the notions of child and parent are used with different meanings we emphasize again the way they will be used in this thesis.

For two GO nodes u and v, we will say that u is a parent of v if and only if v has any type of relation (*is a* or *part of*) with u. Thus, node v is more specific than node u. In this case v is the child of u.

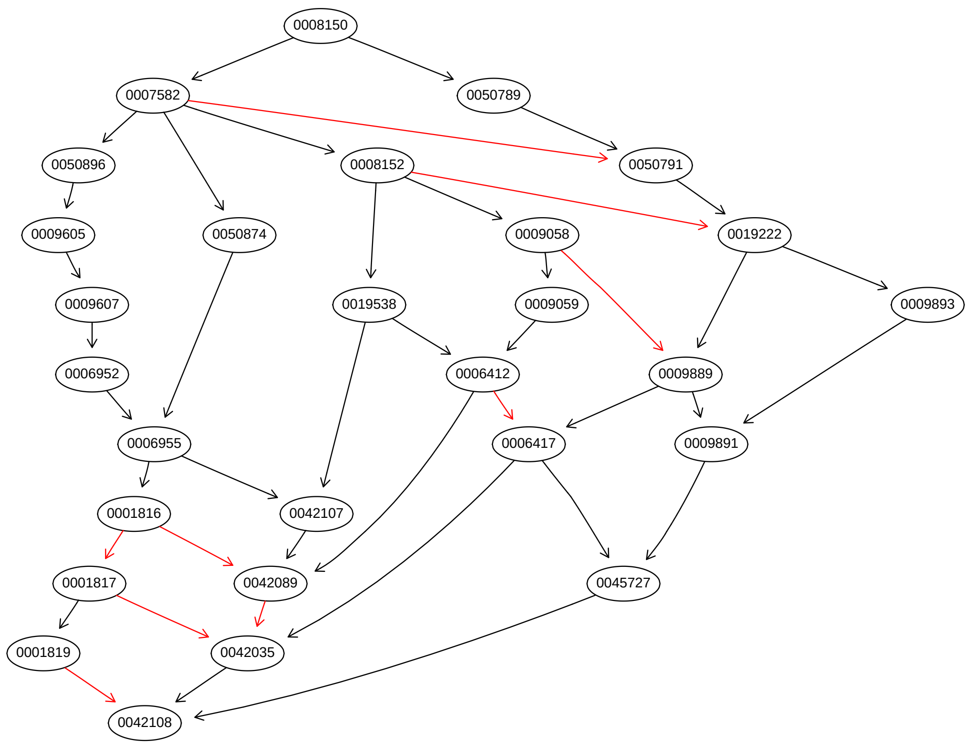


Figure 2.7: *A part of the GO graph (the black edges represent is a relation, and the red edges represent part of relation). The label of the nodes is the GO identifier: 0008150 ≅ GO:0008150*

Annotations. At this point the GO is just a structured vocabulary that defines GO terms for gene products with respect to the functions that they fulfil, the processes to which they contribute and their location within cells.

As we mentioned above the GO term represents the function or the process that a gene product has, respectively accomplishes, and does not represent the entity itself. Thus we need to associate genes and gene products with one or more GO terms, a gene product can have multiple functions, can be involved in more than one biological process, or can be found in different cells.

This is precisely the goal of the Gene Ontology Annotation (Camon et al., 2004).

> *To provide assignments of GO terms to all well characterized proteins and in particular to that of the human proteome.*

The focus is in providing annotations for the UniProt Knowledgebase.

Since the three ontologies are independent, the annotation of a gene product to one of them is also independent of its annotation to the other two. When a gene product is annotated with a GO term, the source, which may be a literature reference, or a computational analysis, is attributed to the annotation. Together with the source an evidence code is also stored, that tells the evidence on which the annotation is based. There is a standard set of evidence codes such as inferred from mutant phenotype (IMP), inferred from direct assay (IDA) or inferred from electronic annotation (IEA).

The GO annotations are obtained using both manual and electronic techniques (Camon et al., 2004).

- **Manual annotations** are obtained by exploring the information from published scientific literature by trained biologists. Since it involves human labor the process is time consuming but reliable, not considering the subjectivity of the biologist. It is used especially for more specific annotation, i.e. low-level terms in the GO hierarchy.
- **Electronic annotations** are generated using automated methods that either
  - convert old files which contain previous knowledge (e.g. converting the Enzyme Commission (EC) numbers for the already annotated proteins to GO terms, based on some mapping form EC numbers to GO id's),
  - use databases cross-references (e.g. UniProt and InterPro),
  - use automated information extraction methods that try to convert the biological knowledge from different sources (mainly scientific papers) to GO ontologies. There is a big hype in developing such methods (Hennig et al., 2003; Raychaudhuri et al., 200

The electronic annotations provide a fast and efficient way of associating high-level GO terms to a large number of proteins.

Since it is important to distinguish between manual and electronic annotations, the last ones are attributed with the IEA evidence code.

In the present thesis we make use of the GO annotation as a link between biological knowledge (the GO ontologies) and the gene expression data. Using the annotations we can map the genes from microarray experiments to the GO terms.

Since GO term definitions are widely accepted and assumed to represent as best as possible the biology behind the gene products it make sense to try to find out which GO terms are *relevant* for some experiment.

Chapter 3

Current methods for scoring GO terms

New methods have been proposed for analyzing gene expression data in which annotations from the GO database are integrated into the analysis in order to gain biological understanding. Gene classes obtained from GO are scored with respect to their significance.

In the following we analyze some of the proposed approaches for solving this problem.

Onto-Express. One of the first approaches is Onto-Express (Draghici et al., 2003). The authors' goal was to automate the process of translating the result of a microarray experiment to a list of functional categories, which should give better understanding of the underlying biological phenomena. At the same time, they proposed a statistical analysis of such functional categories.

To construct functional profiles of the experiment under study, first the functional groups need to be formed. For this, Gene Ontology (see Section 2.3.1) and Proteome[1] databases are used. Further, a list of *interesting genes* (genes that are regulated, differentially expressed genes, genes that are correlated with a phenotype and so on) is needed as the input. Since the origin of this list of genes does not matter, the method is not limited to microarray experiments.

For each functional group (e.g. a GO term) a test statistic is computed. The significance of the group and the number of genes mapped to the group are returned.

For the statistical analysis, the authors argue that by *comparing* the expected number of genes found in a specific functional group with the outcome of the experiment for the same group, the researcher knows if that functional group is relevant or not. For example, suppose that for a GO term we find 100 genes that are annotated (mapped) to it (see the GO annotations from Section 2.3.1) from the list of interesting genes and we expect 100 genes to be mapped to this GO term, then this functional group should not be reported as significant in spite of the large number of interesting genes. On the other hand, if for a GO term we find 30 genes in the list and we expect around 5 genes to be mapped to this group, then

[1]http://www.proteome.com/

this means that *we observe six times more genes than expected* and such a group should be considered significant.

To obtain statistically correct results, the authors use an urn model. Let $\mathtt{Bio}$ be a functional group. Suppose that there are N genes on the microarray (balls in the urn), M of which are in group $\mathtt{Bio}$ (white balls) and $N - M$ are not (black balls). If K genes are selected at random (in our case the list of interesting genes), then we are interested in the probability that exactly x genes from K are in $\mathtt{Bio}$. In this case we do sampling without replacement (a gene once selected cannot be selected again) and the distribution of x is modeled by a hypergeometric distribution, see equation (2.1) from Section 2.2.1.

For calculating the significance of a functional group based on the above model, a p-value can be computed by summing the probabilities of getting at least x genes in group $\mathtt{Bio}$. This test is similar to Fisher's exact test, see Section 2.2.1.

The authors argue on the use of asymptotically equivalent tests, such as the binomial test and the χ^2 test. They use all these tests depending on the size of the data, using one test where the others lack in giving accurate results.

Some factors that influence the result of the analysis are discussed. First, the method depends on the quality of the input. If there are false positives in the list of interesting genes, then the result will be noisy, the test not accounting for this error. Second, a bias is induced by the arrays that are enriched with a certain type of genes. Third, some caution needs to be taken when judging p-values. There is no multiple testing correction (see Section 2.2.2). Finally, interpreting the significance of a functional group depends on the initial list of interesting genes.

The same ideas are used in several tools: FatiGO (Al-Shahrour et al., 2004), GoMiner (Zeeberg et al., 2003), MAPPFinder (Doniger et al., 2003), GOstat (Beissbarth and Speed, 2004), GO::TermFinder (Boyle et al., 2004) and GOstats (Gentleman, 2004c). Although these tools offer almost the same functionality, the accent being put on the software implementation and availability (issues that are important but beyond the purpose of this thesis), we will briefly emphasize the difference between them.

FatiGO. As the name states, only Gene Ontology categories are used as functional groups. The first difference to the method proposed by (Draghici et al., 2003) is the way in which the GO terms to be investigated are chosen. A GO level (see Section 4.1 for a proper definition of a DAG level) is given by the researcher, and only the GO terms from this level are analyzed. Then, using the GO hierarchical structure the genes are *pushed up* the DAG, until they are mapped to the terms on the specified level, see Section 2.3.1.

For computing the significance of a GO term Fisher's exact test is used. The multiple testing problem is considered by the availability of three p-value adjustment methods: Bonferroni FWE adjustment (see Section 2.2.2), (Benjamini and Hochberg, 1995) and (Benjamini and Yekutieli, 2001) false discovery rate adjustment.

Another problem that is addressed is the non-unique annotation of the genes, the tool being able to cope with it.

GoMiner. As FatiGO, GOMiner only uses the Gene Ontology database. As functional groups all GO terms are scored for significance. Fisher's exact test is used and some multiple testing corrections are possible. The authors argue that the resulting p-values should be considered more as *heuristic measures*, than as measures of the amount of statistical significance.

Two issues are pointed out. The first one is about the dependence of gene data. This means that gene IDs that are mapped to the same GO term can code for the same gene. This affects the significance of a group; it may happen that half of the genes mapped to a GO term are duplicates and thus this GO term is statistically enriched. Normally this issue should be solved in an earlier stage of the analysis, but given its consequences, it should be addressed.

The second issue is the visualization. An interactive DAG browser is available, a tool that helps in understanding the data, somehow showing the strong dependences between the GO terms.

MAPPFinder. This tool integrates GO analysis and biological pathway maps. The functional groups are again GO terms. Three scores are computed for each GO term.

Two of them are simple statistics that show how well represented a GO term is, namely the percentage of interesting genes within each GO term and the percentage of all genes in a GO term.

The third score is a z-test (see (Lehmann, 1986) for more details, roughly a standardized differences test). The reason for applying this test is the following. If two groups (GO terms) contain the same number of genes, then the term that contains more interesting genes should be more significant. To account for the different sizes of the groups there is a need to divide by the standard deviation.

MAPPFinder does not return any p-value (thus no multiple testing correction), the results being sorted by any of the scores.

GOstat. In this approach the authors point out that GO is a hierarchical structure and understanding the biological functions involved in an experiment should include the exploration of this DAG structure. This idea partly motivated our work.

The enrichment of the GO terms is scored using a χ^2 test and Fisher's exact test. Methods for p-value adjustment are also available.

Some topological notions like the *path* of a GO term and *splits* are defined. The authors argue that the GO terms from the same path (the splits) are strongly correlated. To ease the interpretability of the result they propose to cluster the GO terms (by the genes that are mapped to them), somehow looking for subgraphs in the GO graph that are relevant for the experiment under study.

GOstats. This software is a package from Bioconductor Project[2]. Although no new methods are proposed, the author points out the challenges that such an analysis poses.

The score for a GO term is computed with a hypergeometric test. Two issues are raised. The first is the interpretation of the p-values. The author argues that given the strong dependence between the tested hypotheses there is no clear way in which a multiple testing correction should be done. The researcher should concentrate more on *'patterns of p-values that correspond to structure in the GO graph'* rather than the choice of a cutoff.

[2]www.bioconductor.org

The second issue is the number of genes that are mapped to a particular GO term. The most specific GO terms will have a relatively small number of genes annotated to them and this will influence the result of the experiment. For example, if a GO term contains 3 genes and all of them are in the list of interesting genes, then a small p-value is obtained. The author proposes that only GO terms with *a reasonable number of genes* should be considered for the analysis.

Since the analysis is performed in the R environment different types of statistics and visualizations are possible.

Other methods for the analysis of gene set enrichment were proposed. In (Mootha et al., 2003) a normalized Kolmogorov-Smirnov test was used for scoring gene sets (pathways, GO terms, gene clusters). The method of (Breitling et al., 2004) iteratively compiles gene sets from the list of differentially expressed genes, based on *evidence graphs*.

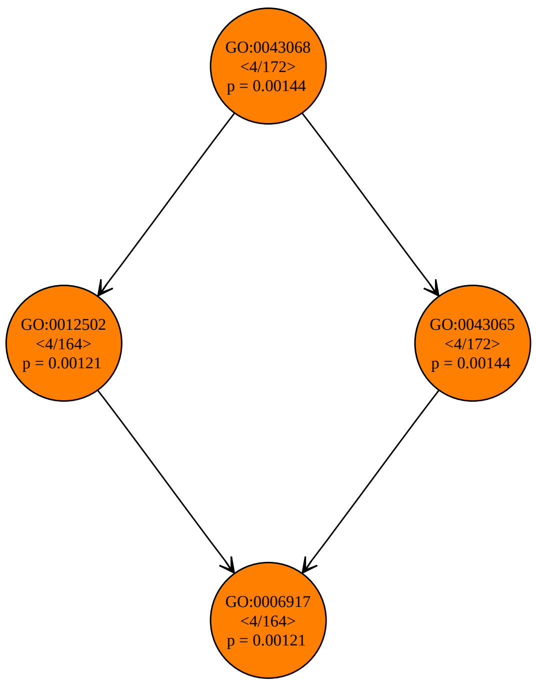
3.1 Remarks

As we have seen in some of the above methods the dependences between the functional groups - the GO terms - are not taken into account. Of course in tools like Onto-Express and MAPPFinder in which databases other than Gene Ontology are used this is impossible.

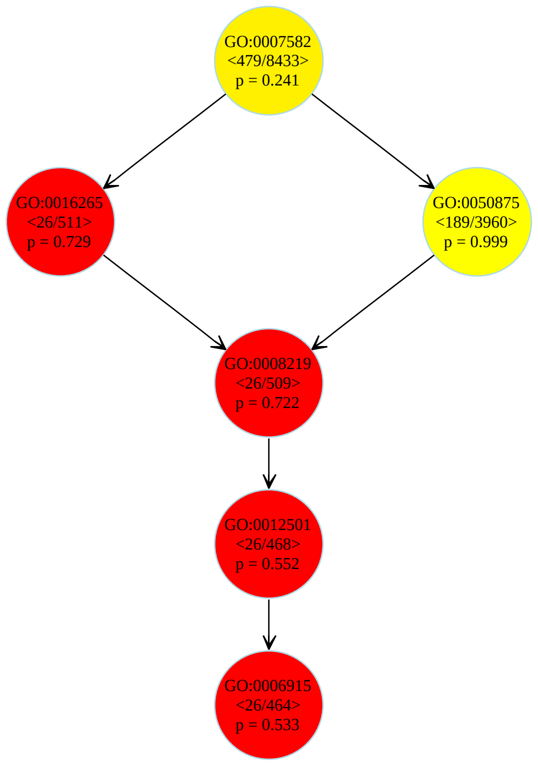
There is one important reason why the DAG structure should be exploited. As we argue in Section 2.3, the hierarchical structure refers to the biological relations between GO terms: GO terms that are neighbors have similar functions, only that the child is more specific than its parent. The analysis should try to *control* the level of specificity. This does not mean that only GO terms from a particular level, as in FatiGO, should be considered for investigation, but the result of the analysis should account for it.

The strong correlation between neighboring GO terms can be seen in Figure 3.1. In the example from figure 3.1(a) the GO terms GO:0043068, GO:0012502 and GO:0006917 are on the same path. The GO terms GO:0012502 and GO:0006917 have the same genes mapped to them, so they are identical from this point of view. In this case, when Fisher's exact test is employed (or any other of the tests previously mentioned) the same p-value will be assigned to these terms and thus they are equally relevant. Even if the numbers of genes that are mapped to these nodes are not exactly equal, as for the GO terms GO:0043068 and GO:0043065, the resulting p-values are almost equal. In the example from figure 3.1(b) there are four GO terms on the same path (the red ones) with similar numbers of mapped genes, and thus with similar p-values. This situation will appear quite often in practice and should be considered in the analysis. By clustering the GO terms, (Beissbarth and Speed, 2004) try to account for these dependences.

Another important issue to be addressed is the omission of some genes and some GO terms. In FatiGO it is not clear what will happen with the genes that are annotated at a GO level higher than the selected one. Some of these genes can belong to the list of interesting genes. Not considering GO terms that have few genes, as in (Gentleman, 2004c), one could leave out some relevant terms. Thus, the analysis should consider all terms and all available genes.



(a) Example 1



(b) Example 2

Figure 3.1: *Examples of node dependences. For each GO term the counts and the* $\mathtt{p}$*-values are displayed.* $< \mathtt{x}/\mathtt{y} >$ *denotes that out of* $\mathtt{y}$ *genes mapped to the node,* $\mathtt{x}$ *belong to the list of interesting genes.*

All methods described above, except (Mootha et al., 2003), are just looking at simple gene counts inside the functional group. Some tests that can account for the distribution of the genes in the functional group, like a Kolmogorov-Smirnov test, can improve the inference.

There is no method that uses different statistical tests. It will be interesting to compare the results of tests like Fisher's exact test, z-score and Kolmogorov-Smirnov test.

Chapter 4

Integrating the GO topology

We have seen in the previous chapters, that an important issue is the biological interpretation of the reported scores - the p-values. It is emphasized that using adjusted p-values, obtained with various multiple testing correction methods, can improve the statistical interpretation of the result.

Multiple testing correction. We believe that the main problem is not the multiple testing correction, even if it is necessary, but it is the test statistic used to compute the significance of a GO term. Looking at the examples from Figure 3.1 we see that even after a p-value adjustment the resulting p-values are equal. A test statistic is needed that *considers the neighborhood* of a GO term - its parents and its children - when computing its p-value. By considering the neighborhood we mean to decorrelate the GO terms such that if a GO term is reported as significant, then none of its neighboring nodes that have similar counts should be reported significant. The problem can be seen in Figure 3.1(a) in which all four GO terms are almost identical w.r.t. the number of mapped genes.

The methods presented in this chapter try to *augment* (enhance) the analysis of the functional groups - the GO terms - using the graph topology. There are basically two types of algorithms. In the first type of algorithm some genes from a GO term are removed based on the significance of its children. The second type of algorithm assigns arbitrary weights to the genes from a GO term.

For the first type (Section 4.2), we present three algorithms, with different levels of complexity.

topo: The idea of the first algorithm is to compute the p-value of a node after all the genes from its significant children have been removed from it.

elim: In the second algorithm, when a node x is found to be significant, all its genes are removed from all ancestors of node x. Thus, the ancestors are becoming less enriched.

readjust: For the third algorithm two p-values are computed for each node. The first p-value is computed disregarding the neighborhood of the node. The second p-value is computed in the same way as in topo algorithm, but this time a child is considered only if its first p-value is smaller than a specified cutoff.

The second type of algorithm generalizes the ideas introduced by the above algorithms (Section 4.3).

weight: The central idea is to associate single genes mapped to a GO term with weights, denoting their relevance. Then, a modified Fisher's exact test is employed for the weighted GO term. The weights for the genes mapped to a GO term are computed based on the current p-values of the investigated GO term and its neighbors. The presented algorithm is a *template* that leaves room for customization, regarding the choice of the weights, the choice of the strategy in which the weights are spread into the graph and the way the weights are updated.

Before describing the algorithms in detail, we introduce some definitions and notations.

4.1 Definitions and notations

In the rest of this chapter we present the algorithms that try to solve the problem stated in Figure 1.1. The algorithms are explained using more general notions, since we want to emphasize that they work not only for microarray data. Thus we will use the following notions: The genes are regarded as *items*, the GO graph is a *DAG* (directed acyclic graph) and the GO terms are the *nodes* in the DAG.

Next we introduce some graph notions. For a review on graph definitions and algorithms we refer to (Cormen et al., 2003).

Graphs. We call the roots of a DAG all nodes that have in-degree (the number of edges entering the node) equal to 0. Since the GO DAG has only a single root we only consider DAGs with exactly one root. The algorithms also work for DAGs with more than one root. In Section 2.3.1 we define the notions of *child* and *parent* for such graphs. The root is the only node in the DAG that has no parent. Similarly the nodes that have no children are called *leaves*. The edges are thus directed from a parent to a child. For a DAG we can define the *level* of a node: The level of a node u is *the length of the longest path from the* root *to* u. Thus, the root is on level 0, its direct children are on level 1 and so on. Note that we can have leaves on each level (except level 0) and nodes on the same level do not have any edge between them. The nodes' levels can be computed easily with a modified BFS traversal. For a set of nodes $\mathcal{U}$, lower.inducedGraph($\mathcal{U}$) is defined as the subgraph induced by all nodes reachable from $\mathcal{U}$. Similarly, we define upper.inducedGraph($\mathcal{U}$) as the subgraph induced by all nodes reachable from $\mathcal{U}$ if we reverse all edges in the DAG (or equivalently the graph that contains all paths from root to any node $u \in \mathcal{U}$). If $\mathcal{U} = \{u\}$, then we write lower.inducedGraph(u). Reversing all edges in a graph is a simple operation. For example, if the graph is stored in an adjacency matrix, we only have to transpose this matrix. Thus, both lower.inducedGraph() and upper.inducedGraph() can be obtained fast.

Items. As stated in the problem from Figure 1.1 the nodes in the DAG contain *items*. items(u) denotes the set of items from node u. Since items(u) is a set, the usual set operations are used, e.g. $(\mathsf{items}(u) \setminus \mathsf{items}(v)) \cup \mathsf{items}(w)$ denotes the union of the items from node w with the set difference between the items from nodes u and v. If $\mathcal{L}$ is a list of nodes, then

$$\mathsf{items}(\mathcal{L}) = \bigcup_{u \in \mathcal{L}} \mathsf{items}(u).$$

When the union of two or more sets of items is computed, duplicates are removed.

Fisher's exact test. In our algorithms we use as scoring function for a node u the degree of independence between the two characteristics: $\mathcal{A} = \{$*item is in the list of interesting items*$\}$ and $\mathcal{B} = \{$*item is found in node* $u\}$. We saw in Section 2.2.1 that this is achieved with Fisher's exact test. Here we present how the contingency table is formed in our case. Let $sigItems$ represent the set of the interesting (significant) items, e.g. differentially expressed genes, and $allItems$ denote the set of all items mapped in the DAG. To build a contingency table as shown in Table 4.1, we define for a node u the following:

$$\overline{\mathsf{items}(u)} = allItems \setminus \mathsf{items}(u)$$
$$\overline{sigItems} = allItems \setminus sigItems$$

$$X = |sigItems \cap \mathsf{items}(u)|, \qquad \overline{X} = |sigItems \cap \overline{\mathsf{items}(u)}| \tag{4.1}$$
$$Y = |\mathsf{items}(u) \setminus sigItems|, \qquad \overline{Y} = |\overline{\mathsf{items}(u)}| - Y \tag{4.2}$$

	sig items	not sig items	sum
items in u	X	$\overline{X}$	$\lvert\mathsf{items}(u)\rvert$
items not in u	Y	$\overline{Y}$	$\lvert\overline{\mathsf{items}(u)}\rvert$
sum	$\lvert sigItems\rvert$	$\lvert\overline{sigItems}\rvert$	$\lvert allItems\rvert$

Table 4.1: *Items contingency table*

The function $\mathsf{Fisher.test}(\mathsf{items}(u), sigItems)$ is computing a p-value based on the contingency table from Table 4.1.

Pseudo-Code. The algorithms are presented in pseudo-code. We use general lists as data structures. For example, $\mathtt{nodeSig}$ denotes a list. To access the elements we use the operator [ ]. Lists can be indexed with nodes from the DAG, e.g. to assign the $value$ to a node u we write $\mathtt{nodeSig}[u] \longleftarrow value$. Lists are seen as sets and thus the set operations are used on them.

Preprocessing. The input of the algorithms are a DAG and two lists of items, namely the list of all items and the list of interesting items. For each node u, the list of all items that are mapped to it is given by $\mathsf{items}(u)$. The list of all items is denoted by $allItems$. In Section 5.2 we explain how the DAG and the two list of items (genes) are obtained using the Gene Ontology and a microarray experiment.

The classic algorithm

With all these steps done, the classical approach in which each node is tested separately for significance (see Chapter 3) can be summarized in Algorithm 0. We call this algorithm classic.

Algorithm 0 classic

```
1 $nodeSig \leftarrow \emptyset$
2 for $x$ in nodes(DAG)
3     $nodeSig[x] \leftarrow$ Fisher.test(items($x$), $sigItems$)
4 end
5 return $nodeSig$
```

After the p-value is computed for each node in lines 2-4, different multiple testing corrections can be employed. In the rest of this chapter we assume that the p-value adjustment is done by the function Fisher.test$(\cdot, \cdot)$, if necessary.

4.2 Eliminating items

In Algorithm 0 the significance of a node is computed independently of the significance of the neighboring nodes. Our belief is that the neighborhood of a node influences the significance of the node. Since in the GO graph the children of a node are more specific than the parent, the idea is to compute the significance of a node considering the significance of its children.

4.2.1 The topo algorithm

Here we directly implement this idea. Figure 4.1 shows the children of a node x together with their p-values. If we consider a level 0.01 test, then only $x.ch[2]$ is considered significant.

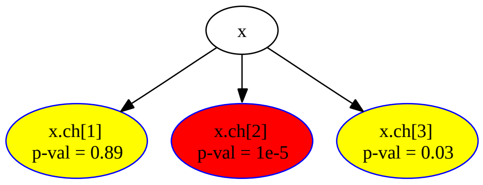


Figure 4.1: *The adjusted p-values for the children of node x. The red node is the only significant child of node x.*

Node $x.ch[2]$ is enriched with interesting items, since it is significant. Given the items dependences between a parent node and its child, all items from node $x.ch[2]$ are also found in node x. Thus, the significance of node $x.ch[2]$ influences the enrichment of node x. We want to know how enriched the parent is if we do not consider the items from its significant children. If node x is found significant even after the items from node $x.ch[2]$ are removed from it, then it is clear that node x also represents the list of interesting items (the biological function represented by the GO term x is more relevant for the experiment under study), independent of the more specific child $x.ch[2]$.

Algorithm 1 topo

```
1  $sigNodes.LookUP \leftarrow \emptyset$
2  $nodeSig \leftarrow \emptyset$
3  get the DAG levels list $DAG.level$
4  for $i$ from max($DAG.level$) to 1
5      $currNodes \leftarrow DAG.level[i]$
6      for $x$ in $currNodes$
7          $x.ch \leftarrow \text{children}(x)$
8          $sigCh \leftarrow sigNodes.LookUP[x.ch]$
9          $\text{items}(x) \leftarrow (\text{items}(x) \setminus \text{items}(sigCh)) \cup \text{items}(x.ch \setminus sigCh)$
10         $nodeSig[x] \leftarrow \text{Fisher.test}(\text{items}(x), sigItems)$
11         if $nodeSig[x] \leq cutOff$ then
12             add $x$ to $sigNodes.LookUP$
13         fi
14     end
15 end
16 return $nodeSig$
```

The pseudo-code for algorithm topo is shown in Algorithm 1. In line 1 a lookup table that keeps the significant nodes is initialized. Similarly, in line 2 the list for storing the p-values of all nodes is initialized. Since we need to compute the significance of the children before computing the significance of the parent node itself, we need to start with the nodes on the first level. The $DAG.level$ list computed in line 3 is used to address this task. $DAG.level[i]$ contains all nodes from level i in the DAG. $max(DAG.level)$ gives the maximum level in the DAG. As stated in Section 4.1 there is no parent-child relation between nodes on the same level. Thus, we can start from the lowest level and independently compute the significance of the nodes in this level. Then iteratively we decrease the level. This is done by the loop in lines 4-15.

In line 5 all nodes from level i are selected. Then, each of them is sequentially processed in lines 6-14. For each node x its children are selected, and the list of significant children $sigCh$ is computed. Then, the items from $sigCh$ are removed from node x. It can happen, due to the structure of the GO graph, that some items in $sigCh$ are also found in other not-significant children. We do not want to remove these items from x since they can come from different most specific nodes (nodes where the items are originally mapped). Thus, after having removed items from $sigCh$ we add all the items from all other children, see line 9. Then Fisher's exact test is employed for computing the p-value of node x. Based on this p-value we decide if node x is significant or not. Here we can use the simple Bonferroni adjustment of p-values. If the node is found significant then it is added to sigNodes.LookUP list. After finishing level i we move to nodes from level $i - 1$.

The algorithm returns the list with the nodes' significance. This algorithm is linear in the number of nodes and edges of the DAG: For each node we look at its children, thus we access each edge only once.

4.2.2 The elim algorithm

In the topo algorithm we only consider the children of a node when its p-value is computed. The problem with this approach is that it does not account for a larger neighborhood. All the knowledge about the children's neighborhood or the nodes from the lower levels is not used. Using the current p-value of a child to determine if the items from this child should be removed from the parent node can be misleading.

Algorithm 2 elim

```
1  sigNodes.LookUP ← ∅
2  elimItems.LookUP ← ∅
3  nodeSig ← ∅
4  get the levels list DAG.level
5  for i from max(DAG.level) to 1
6      currNodes ← DAG.level[i]
7      for x in currNodes
8          items(x) ← items(x) \ elimItems.LookUP[x]
9          nodeSig[x] ← Fisher.test(items(x), sigItems)
10         if nodeSig[x] ≤ cutOff then
11             add x to sigNodes.LookUP
12             for u in upper.inducedGraph(x)
13                 elimItems.LookUP[u] ← elimItems.LookUP[u] ⊕ items(x)
14             end
15         fi
16     end
17 end
18 return nodeSig
```

The idea of the elim algorithm is to remove items of a significant node from all its ancestors. This is a quite *radical* approach since nodes at the lower levels (GO terms that are more specific) will be reported as significant; their ancestors will loose significance. This approach works well for the example in Section 3.1.

The pseudo-code that implements this idea is shown in Algorithm 2. In the first lines we initialize the lists. The elimItems.LookUP list is used for storing for each node x the list of items that should be removed when it is investigated. The nodes are processed in the same way as in Algorithm 1. Each node x from level i is processed in lines 7-16. We know from previously processed nodes (also its children) which items we should remove from node x, see line 8. Based on the p-value computed with Fisher's exact test node x is declared significant, and if so, we remove items(x) from all its ancestors. This is done in lines 12-14. All the ancestors of node x are nodes in upper.inducedGraph(x). The $\oplus$ operation from line 13 means in the general case *union*, but other operations can be used.

Since we need to access the nodes form upper.inducedGraph(x) for some nodes x, this algorithm is quadratic in the number of nodes of the DAG in the worst case.

4.2.3 The readjust algorithm

Another way to address the problem that motivated the elim algorithm is to compute two p-values for each node.

In the topo algorithm, the items from the significant children are eliminated from the parent node x. Assume that a node x is found non-significant after this operation. When one parent of node x is processed, the items from node x are not removed, since x is a non-significant child. Thus the significance of its parent is artificially increased.

Algorithm 3 readjust

```
1  $classic.nodeSig \leftarrow \text{classic}()$
2  for $x$ in nodes($DAG$)
3      if $classic.nodeSig[x] \leq cutOff$ then
4          add $x$ to $sigNodes.LookUP$
5      fi
6  end
7  $nodeSig \leftarrow \emptyset$
8  get the levels list $DAG.level$
9  for $i$ from max($DAG.level$) to $1$
10     $currNodes \leftarrow DAG.level[i]$
11     for $x$ in $currNodes$
12         $x.ch \leftarrow \text{children}(x)$
13         $sigCh \leftarrow sigNodes.LookUP(x.ch)$
14         $\text{items}(x) \leftarrow (\text{items}(x) \setminus \text{items}(sigCh)) \cup \text{items}(x.ch \setminus sigCh)$
15         $nodeSig[x] \leftarrow \text{Fisher.test}(\text{items}(x), sigItems)$
16     end
17 end
18 return $nodeSig$
```

In Algorithm 3 the pseudo-code for solving this problem is shown. For each node x two p-values are computed. The first p-value is independent of the neighborhood of node x. This p-value tells if node x was significant or not before accounting for the topology. Thus it acts as a memory. The second p-value depends on the children and gives the final significance of a node. The same elimination of the items as in the topo algorithm is used, but this time we use the first p-values in the computation. A child of node x is significant if its first p-value is less than a specified cutoff. In this way we better control the way items are removed from node x.

The first p-values are computed using the classic algorithm. Having the p-values for all nodes computed, different multiple testing adjustment procedures can be employed. Thus the cutoff is more robust. Computing the list of significant nodes is done in lines 1-6. We assume the classic() procedure returns adjusted p-values.

The rest of the algorithm is almost identical to algorithm topo, see Algorithm 1. By removing items from a node the p-value typically increases. Thus, there is no need to add nodes to $sigNodes.LookUP$ list, see lines 11-13 in Algorithm 1. The running time of the readjust algorithm is linear. It is essentially the same as the running time of the topo algorithm, since the classic() procedure is linear in the number of nodes.

4.3 Weighting items

In the previous section we account for the correlation between nodes by removing items from them. Removing items from a node can be seen as a weighting of the items with weights equal to either 0 or 1. In this section we generalize to arbitrary weights. Using arbitrary weights the elimination of items is smoother.

Briefly, the weight algorithm works as follows. We want to decide if a node x is better representing the interesting items than any other node from its neighborhood. To do this, we look at its children and we compare its p-value with the p-values of its children. If there are children that have a p-value less than node x than we decide that these children better represent the interesting items. Let $sig.children$ denote the set of these children. Then we should report as significant nodes from $sig.children$ and not node x. To do this we down-weight the items that are mapped to the $sig.children$ in node x and in all ancestors of node x. Down-weighting means that the weighted items will contribute less when computing the p-value for the weighted group.

The children that have a p-value above the p-value of node x should not be reported as significant. To achieve this, all the items from these children are down-weighted. The reason is similar to the one presented above: All the items mapped to the children are found in node x and since we 'decided' that node x better represents the interesting items, we want these items to contribute less in the children's p-values.

Three issues need to be addressed. First, for scoring of a node, Fisher's exact test needs integer quantities, but by assigning arbitrary weights to the items, non-integer numbers can occur in the contingency table. The second issue is the way in which the items should be weighted. The last issue is the order in which the children of a node should be processed.

4.3.1 Scoring weighted groups

In the beginning all items have weight 1. The weights that are assigned to items during the run of the algorithm lie in the interval [0, 1]. For scoring a group we apply Fisher's exact test on a *weighted contingency table* built in the following way. The quantity X (the number of significant items that are mapped to node u in Table 4.1) is computed by summing up the items' weights followed by rounding up to the next integer.

$$\mathsf{X} = \left\lceil \sum_{i \in \{sigItems \,\cap\, \mathsf{items}(u)\}} weight[i] \right\rceil . \tag{4.3}$$

Similarly, all the quantities of the contingency table shown in Table 4.1 are computed by summing up the weights and rounding up of the items from each set. Here, we are replacing the cardinality function $|\mathcal{S}|$ with the function $\lceil \sum_{i \in \mathcal{S}} weight[i] \rceil$. We consider the weighted contingency table as a transformation of the table in which all weights are equal to 1. The idea here is to first decorrelate the nodes by giving weights to their items and then apply Fisher's exact test on the decorrelated nodes. Note that the quantity in equation (4.3) is always less than the quantity in equation (4.1). On the other hand, the smallest value for the quantity in equation (4.3) is obtained when all weights are either 0 or 1. This is the case for all algorithms presented in Section 4.2. Thus by using weights from the interval [0, 1] we are between the classical approach and the elimination approach. The function $\mathsf{WFisher.test}(\cdot,\cdot,\cdot)$ is used for scoring a group of items.

4.3.2 Weights computation

The weights for a group of items are computed in the following way. Let x be the node that is currently processed and let $x.ch$ be a child of x. Then we define the weight of this pair of nodes by

$$w = sigRatio(sig(x.ch), sig(x)), \tag{4.4}$$

where $sig(x)$ denotes the 'significance' of node x. This can be either the p-value or the value of the test statistic. Typically, we use the p-value returned by Fisher's exact test. The function $sigRatio(\cdot,\cdot)$ is a ratio function of the form

$$sigRatio(a,b) = \frac{f(a)}{f(b)} \quad \text{or} \quad sigRatio(a,b) = f\left(\frac{a}{b}\right). \tag{4.5}$$

The form of the function $sigRatio(\cdot,\cdot)$ depends on the function $sig(\cdot)$. When $sig(\cdot)$ denotes the p-value, the first form is preferred. The function $f(\cdot)$ is an increasing function and it is chosen depending on the degree of desired weighting of the two nodes. The main idea behind this formula is that we want to obtain a number smaller than 1 if the child $x.ch$ is less significant than its parent x.

4.3.3 The weight algorithm

The main body of the algorithm is presented in Algorithm 4.

In lines 1-4 we assign the weight 1 to all items. Each node is associated with the weights for its items. We use two lists: $downNodes.LookUp$ stores for each node that has been visited the weights of the items mapped to it, and $upNodes.LookUp$ stores the items' weights for the ancestors of visited nodes; these are nodes that have not been processed until now. We could also use only a single list that stores the items' weights for each node, but having two

Algorithm 4 weight (part I)

```
function main()
1  for u in nodes(DAG)
2      upNodes.LookUP[u] <- 1
3      downNodes.LookUP[u] <- 1
4  end
5  nodeSig <- $\emptyset$
6  get the levels list DAG.level
7  for i from max(DAG.level) to 1
8      currNodes <- DAG.level[i]
9      for node in currNodes
10         computeTermSig(node, children(node))
11     end
12 end
13 return  nodeSig
```

lists gives us more flexibility. For each list we can have different strategies of updating the weights.

Similar to the algorithms presented in the previous section we process the nodes bottom-up level by level, see lines 7-12. The core of this algorithm is the function computeTermSig$(\cdot, \cdot)$ whose pseudo-code is shown in Algorithm 5.

computeTermSig$(\cdot, \cdot)$ is a recursive function. We start with all children of the currently processed node x, see line 10 in Algorithm 4. Some of the children are eliminated and in the next call of the function we recurse on the remaining children, see line 20. The children that are considered by the computeTermSig$(\cdot, \cdot)$ function in each call are denoted *active children.* We finish processing node x, when there are no more active children. We take care of this in line 3.

In line 1 the weights of node x are obtained. Based on these weights a weighted contingency table is computed and Fisher's exact test is employed, see line 2. Thus, the p-value for node x is computed each time the computeTermSig$(\cdot, \cdot)$ function is called. The reason is that the items' weights for node x were updated during the previous run of the function.

At this point nodeSig[x] holds the temporary p-value of node x. This p-value can be modified when a parent or an ancestor on node x is processed (by the function recomputeSig(x)). In lines 4-6 the weights for node x and each active child are computed, see Section 4.3.2. Based on these weights we split the active children in two sets, namely children that are more significant than node x, denoted by sig.children, and children that are less significant than node x. The children in the second set will later become the new active children. The splitting is done in line 7. The idea behind such a splitting is the following: Children that have a p-value less than node x (we say that the children are more 'significant' than the parent) should be considered local optima (w.r.t. the node significance). To emphasize this the p-value of their parents and their ancestors is increased. This is done by weighting the items from the parents and from the ancestors with the computed weights. The children that

Algorithm 5 weight (part II)

```
function computeTermSig($x$, $children$)
 1  $itemsWeight \leftarrow upNodes.LookUP[x]$
 2  $nodeSig[x] \leftarrow$ WFisher.test(items($x$), $sigItems$, $itemsWeight$)
 3  if $children = \emptyset$ then return fi
 4  for $ch$ in $children$
 5      $weights[ch] \leftarrow$ sigRatio($nodeSig[ch], nodeSig[x]$)
 6  end
 7  $sig.children \leftarrow \{ch \mid weights[ch] \geq 1, ch \in children\}$
 8  if $sig.children = \emptyset$ then                                /* CASE 1 */
 9    for $ch$ in $children$
10        $downNodes.LookUP[ch] \leftarrow downNodes.LookUP[ch] \oplus weights[ch]$
11        recomputeSig($ch$)
12    end
13    return
14  fi
15  for $ch$ in $sig.children$                                       /* CASE 2 */
16      for $w$ in upper.inducedGraph($x$)
17          $upNodes.LookUP[w] \leftarrow upNodes.LookUP[w] \oplus \frac{1}{weights[ch]}$
18      end
19  end
20  computeTermSig($x$, ($children \setminus sig.children$))

function recomputeSig($x$)
 1  $itemsWeight \leftarrow downNodes.LookUP[x]$
 2  $nodeSig[x] \leftarrow$ WFisher.test(items($x$), $sigItems$, $itemsWeight$)
 3  return   $nodeSig$
```

have p-values greater than node x should be investigated after the new p-value of node x is recomputed (due to the updates in the weights). Two cases are considered.

In CASE I, lines 9-13, there are no active children that have a p-value smaller than node x, thus node x is a local optimum. One strategy here is to increase the p-value of the active children even more, since the aim is to find the nodes that best represent a particular area in the graph. Thus, we update the old weights of these children with the new weights. Given the DAG structure, a node for which weights need to be updated can already have weights assigned to it by another parent that was processed before node x. The assignment in line 10 is addressing this problem. Different vector operators $\oplus$ can be used here, for example a function that returns the minimum on the components or a function that returns the product on the components. After the weights are updated, the p-value of the child is recomputed.

This is done by the function recomputeSig($\mathtt{ch}$). Again, here we can use different strategies. A simple example is just to recompute the $\mathtt{p}$-value of node $\mathtt{ch}$. In a more general case, all nodes in lower.inducedGraph($\mathtt{ch}$) can be revisited and their $\mathtt{p}$-values can be adjusted. After the weights for all the active children are updated, the processing of node $\mathtt{x}$ is finished.

In CASE II, there are some children that are more significant than node $\mathtt{x}$, thus these children represent local optima. Similar to the approach taken in the elim algorithm we want to weight the items in upper.inducedGraph($\mathtt{x}$). By doing this, their items will count less when the ancestors of node $\mathtt{x}$, including node $\mathtt{x}$, are scored, thus increasing those $\mathtt{p}$-values. Observe that we do not weight the ancestors of the processed child $\mathtt{ch}$, but the ancestors of node $\mathtt{x}$, line 15. We do not know what the relation between the $\mathtt{p}$-values of node $\mathtt{ch}$ and a parent of node is. The operator $\oplus$ from line 17 is similar to the one from line 10. The difference is that the two sets on which it operates have different cardinality. After all nodes from $\mathtt{sig.children}$ are processed, the items' weights of node $\mathtt{x}$ have been updated.

At this point the nodes processed in CASE II are removed from the list of active nodes. Note that the newly active nodes are nodes that have not been processed. The function computeTermSig($\cdot, \cdot$) is called for these nodes in line 20.

The running time of the algorithm depends on the function recomputeSig($\mathtt{ch}$). With the function presented in Algorithm 5, which requires constant time, the running time of the weight algorithm is quadratic in the worst case. This is the case because the updating of the weights described in lines 15-19 can be done after having finished processing node $\mathtt{x}$.

Chapter 5

Implementation and experiments

In this chapter we give some implementation details necessary for a better understanding of the experiments. The evaluation of the algorithms consists of a study based on real data and a study based on simulated data. For both studies the advantages and the disadvantages of each algorithm are discussed. We also present some ideas regarding the visualization of the results.

The algorithms were implemented using the R programing language (Gentleman and Ihaka, 1996). This choose is motivated by the large amount of statistical and visualization software packages available for this language. On the other hand, the Bioconductor Project[1] provides software infrastructure for working with biological data.

5.1 The ALL dataset

For running our methods on a real microarray expression data we choose **ALL** (Acute Lymphoblastic Leukemia) dataset from the Ritz Laboratory.

This dataset was extensively studied in (Golub et al., 1999; Chiaretti et al., 2004; von Heydebreck et Gentleman, 2004c), thus we can compare our results with the results published. The dataset is available in R, as a Bioconductor package.

The dataset was introduced in (Chiaretti et al., 2004). The gene expression matrix is already normalized with quantile normalization, and the expression estimates are computed using RMA (Irizarry et al., 2003). The data is presented in the form of an 'exprSet' object.

The **ALL** data consists of microarrays from 128 different patients with Acute Lymphoblastic Leukemia. On each microarray there are 12625 probes. Different probes can code for the same gene. This many-to-one mapping of the probes to gene identifiers, in our case LocusLink identifiers, needs in general some form of adjustment for a correct inference. On the other side probes that code for the same gene can measure different things. Thus, it is not clear if the duplicates should be removed. For our experiments we choose not to remove the duplicates. Given this, in the rest of this chapter we refer to the probes as genes.

[1]www.bioconductor.org

Differentially expressed genes. The first step in analyzing the data is to determine the differentially expressed genes. We showed in Section 2.2.1 that this can be achieved with a two sided t-test. We choose two ways for discriminating between patients.

Setup 1: It is known that the ALL cells are delivered from either B-cell or T-cell precursors. We split the patients according to the type and stage of the disease: There are 95 patients with B-cell ALL and 33 patients with T-cell ALL.

Applying a two sided t-test for these groups, we obtain the raw p-values. As discussed in Section 2.2.2 we need to account for multiple testing. Adjusting the p-values using the Bonferroni method we obtained 540 differentially expressed genes for a level $\alpha = 0.01$ test. Controlling the false discovery rate with the method from (Benjamini and Yekutieli, 2001) gives us 915 differentially expressed genes.

Setup 2: A more interesting study is the comparison within B-cell ALL patients. From 95 B-cell ALL patients we can find 37 patients with BCR/ABL (fusion gene that result from a translocation of the chromosomes 9 and 22) and 42 patients that are cytogenetically normal, the NEG group.

Using the Bonferroni method to adjust the raw p-values obtained with the two sided t-test, 10 differentially expressed genes are obtained for a level $\alpha = 0.01$ test. For a level $\alpha = 0.05$ we obtain 9 additional genes. Using the (Benjamini and Yekutieli, 2001) adjustment method for controlling the false discovery rate we obtain 16 differentially expressed genes for a level $\alpha = 0.01$ and 28 differentially expressed genes for $\alpha = 0.05$.

The list of differentially expressed genes will be referred to as the list of interesting genes in the rest of this chapter.

For mapping the gene set from the **ALL** data we need the annotation for the Affymetrix chip HGU95aV2, that was used for these studies (Chiaretti et al., 2004). The *hgu95av2* Bioconductor package provides information about the genes found on this chip. It includes LocusLink identifiers, gene names, Unigene cluster identifiers, pathway associations and also information about the GO classification. All the mappings can be found using the function hgu95av2() from this package.

The mappings between the genes and the corresponding GO term (GO identifier) are stored in the hgu95av2GO environment. Each gene is mapped to a list containing one or more GO terms. Each such element (if the list is not empty) contains a sublist with three named elements:

GOID: gives the Gene Ontology term

Ontology: is one of the three biological areas: *MF* for molecular function, *BP* for biological process and *CC* for cellular component.

Evidence: contains the evidence code that supports the association of the GO term to the gene found on the array, see Section 2.3.1.

5.2 Implementation

In this section we give short implementation details. We focus on preprocessing steps and visualization issues. The preprocessing of the data can be summarized in the following steps.

Step 1: Selecting the ontology. The first step is the selection of the ontology to be used. As discussed in Section 2.3.1 there are three distinct ontologies: molecular function (MF), biological process (BP) and cellular component (CC). The experiments in this chapter were done for the BP ontology.

The Bioconductor package *GO* (Gentleman, 2004a) provides these ontologies. All the GO terms from the selected ontology are retrieved for further use.

Step 2: Obtaining the list of all genes. The microarray dataset is processed here. In the case of the **ALL** dataset all the data are stored into an *exprSet* R object. For the use of other datasets first such an object must be formed before further analysis.

The genes found on the microarray are selected. Using Bioconductor packages different filters can be applied to reduce the set of genes. Also in this step the multiplicities between microarray probes and gene identifiers can be removed if required. After the list of genes to be analyzed is set, the mapping to the GO terms is performed, see Section 2.3.1. The genes that can not be mapped to the available GO terms are discarded. This can happen either because there is no GO term in the selected ontology to which the respective gene can be annotated, or the ontologies are incomplete. In our studies we could map 9623 from a total of 12625 genes to the GO terms from the BP ontology. The list of genes obtained is the *list of all genes*.

Step 3: Obtaining the list of interesting genes. Next the list of *interesting genes* is computed. This list can be given as a predefined list of genes or can be computed from the microarray data. For the experiments involving the **ALL** dataset the differentially expressed genes form the list of interesting genes, see Section 5.1. Different test statistics and different multiple testing correction for finding differentially expressed genes can be employed. We use the *multtest* package (Dudoit and Yang, 2003; Ge et al., 2003) from Bioconductor. The package also offers several types of analysis for determining the set of interesting genes.

Step 4: Building the DAG structure. The GO terms obtained by mapping all genes to the selected ontology are only the most specific GO terms. In Section 3.1 we argue that all GO terms with at least one annotated gene to them should be analyzed. Thus, we must map the genes to all ancestors of specific GO terms.

The *GO* package keeps the graph structure of the ontologies in environments (an environment is a data-structure similar to a list from a user point of view). This type of encoding the DAG structure is cumbersome if graph specific operations are needed. Another issue is the visualization of the graph topology. To overcome these problems we use the *graph* package from Bioconductor, see (Gentleman and Whalen, 2004). This package offers basic data-structures for graphs and basic functions for working with them. In addition, there are packages like *RBGL* and *Rgraphviz* which offer complex functionalities based on the graph package.

Starting from the most specific GO terms the DAG structure of the GO is constructed. All added nodes are ancestors of the most specific nodes with respect to the GO relations, see Section 2.3.1. Note that given the complexity of the GO DAG and the way the genes are annotated to the GO terms, the most specific GO terms are not necessarily leaves in this graph. The root node of this graph is the root of the specified ontology. For computational reasons two graphs are computed. The first graph has the edges directed from the root to the leaves. The second one has the edges directed from the leaves to the root. In this way accessing the parents and the children of a node is easier. At this point all the knowledge of the GO is embedded in these two graphs. Only these two graphs are used in the further analysis.

To map the genes to all GO terms (nodes), the genes from the leaves are *pushed up* the DAG to the ancestors of the specific nodes. After this step all the nodes in the DAG contain all the genes that can be mapped to them. For example the root of the DAG contains all genes.

After all these preprocessing steps, each of the algorithms presented in Chapter 4 can be used. Input for all algorithms are the list of interesting genes and the graphs. For each algorithm there is a list of parameters that need to be set.

Multiple functions for analyzing the obtained DAG and the result of the algorithms are available. For each node in the graph (each GO term) the counts of the mapped genes and of the interesting genes, the contingency table used for Fisher's exact test, the expected number of genes for the respected node and other information can be obtained.

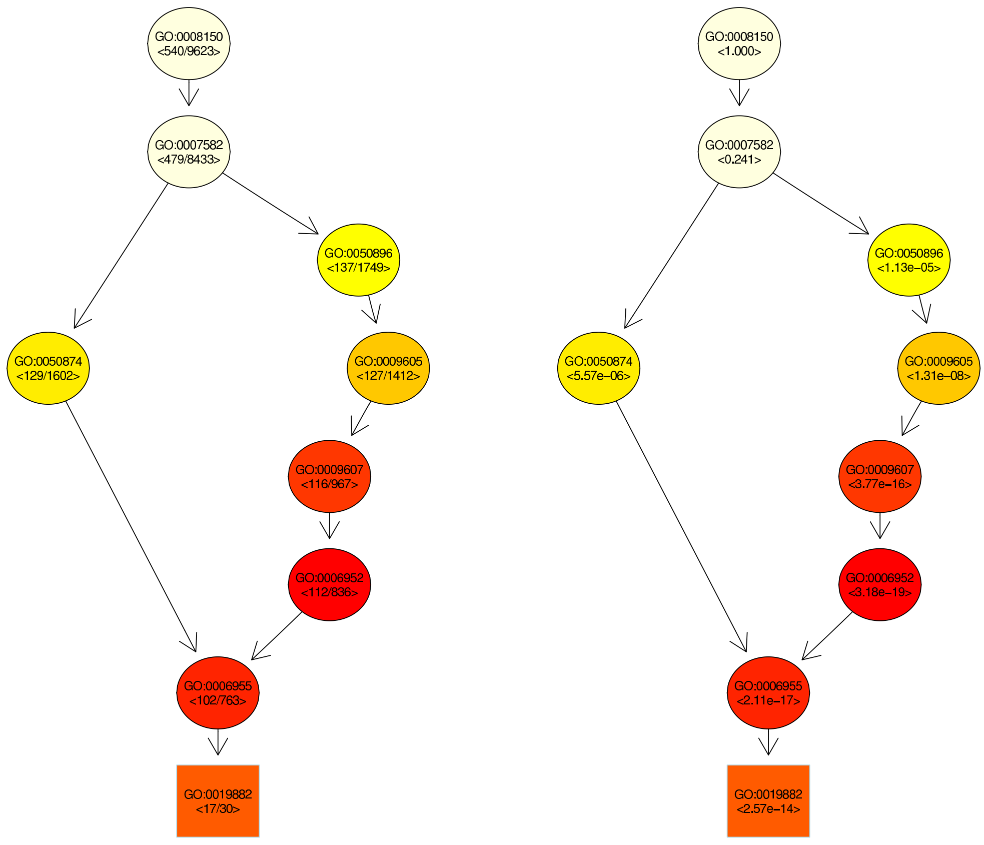
5.2.1 Visualizing graphs

One main tool for analyzing the results, is the visualization of the graph topology. To plot graphs we used the package *Rgraphviz* (Gentry, 2004) from Bioconductor. This package is an R interface for the AT&T GraphViz library[2], thus having all the facilities that the GraphViz library offers.

Since the GO DAG is large, comprising 2311 nodes and 3525 edges for the BP ontology and the genes from the **ALL** dataset, plotting only parts of it (subgraphs) is desired. Keeping the GO DAG structure into a *graph* object makes plotting of the DAG or parts of it easier. Of particular interest are the subgraphs induced by some significant nodes. We defined in Section 4.1 the $\mathsf{upper.inducedGraph}(\mathsf{U})$ and the $\mathsf{lower.inducedGraph}(\mathsf{U})$ for a list U of nodes. These subgraphs can easily be obtained with the functions from the *graph* package. By plotting these subgraphs more insight into node x can be obtained.

In Figure 5.1 we plotted the $\mathsf{upper.inducedGraph}(\text{GO:0019882})$. Different types of information can be displayed for each node. Since usually there is no room left to label the nodes of the DAG with all the available information, we plot the same graph twice, where each plot gives a particular type of information. In Figure 5.1 there are two types of node shapes. We plotted the node GO:0019882 using a box shape to emphasize it. Nodes that are of particular interest are plotted using a box shape or any other shape preferred by the user. The edges of the graph are colored with two colors. The black edges denote *'is a'* relationships between the nodes and the red edges denote *'part of'* relationships, see Section 2.3.1. In Figure 5.1 all edges represent an *'is a'* relationships.

[2]http://www.research.att.com/sw/tools/graphviz/



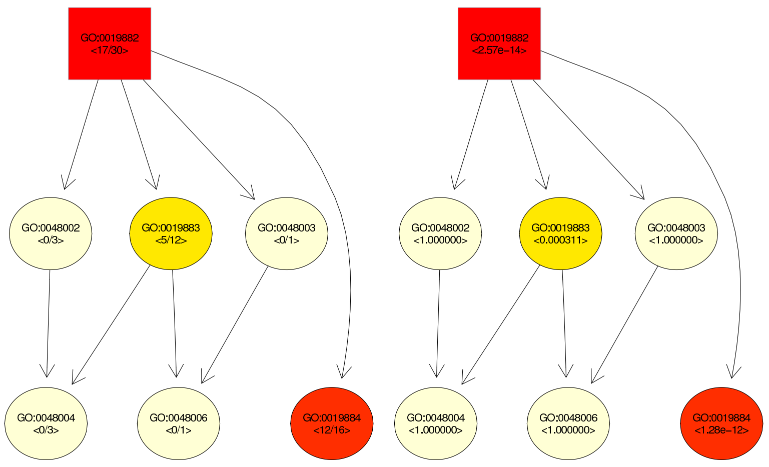
(a) Nodes with the count info (b) Nodes with the pval info

Figure 5.1: *Example of the subgraph induced by node GO:0019882. All paths from the root to this node are plotted. In figure (a) the counts for each node are displayed:* x/y *denotes that out of* y *genes mapped to the node,* x *belong to the list of interesting genes. In figure (b) the* p*-value of the node is displayed.*

To better understand the dependences between the nodes in the DAG, the nodes are colored with different intensities (from dark red to light yellow) depending on their p-value. A node with a low p-value will be colored dark red. The non-significant nodes, p-values close to 1, will be almost white. This type of coloring is useful for finding the interesting patterns in the DAG and detecting dependences induced by the parent-child relationships.

Figure 5.2 shows the lower.inducedGraph(GO:0019882). For a specified node it is possible to plot all ancestors not further than three levels and all children not further than two levels from this node, for example. Also all ancestors and children of a node or of a set of nodes can be plotted if required.

Another available type of labeling the nodes inspired from (Gentleman, 2004d) is shown in Figure 5.3. The nodes are plotted as pie charts showing the ratio of the observed interesting genes and the total number of genes mapped to a node. Similar to the other plots, the nodes that are of particular interest can be distinguished by the coloring.



(a) Nodes with the count info (b) Nodes with the pval info

Figure 5.2: *The lower induced subgraphs for node GO:0019882*

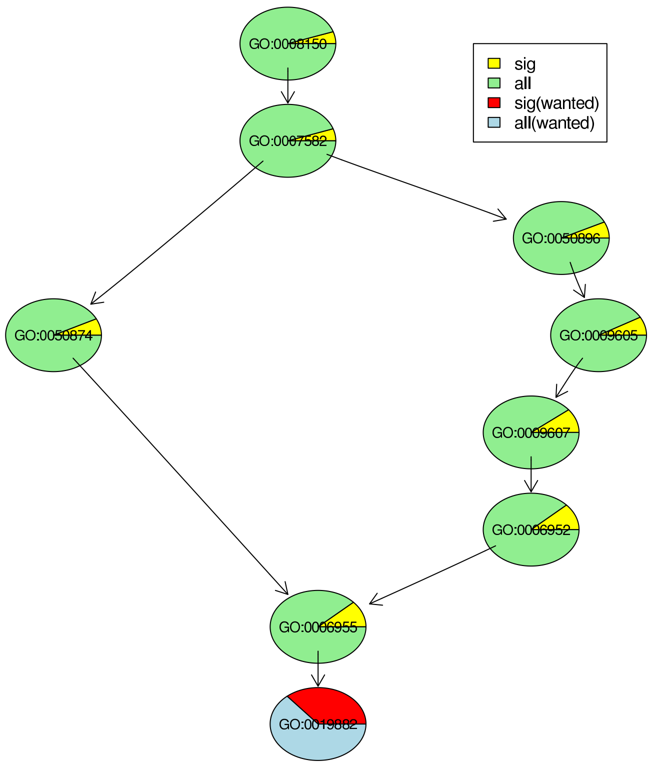


Figure 5.3: *Example of a pie plot*

We believe that such plots are improving the understanding of the analyzed data. In the next section we use these plots to discuss the results obtained by running the algorithms presented in Chapter 4 on the **ALL** data.

5.3 Comparing the algorithms on the ALL dataset

In this section we compare the results obtained by running the algorithms presented in Chapter 4 on the two setups described in Section 5.1. The preprocessing steps are performed as described in Section 5.2. Unless specified differently, the values stated there are the ones used.

For both setups we compare the following algorithms (Fisher's exact test is used for computing the p-values for each algorithm):

classic.f: This is the algorithm in which each GO term is tested independently of the others, see Algorithm 0.

topo: This is Algorithm 1. The $\mathtt{cutOff}$ parameter is set to 0.01.

elim: This is Algorithm 2. We use the same $\mathtt{cutOff}$ 0.01.

readjust: This is Algorithm 3. The p-values obtained with the classical algorithm are adjusted using the (Benjamini and Hochberg, 1995) false discovery rate procedure. The level of the test was set to 0.05.

weight: This is Algorithm 4. The algorithm is using the recomputeSig$(\cdot)$ procedure presented in Algorithm 5. The lists $upNodes.\mathsf{LookUP}$ and $downNodes.\mathsf{LookUP}$ refer to the same list. The choice of the weighting function is discussed later in the section.

classic.z: This is the same algorithm as classic.f, but a z-test is used instead of Fisher's exact test.

all.M: This is a combination of all the above algorithms except classic.z. As p-value the *mean* of the p-values of all other algorithms is computed using a log scale. If $p.val$ is the vector of all the p-values for a GO term, then the new p-value is defined by

$$\overline{p} = \exp(\overline{\log(p.val)}).$$

5.3.1 Setup 1: Comparison of B-cell ALL and T-cell ALL.

For the first test we choose as interesting genes the first 540 genes ordered by the t-test p-values, see Section 5.1. The ontology used is the BP ontology, see Section 2.3.1. For this ontology we obtain 2311 GO terms that must be investigated.

The p-values reported for all algorithm are adjusted p-values using the (Benjamini and Yekutieli, 2001 false discovery rate procedure. The results are presented in Table 5.1.

The GO terms shown in Table 5.1 are sorted by their p-values computed with the classic.f algorithm. Since a multiple testing adjustment is employed, we pick all the GO terms with an adjusted p-value less than 0.05. Table 5.2 shows the number of significant GO terms for different level α tests.

Table 5.3 gives more information on the significant GO terms, namely the name of the GO term, the total number of genes that can be mapped to the GO term (column 'Annotated') and the number of interesting genes mapped to the GO term (column 'Observed'). To obtain

	GO ID	classic.f	topo	elim	readjust	weight	classic.z	all.M
1	GO:0006952	6.1e−15	0.572	1.000	0.9	1.0e−11	2.6e−21	1.5e−05
2	GO:0006955	2.0e−13	4.1e−13	5.9e−09	4.4e−08	9.3e−09	4.7e−19	3.2e−10
3	GO:0009607	2.4e−12	3.6e−12	1.000	1.0	9.3e−07	1.3e−16	1.9e−05
4	GO:0019882	1.2e−10	0.572	0.647	1.0	2.5e−10	4.4e−30	0.00062
5	GO:0030333	4.2e−10	0.572	0.647	1.0	3.5e−10	4.2e−28	0.00083
6	GO:0019884	4.1e−09	8.2e−09	1.2e−08	2.5e−08	3.0e−06	7.4e−30	4.6e−08
7	GO:0019886	3.2e−08	5.7e−08	7.6e−08	7.6e−08	9.9e−05	4.7e−26	3.8e−07
8	GO:0009605	3.2e−05	1.000	1.000	1.0	0.0020	2.4e−06	0.92887
9	GO:0050874	0.012	1.000	1.000	1.0	0.0071	0.0020	1.00000
10	GO:0016126	0.019	0.036	0.047	1.0	0.0187	4.4e−07	0.11467
11	GO:0050896	0.020	0.036	1.000	1.0	0.0726	0.0041	0.87163

Table 5.1: *The significant GO terms at a level* $\alpha = 0.05$.

level α	classic.f	topo	elim	readjust	weight	classic.z	all.M
0.01	8	4	3	3	9	68	7
0.05	11	6	4	3	10	83	7
0.1	12	6	4	3	11	90	7

Table 5.2: *The number of significant GO terms for different levels of the test.*

a better idea of how many interesting genes should be mapped to a GO term in a random case, we compute this number in the column 'Expected', as

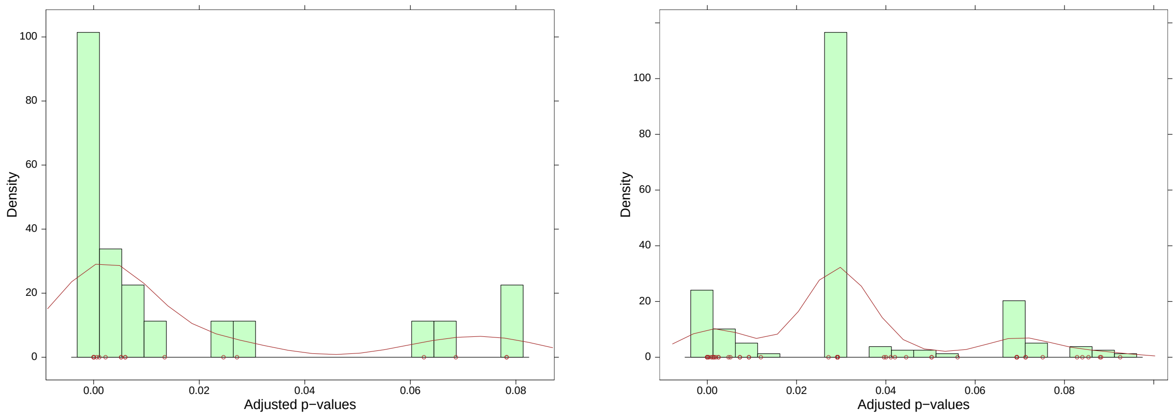
$$\text{expected} = \#\text{all mapped} \times \frac{\#\text{all interesting}}{\#\text{all genes}}.$$

	GO ID	Term	Observed	Expected	Annotated
1	GO:0006952	defense response	112	46.913	836
2	GO:0006955	immune response	102	42.816	763
3	GO:0009607	response to biotic stimul...	116	54.264	967
4	GO:0019882	antigen presentation	17	1.683	30
5	GO:0030333	antigen processing	17	1.796	32
6	GO:0019884	antigen presentation, exo...	12	0.898	16
7	GO:0019886	antigen processing, exoge...	12	1.01	18
8	GO:0009605	response to external stim...	127	79.235	1412
9	GO:0050874	organismal physiological ...	129	89.897	1602
10	GO:0016126	sterol biosynthesis	9	1.515	27
11	GO:0050896	response to stimulus	137	98.146	1749

Table 5.3: *Some statistics for the significant GO terms*

We can see that the results of the algorithms are quite different. Some GO terms that are found significant by the classical method are completely discarded by other methods. For example GO:0006952 is reported as non-significant by the methods topo, elim and readjust, but it is significant for the others. Another example is the term GO:0050874 which is found significant only by the algorithms classic.f and weight. Even the z-test for this term is not 'so significant', considering the number of GO terms with a p-value less than 0.05, see Table 5.2.

The GO terms GO:0019884 and GO:0019886 which are reported significant by all algorithms are leaves in the GO DAG. They are the smallest GO terms w.r.t. the number of genes mapped to them, see Table 5.3. In fact, for this setup the GO terms found to be significant have considerably large numbers of mapped genes. Among all GO terms with an adjusted p-value smaller than 1, the GO term GO:0009595 containing 11 genes has the smallest number of genes. Thus the issue regarding a minimal necessary size for GO terms raised in (Gentleman, 2004c) is not relevant in this case.



(a) classic.f algorithm (b) classic.z algorithm

Figure 5.4: *The distribution of the* p*-values without taking into account the topology: Fisher's exact test (a) and* z*-test (b).*

By looking at the p-values and the number of significant GO terms for a specified level α we see that there is a significant difference between Fisher's exact test and the z-test. Figure 5.4 shows the distribution of the adjusted p-values for these two tests.

For the z-test 68 GO terms have a p-value below 0.01, which is a relatively high number of terms especially after p-value adjustment. Moreover, we see that specific terms like GO:0019884 and GO:0019886, are very significant. Figure 5.6 better illustrates this behavior. Some GO terms are found significant by a z-score but not by Fisher's exact test. For example node GO:0009164 is ranked 34 with an adjusted p-value of 1 by classic.f and is ranked 10 with an adjusted p-vale of 6.36e-06 by classic.z.

	classic.f	topo	elim	readjust	weight	classic.z	all.M
classic.f	1.000	0.920	0.830	0.602	0.814	0.502	0.769
topo	0.920	1.000	0.908	0.697	0.688	0.474	0.756
elim	0.830	0.908	1.000	0.785	0.565	0.439	0.688
readjust	0.602	0.697	0.785	1.000	0.568	0.289	0.691
weight	0.814	0.688	0.565	0.568	1.000	0.349	0.879
classic.z	0.502	0.474	0.439	0.289	0.349	1.000	0.338
all.M	0.769	0.756	0.688	0.691	0.879	0.338	1.000

Table 5.4: *The correlation between the resulted* p*-values.*

Table 5.4 gives the correlation between adjusted p-values for all methods. We see that the classic.z method is less correlated with the others. This behavior is partially explained by the values in Table 5.2; the adjusted p-values reported by a z-test are lower than the ones obtained with Fisher's exact test.

There are a few issues with the results from Table 5.4. A large number (almost 90%) of GO terms have an adjusted p-value equal to 1. Even for the raw p-values, more than half of the GO terms have a p-value equal to 1. Thus taking into account these GO terms can be misleading. Another issue is that we are more interested in the order in which important GO terms are reported by each method. To overcome these issues we compute (Table 5.5) a rank correlation on a subset of interesting GO terms, see (Lehmann, 1986) for more details on rank correlation.

	classic.f	topo	elim	readjust	weight	classic.z	all.M
classic.f	1.000	0.374	0.156	−0.070	−0.036	0.802	0.979
topo	0.374	1.000	0.677	0.292	−0.073	0.339	0.362
elim	0.156	0.677	1.000	0.202	−0.105	0.191	0.131
readjust	−0.070	0.292	0.202	1.000	−0.002	0.066	−0.080
weight	−0.036	−0.073	−0.105	−0.002	1.000	0.037	0.052
classic.z	0.802	0.339	0.191	0.066	0.037	1.000	0.722
all.M	0.979	0.362	0.131	−0.080	0.052	0.722	1.000

Table 5.5: *Rank correlation for a sample of significant GO terms.*

The subset of GO terms was compiled in the following way. For each method we retrieve the 100 most significant GO terms. Then we define the union set of all resulting GO terms. In our case we obtain a set of 147 distinct GO terms. For these terms we retrieve the raw p-values assigned by each method. The result is a matrix with 7 columns, one column for each method, and 147 rows. The rank correlation for this matrix is shown in Table 5.5.

The first observation is that on this subset of GO terms the weight algorithm is almost uncorrelated with all other methods. Especially there is no correlation with all.M, the average of all the methods that use Fisher's exact test. This contrasts with the result from Table 5.4 in which these two methods are highly correlated. Similar behavior is found for readjust, but in this case there is some correlation with topo and elim. Finally we note that there is a strong correlation between the z-test and Fisher's exact test and the most correlated method with classic.f is all.M, for this set of GO terms.

The weighting function used in the weight algorithm is the ratio of the log of the p-values. If a and b are two nodes in the DAG then we have

$$\mathsf{sigRatioL}(a, b) = \frac{\log(\mathtt{sig}(a))}{\log(\mathtt{sig}(b))}.$$

Beside this weighting function we test two other versions. The first is the simple ratio defined by

$$\mathsf{sigRatio}(a, b) = \frac{\mathtt{sig}(b)}{\mathtt{sig}(a)}.$$

The reason of dividing the significance of node b by the significance of node a is explained in Section 4.3.2. The second function is a variant of $\mathsf{sigRatio}(a, b)$ in which the significance values are considered equal if their difference is smaller than a predefined value ϵ. Here, we use as threshold $\epsilon = 10^{-4}$. We call this weighting function sigRatio.weak.

	GO ID	classic.f	weight.log	weight.ratio	weight.weak
1	GO:0006952	6.1e−15	1.0e−11	5.4e−12	5.4e−12
2	GO:0006955	2.0e−13	9.3e−09	1.000	1.6e−10
3	GO:0009607	2.4e−12	9.3e−07	1.000	1.4e−08
4	GO:0019882	1.2e−10	2.5e−10	5.9e−08	1.6e−10
5	GO:0030333	4.2e−10	3.5e−10	0.757	5.3e−10
6	GO:0019884	4.1e−09	3.0e−06	1.000	4.9e−09
7	GO:0019886	3.2e−08	9.9e−05	1.000	3.2e−08
8	GO:0009605	3.2e−05	0.0020	1.000	0.026
9	GO:0050874	0.012	0.0071	1.000	1.000
10	GO:0016126	0.019	0.0187	0.062	0.023
11	GO:0050896	0.020	0.0726	1.000	1.000

Table 5.6: *The* p*-values for the three different weighting functions.*

The function that penalizes less is sigRatioL. For example, if the p-value for node a is 10^{-3} and the p-value of node b is 10^{-12}, then the weight given by sigRatioL is equal to 0.25 and the weight given by the other two functions is 10^{-9}. If the significance of node a is 10^{-5}, and thus below the threshold $\epsilon = 10^{-4}$, then the sigRatio is 10^{-7} but the sigRatio.weak score gives 1. We can see that the weight method using sigRatio weighting function is somehow close to the elim method.

In Table 5.6 the adjusted p-values for the weight method using the three weighting functions are presented. We see that for the sigRatio function we obtain very few significant GO terms, a behavior similar as with the elim method. Table 5.7 emphasizes this by showing the number of significant GO terms for different cutoffs.

level α	weight.log	weight.ratio	weight.weak
0.01	9	2	7
0.05	10	2	9
0.1	11	3	11

Table 5.7: *The number of significant GO terms for different cutoffs.*

The correlations between all versions of weight algorithm are shown in Table 5.8. The rank correlation is computed for a sample of GO terms compiled in the same way as in Table 5.5. In this case we obtain a total of 138 GO terms. We see that the method that is most correlated with the elim method is the weight method with sigRatio weighting function.

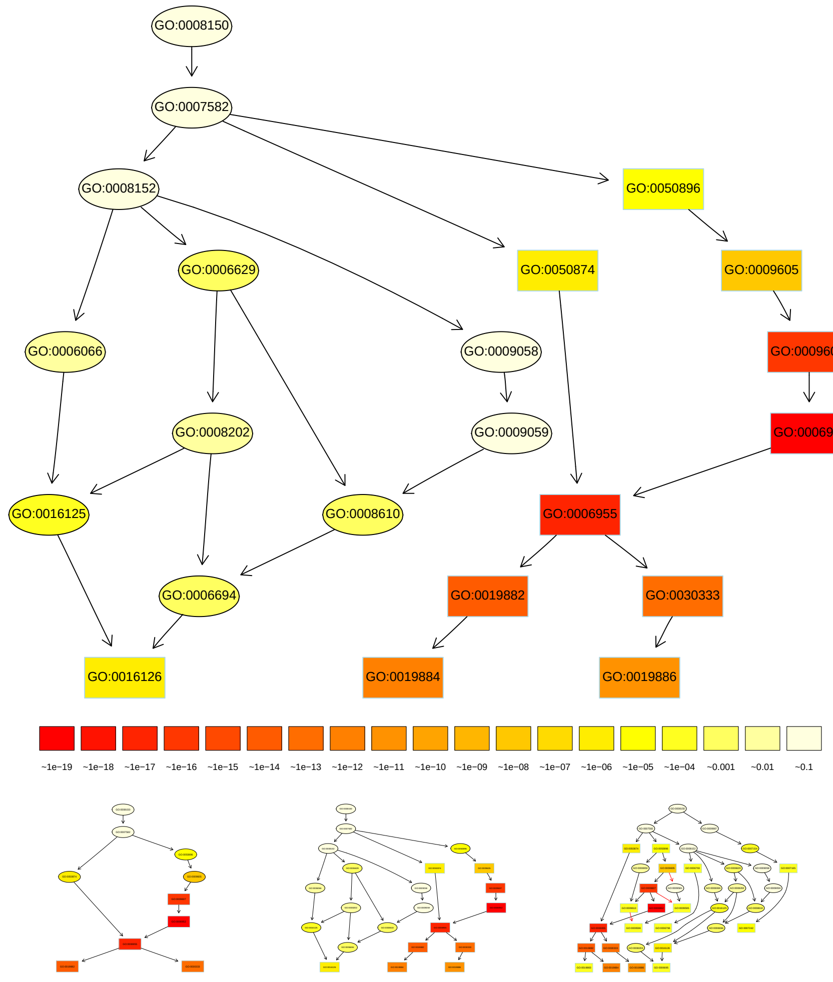
We see that different behavior of the weight algorithm can be achieved when using different weighting functions.

Next we investigate how the significant GO terms are distributed over the GO DAG. For this we plot for each algorithm mentioned above the subgraph induced by the most significant GO terms, see Section 5.2.1. Since the subgraphs have around 40 nodes, in the following figures, for the sake of reading clarity, we display only the GO ID of a node. The graphs presented below can be easily reproduced if there is a need to add more information to the nodes label.

	classic.f	elim	weight.log	weight.ratio	weight.weak
classic.f	1.000	0.310	0.226	−0.102	−0.126
elim	0.310	1.000	−0.006	0.388	0.334
weight.log	0.226	−0.006	1.000	0.462	0.512
weight.ratio	−0.102	0.388	0.462	1.000	0.799
weight.weak	−0.126	0.334	0.512	0.799	1.000

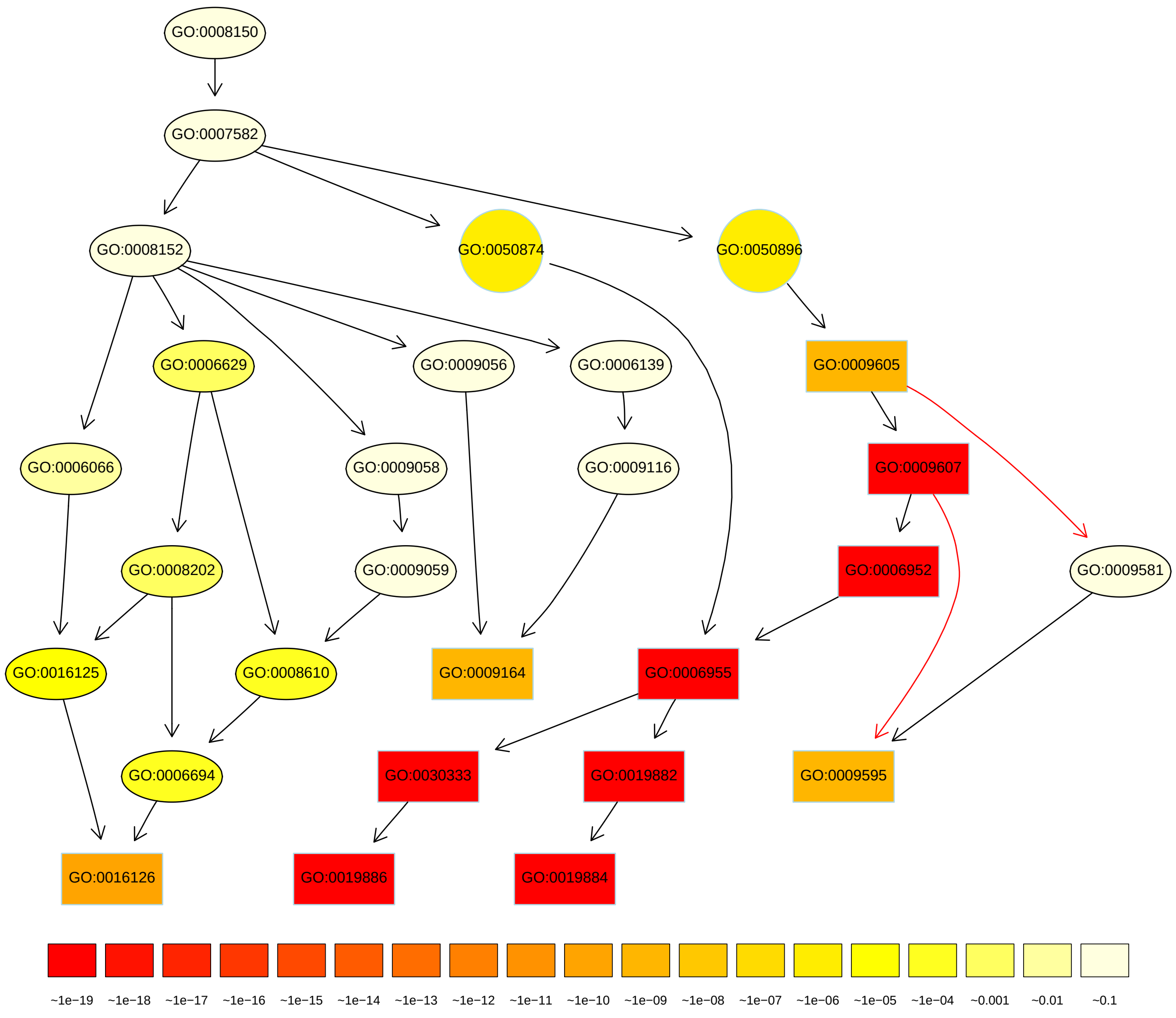
Table 5.8: *Rank correlation for a sample of significant GO terms between different settings of the* weight *algorithm.*

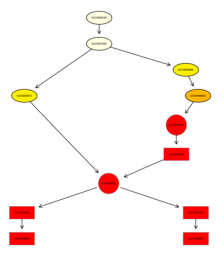
The significant nodes are represented as boxes in the plots. To better emphasize the differences between all methods and classic.f we plot the nodes that are found significant by classic.f but not by the current method as circles.

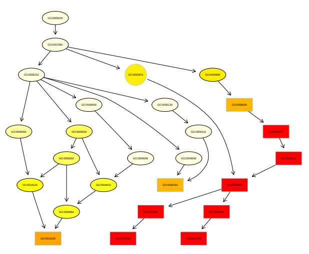


(a) top 5 (b) top 10 (c) top 20

Figure 5.5: *The subgraph induced by the most significant 11 GO terms found by method* classic.f. *This subgraph contains 22 nodes. There are 10 nodes in figure (a), 22 nodes in figure (b) and 36 in figure (c).*




(a) top 5


(b) top 10

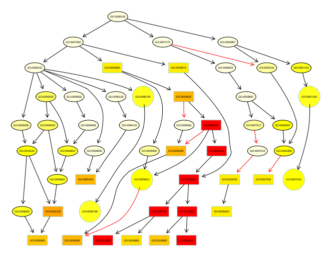
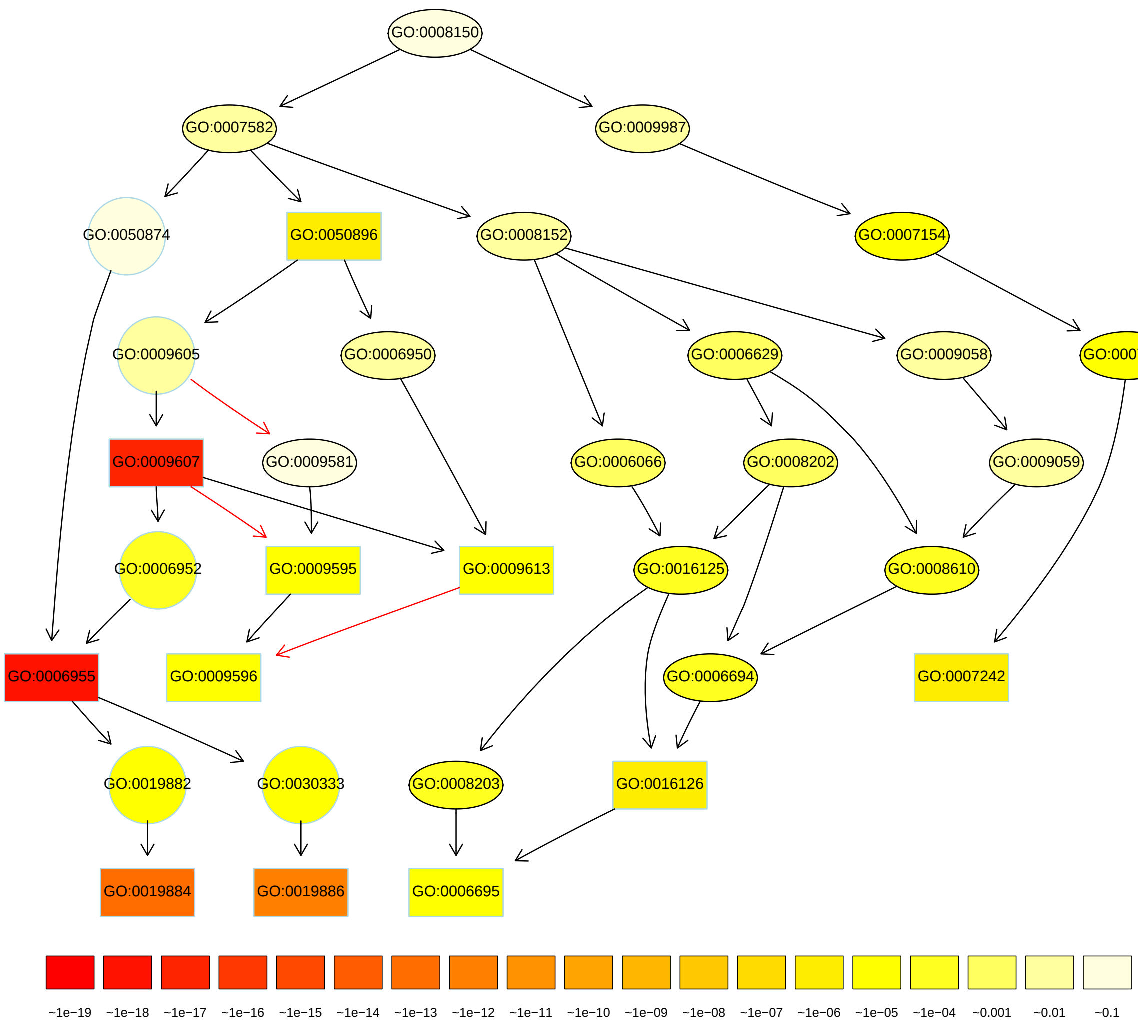
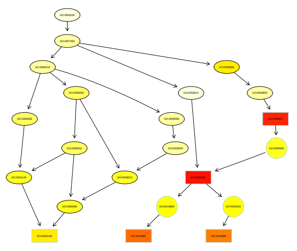
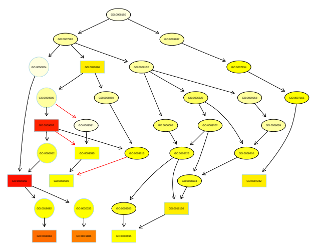

(c) top 20

Figure 5.6: *The subgraph induced by the most significant 11 GO terms found by method* classic.z*. This subgraph contains 28 nodes. There are 12 nodes in figure (a), 26 nodes in figure (b) and 52 in figure (c).*




(a) top 5


(b) top 10

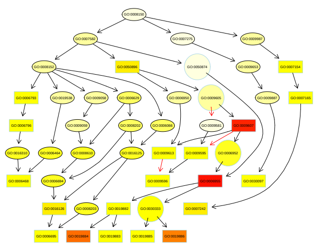
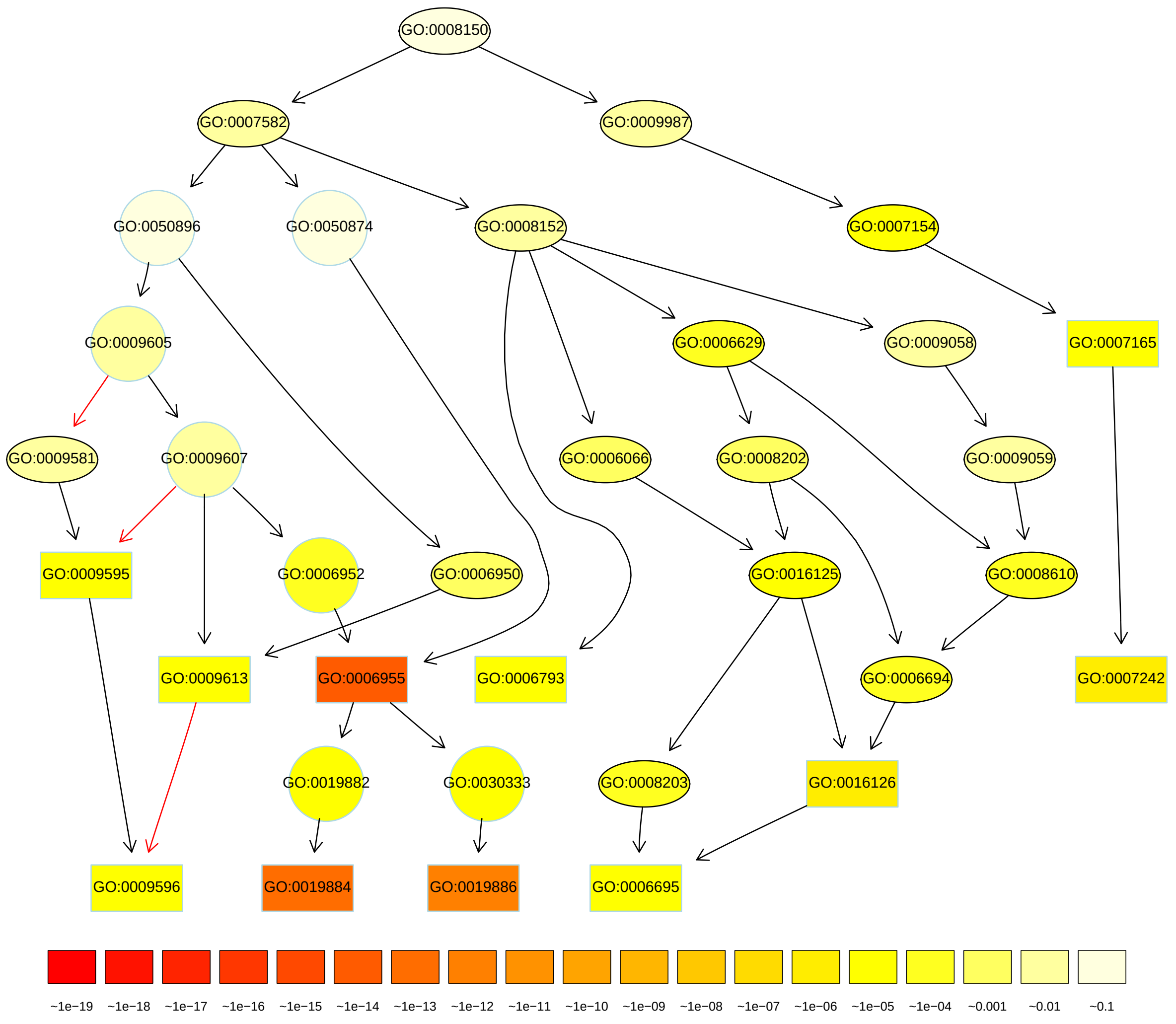
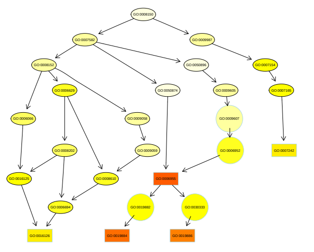
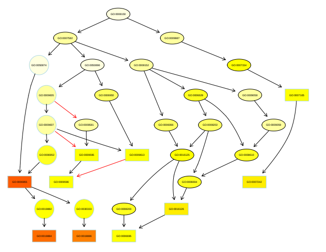

(c) top 20

Figure 5.7: *The subgraph induced by the most significant 11 GO terms found by method* topo. *This subgraph contains 33 nodes. There are 22 nodes in figure (a), 33 nodes in figure (b) and 45 in figure (c).*




(a) top 5


(b) top 10

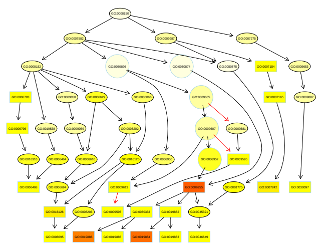
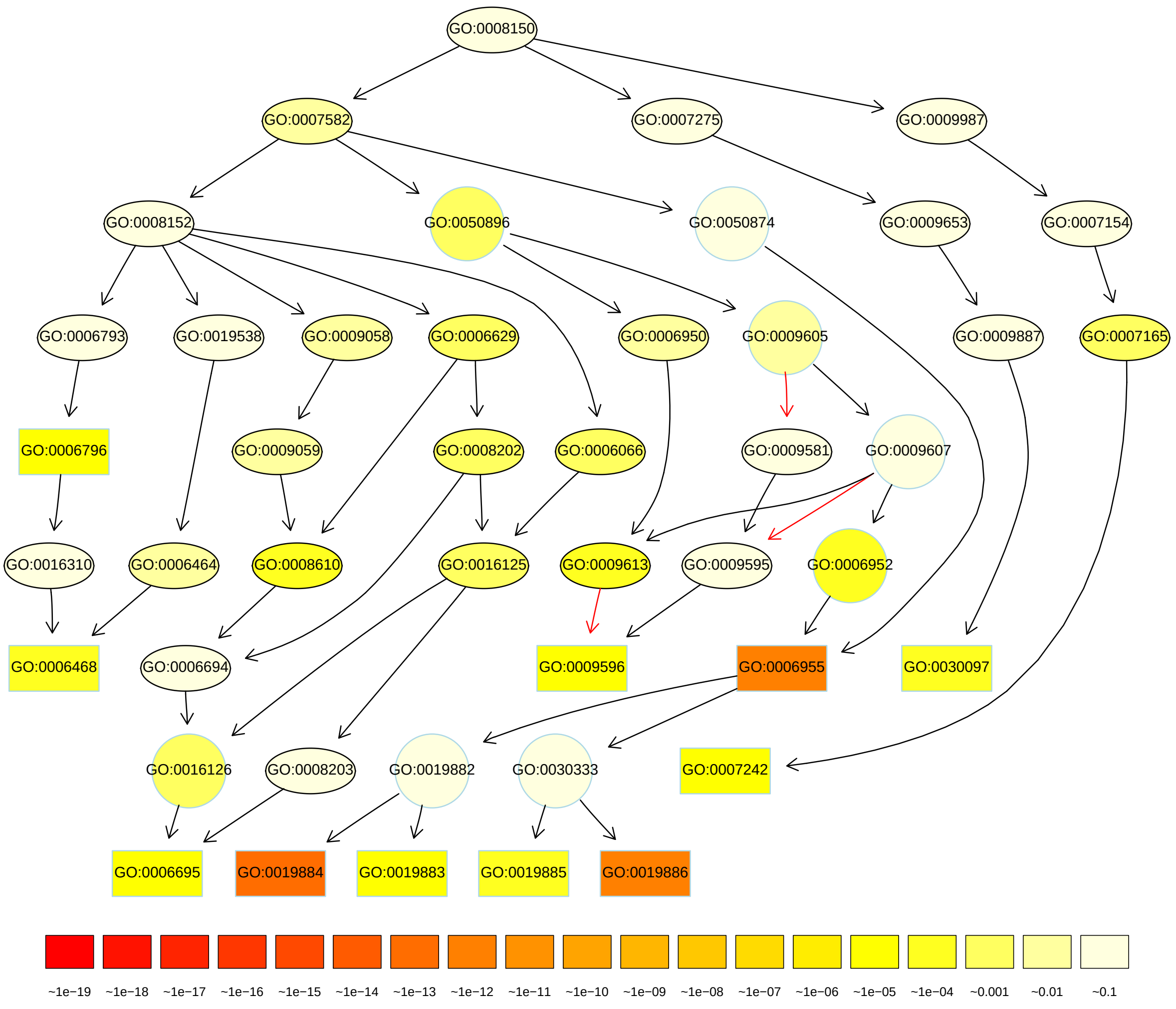
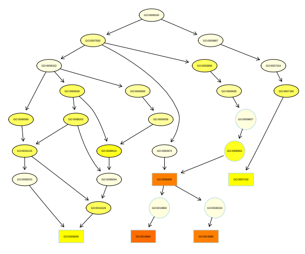
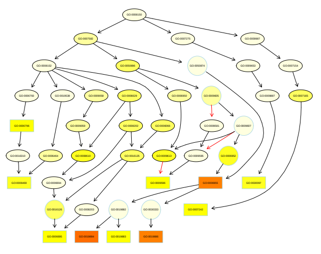

(c) top 20

Figure 5.8: *The subgraph induced by the most significant 11 GO terms found by method* elim. *This subgraph contains 34 nodes. There are 26 nodes in figure (a), 33 nodes in figure (b) and 49 in figure (c).*

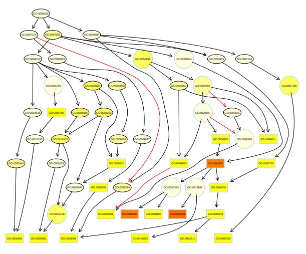




(a) top 5

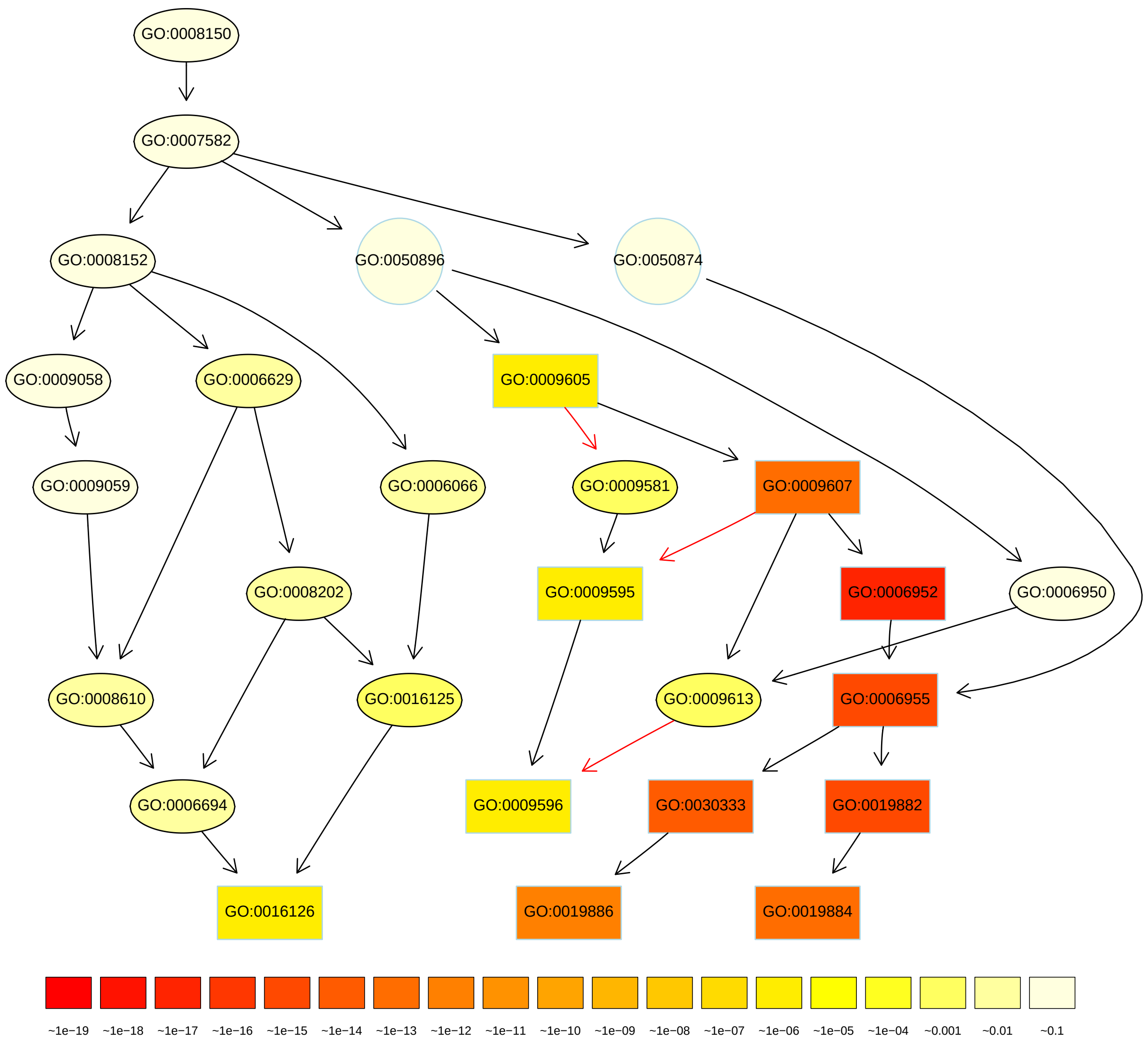


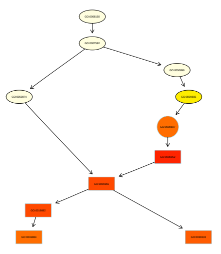
(b) top 10

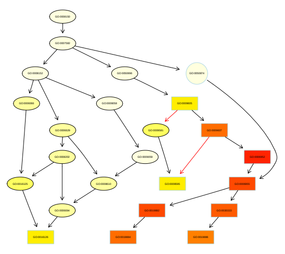


(c) top 20

Figure 5.9: *The subgraph induced by the most significant 11 GO terms found by method* readjust. *This subgraph contains 45 nodes. There are 28 nodes in figure (a), 44 nodes in figure (b) and 53 in figure (c).*




(a) top 5


(b) top 10

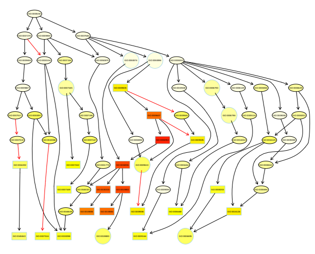
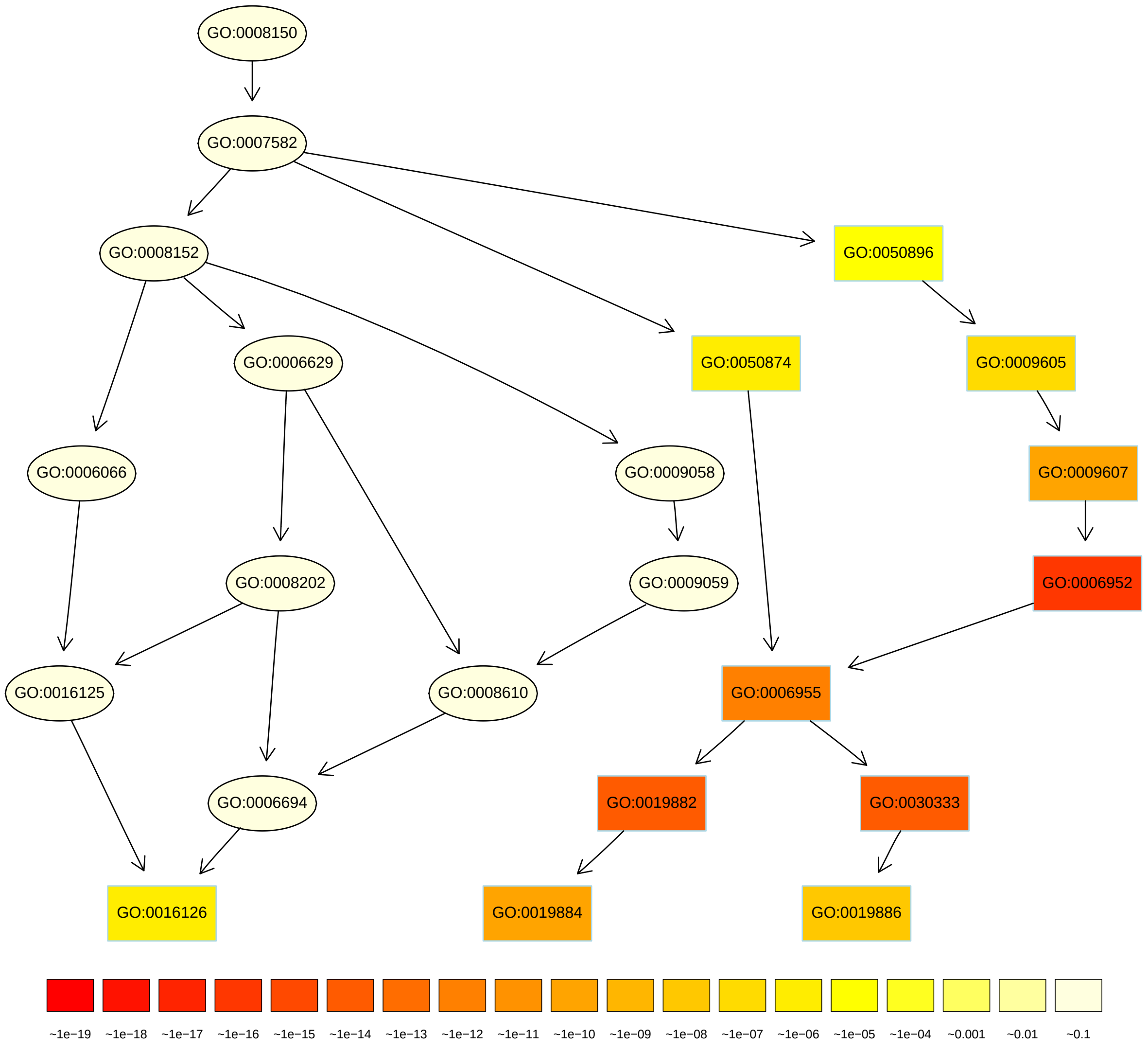
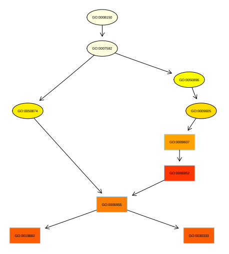
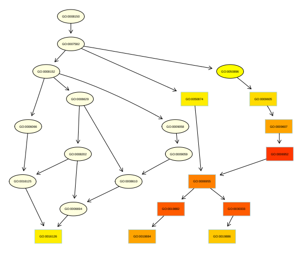

(c) top 20

Figure 5.10: *The subgraph induced by the most significant 11 GO terms found by method* weight. *The weighting function used is* sigRatio.weak( ). *This subgraph contains 24 nodes. There are 11 nodes in figure (a), 24 nodes in figure (b) and 63 in figure (c).*




(a) top 5


(b) top 10

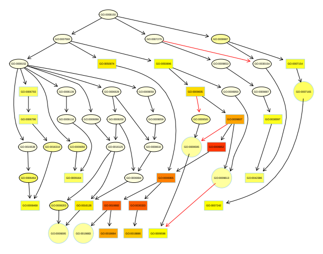
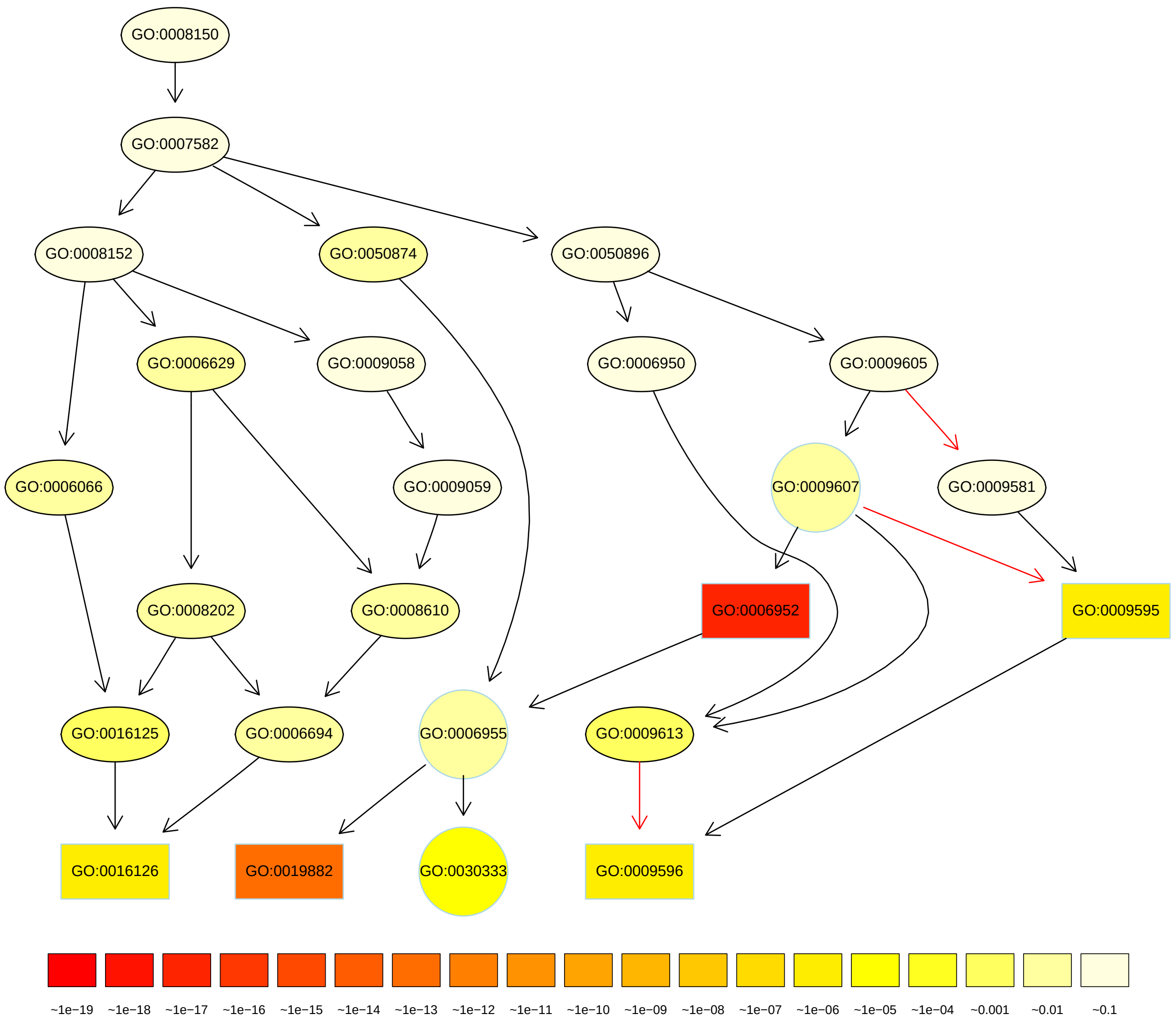
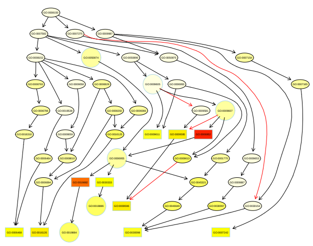
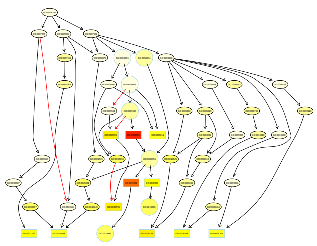

(c) top 20

Figure 5.11: *The subgraph induced by the most significant 11 GO terms found by method* weight*. The weighting function used is* sigRatioL( )*. This subgraph contains 22 nodes. There are 10 nodes in figure (a), 22 nodes in figure (b) and 50 in figure (c).*




(a) top 10


(b) top 11

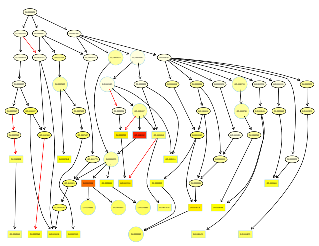
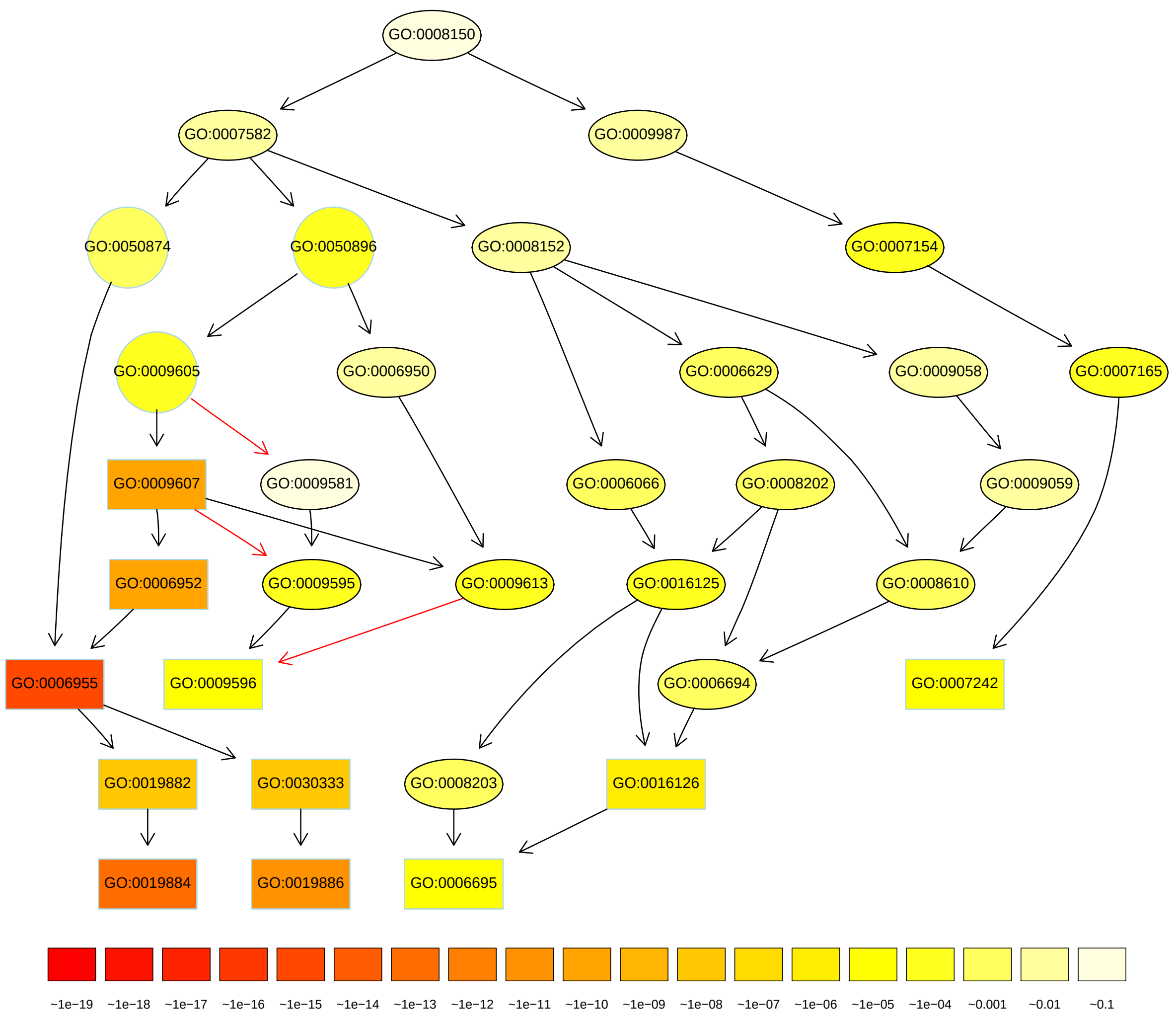
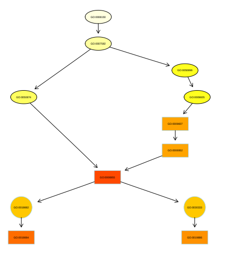
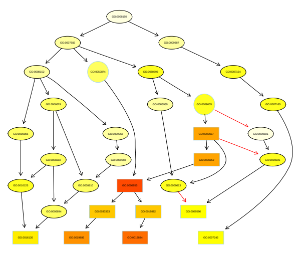

(c) top 20

Figure 5.12: *The subgraph induced by the most significant 5 GO terms found by method* weight*. The weighting function used is* sigRatio( )*. This subgraph contains 25 nodes. There are 48 nodes in figure (a), 52 nodes in figure (b) and 69 in figure (c).*




(a) top 5


(b) top 10

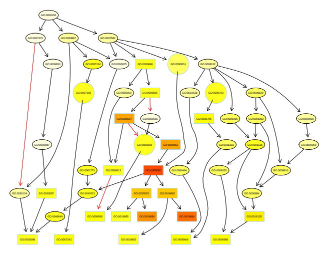

(c) top 20

Figure 5.13: *The subgraph induced by the most significant 11 GO terms found by method* all.M. *This subgraph contains 31 nodes. There are 12 nodes in figure (a), 31 nodes in figure (b) and 51 in figure (c).*

5.3.2 Setup 2: Comparison within B-cell ALL

In this setup the list of interesting genes contains 28 genes, see Section 5.1. We are interested in finding the significant biological processes and thus we used the BP ontology.

As in Setup 1, the p-values reported by each algorithm are adjusted using the (Benjamini and Yekutiel false discovery rate procedure. The results are presented in Table 5.9.

	GO ID	classic.f	topo	elim	readjust	weight	classic.z	all.M
1	GO:0000115	0.024	0.024	0.024	0.024	0.024	$< 1e-30$	0.024
2	GO:0000084	0.025	1.000	1.000	1.000	0.025	$< 1e-30$	1.000
3	GO:0008630	0.040	0.060	0.060	0.060	0.040	$< 1e-30$	0.060
4	GO:0008629	0.206	0.275	0.275	1.000	0.136	1.2e−25	1.000
5	GO:0007612	0.273	0.312	0.312	0.521	0.273	$< 1e-30$	0.521
6	GO:0042770	0.273	0.312	0.312	1.000	0.273	4.7e−18	1.000
7	GO:0006298	0.273	0.312	0.312	0.546	0.273	4.7e−18	0.546
8	GO:0045005	0.273	0.312	0.312	1.000	0.273	4.7e−18	1.000
9	GO:0007165	0.853	0.960	0.961	1.000	1.000	0.041	1.000

Table 5.9: *The GO terms with an adjusted* p*-value less than 1.*

	GO ID	Term	Observed	Expected	Annotated
1	GO:0000115	S−phase−specific transcri...	3	0.023	8
2	GO:0000084	S phase of mitotic cell c...	3	0.029	10
3	GO:0008630	DNA damage response, sign...	3	0.038	13
4	GO:0008629	induction of apoptosis by...	3	0.07	24
5	GO:0007612	learning	2	0.015	5
6	GO:0042770	DNA damage response, sign...	3	0.096	33
7	GO:0006298	mismatch repair	3	0.096	33
8	GO:0045005	maintenance of fidelity d...	3	0.096	33
9	GO:0007165	signal transduction	16	7.196	2473

Table 5.10: *Some statistics for the significant GO terms*

We note that in this case the adjusted p-values are not as significant as the one obtained for Setup 1. In Table 5.11 the number of significant GO terms for different cutoffs are shown. For this setup the conservative (Benjamini and Yekutieli, 2001) false discovery rate p-value adjustment is quite strict.

level α	classic.f	topo	elim	readjust	weight	classic.z	all.M
0.01	0	0	0	0	0	26	0
0.05	3	1	1	1	3	33	1
0.1	3	2	2	2	3	34	2

Table 5.11: *The number of significant GO terms for different cutoffs.*

Table 5.12 shows the raw p-values for the first GO terms presented in Table 5.9. If the raw p-values are considered, the number of GO terms for different cutoffs is shown in Table 5.13.

We see that the weight method is the less conservative method. As in Setup 1, the p-values reported by the z test are very small in comparison with all the others. For example,

	GO ID	classic.f	topo	elim	readjust	weight	classic.z	all.M
1	GO:0000115	1.2e−06	1.2e−06	1.2e−06	1.2e−06	1.2e−06	< 1e−30	1.2e−06
2	GO:0000084	2.6e−06	1.00000	1.00000	1.00000	2.6e−06	< 1e−30	0.00584
3	GO:0008630	6.2e−06	6.2e−06	6.2e−06	6.2e−06	6.2e−06	< 1e−30	6.2e−06
4	GO:0008629	4.3e−05	4.3e−05	4.3e−05	1.00000	2.8e−05	5.1e−29	0.00030
5	GO:0007612	8.1e−05	8.1e−05	8.1e−05	8.1e−05	8.1e−05	< 1e−30	8.1e−05
6	GO:0042770	0.00011	0.00011	0.00011	1.00000	8.5e−05	2.7e−21	0.00066
7	GO:0006298	0.00011	0.00011	0.00011	0.00011	0.00011	2.7e−21	0.00011
8	GO:0045005	0.00011	0.00011	0.00011	1.00000	0.00011	2.7e−21	0.00070
9	GO:0007165	0.00040	0.00040	0.00040	0.00040	0.00333	6.9e−05	0.00061

Table 5.12: *The raw* p*-values for the first 9 GO terms given by* classic.f *method.*

GO:0000115 has a raw p-value in the order of 10^{-85}. It seems that the z-test assigns low p-values particularly to GO nodes with a low number of annotated genes.

level α	classic.f	topo	elim	readjust	weight	classic.z	all.M
0.01	32	30	30	24	20	60	30
0.05	55	53	50	45	37	69	50
0.1	67	65	62	57	56	80	65

Table 5.13: *The number of significant GO terms for different cutoffs. The raw* p*-values are used.*

	classic.f	topo	elim	readjust	weight	classic.z	all.M
classic.f	1.000	0.840	0.682	0.388	0.483	0.877	0.967
topo	0.840	1.000	0.799	0.505	0.372	0.754	0.808
elim	0.682	0.799	1.000	0.525	0.506	0.628	0.759
readjust	0.388	0.505	0.525	1.000	0.193	0.405	0.405
weight	0.483	0.372	0.506	0.193	1.000	0.381	0.654
classic.z	0.877	0.754	0.628	0.405	0.381	1.000	0.819
all.M	0.967	0.808	0.759	0.405	0.654	0.819	1.000

Table 5.14: *Rank correlation for a sample of significant GO terms.*

For this setup the significant GO terms are more specific (the nodes are lower in the DAG hierarchy) than the ones in Setup 1. In fact the first 8 GO terms reported by the classic.f method are leaves or parents of the leaves. This can be seen in Figure 5.14(a). The counts for the nodes are plotted. Nodes that have no interesting genes mapped receive p-values of 1. For the other nodes the p-values are shown in Table 5.12. We note that the phenomena discussed in the example from Section 3.1, Figure 3.1, appear for the nodes GO:0006298 and GO:0045005. They have the same amount of mapped genes and thus their reported p-values are equal. Similarly, nodes GO:0000115 and GO:0000084 have very similar p-values.

In Figure 5.14 we also plot the p-values given by the method readjust and weight. We see that method readjust completely discards all parents of the significant leaves, thus considering as significant more specific GO terms. The p-values assigned by the weight method are smoother. In Section 5.3.3 we discuss in more detail the behavior of the algorithms.

From Table 5.12 we see that some methods return similar p-values. For example, the

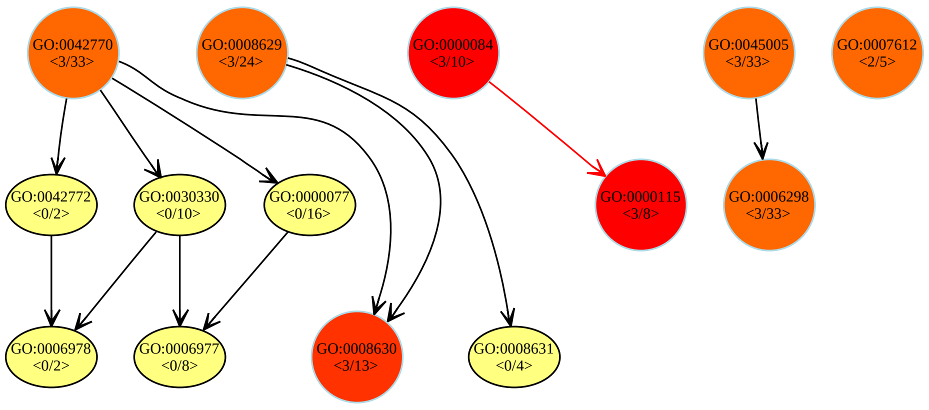
	classic.f	weight.weak	weight.log	weight.ratio
classic.f	1.000	0.339	0.490	0.352
weight.weak	0.339	1.000	0.730	0.990
weight.log	0.490	0.730	1.000	0.742
weight.ratio	0.352	0.990	0.742	1.000

Table 5.15: *Rank correlation for a sample of significant GO terms between different settings of the* weight *algorithm.*

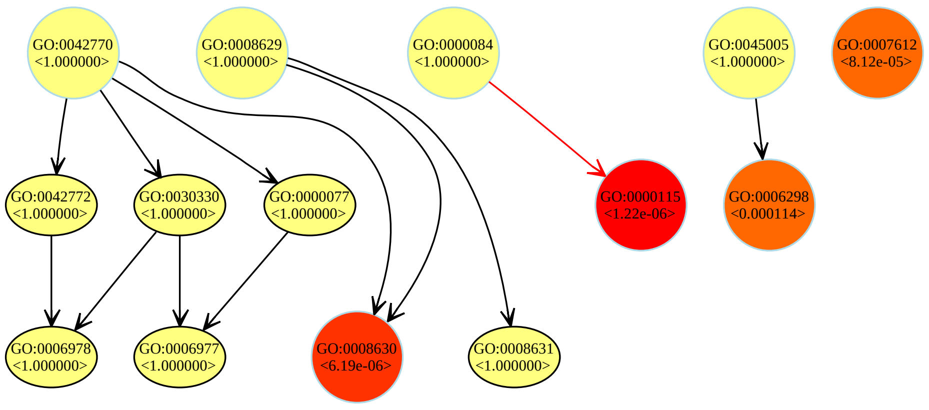
methods topo and elim give the same p-values for the shown GO terms. This behavior is normal, since the significant nodes are leaves or parents of the leaves.

In Table 5.14 the rank correlation for a subset of GO terms compiled as the one in Setup 1 is shown. The difference between the weighting functions used in the weight algorithm are shown in Table 5.15. We see that in this setup the methods are highly correlated. One reason for this is the small number of interesting genes, and thus few nodes, mainly nodes close to leaves, are found significant.

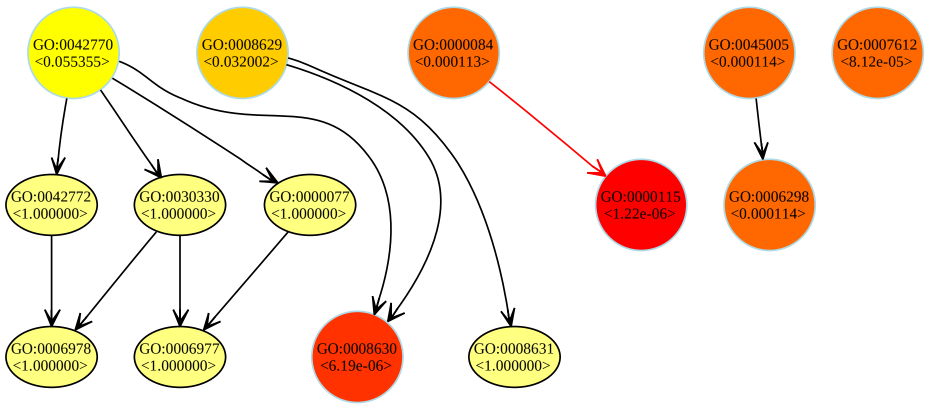
In the following we present the graphs induced by a subset of GO terms. Since some methods are highly correlated the graphs for some methods are skipped. The representation of the nodes is as explained in Section 5.3.1.



(a) classic.f method

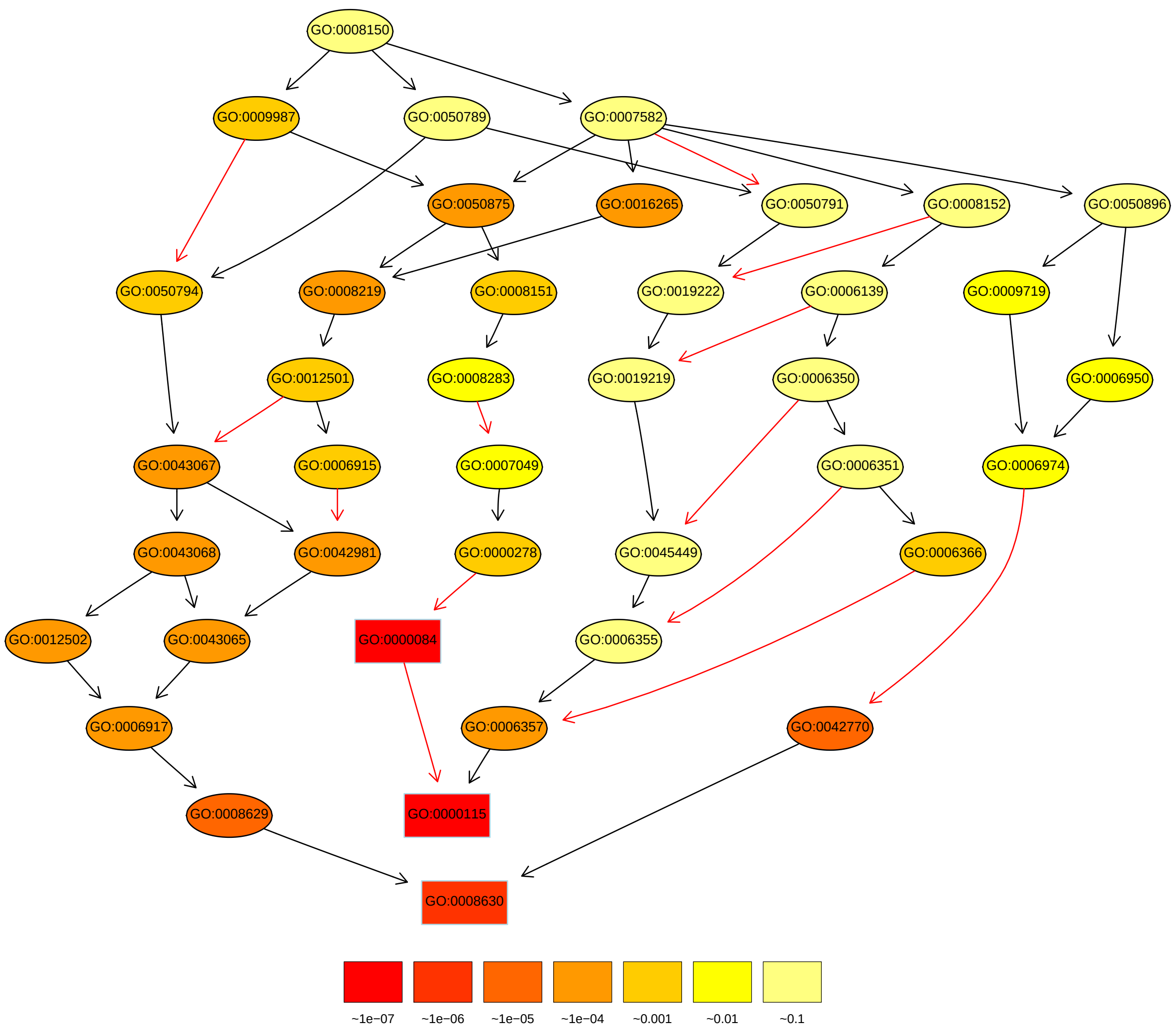


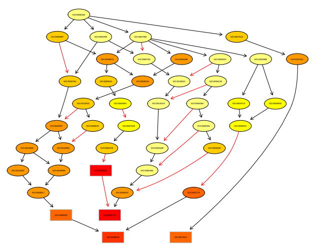
(b) readjust method

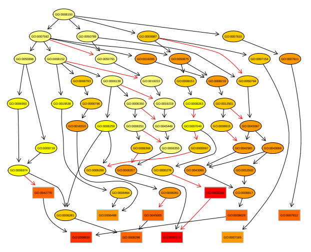


(c) weight method with weighing function sigRatio

Figure 5.14: *The lower induced subgraphs for the most significant 8 GO terms given by the* classic.f *method. In the figure (a) the counts for the nodes are displayed. In figure (b) and (c) the* p*-values for the* readjust*, respectively* weight *method are displayed.*




(a) top 5


(b) top 10

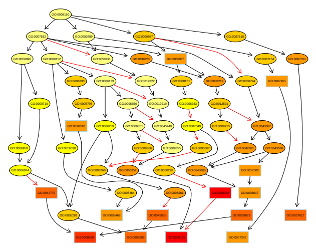
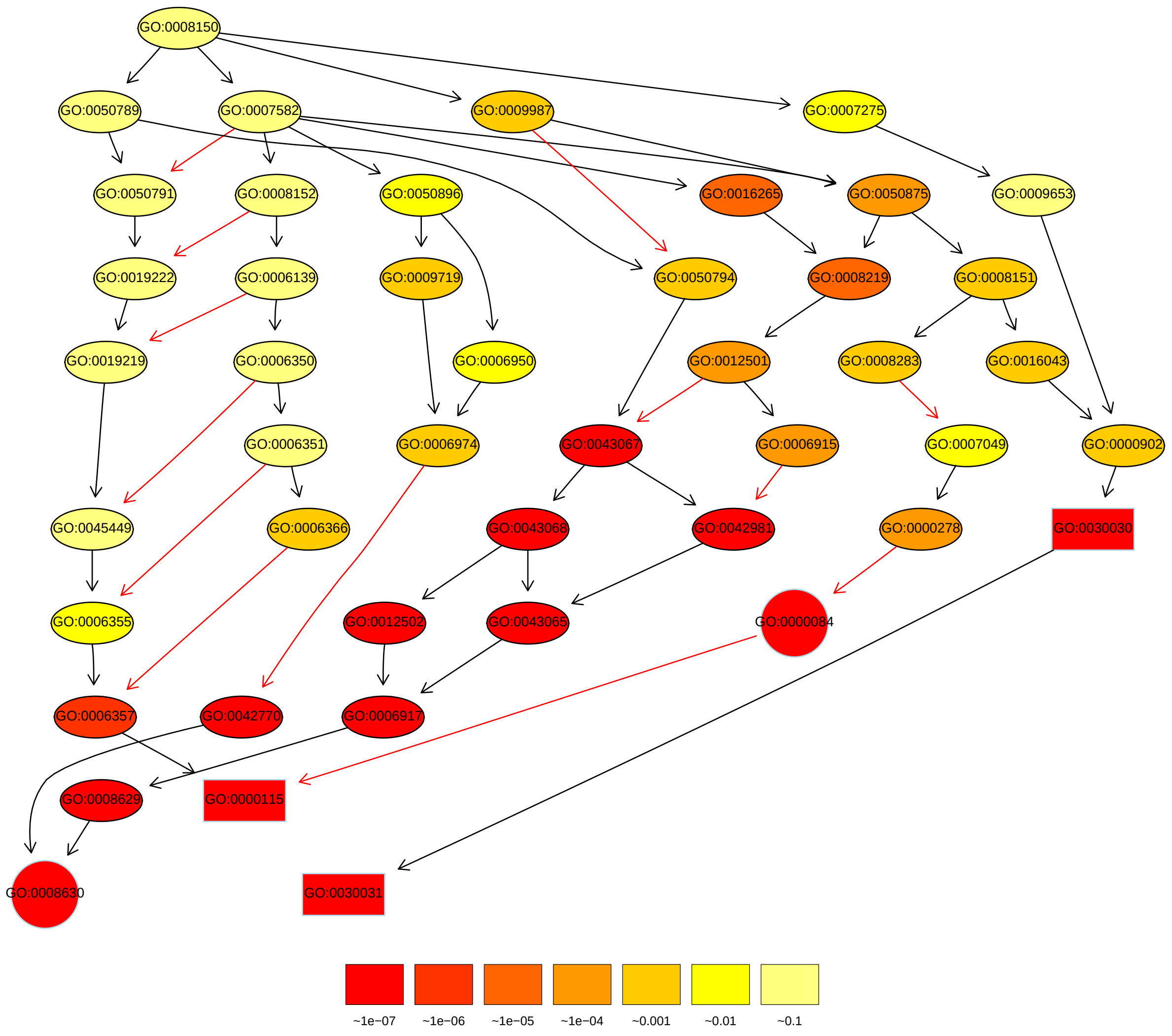
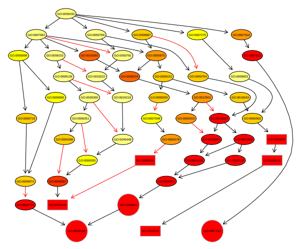
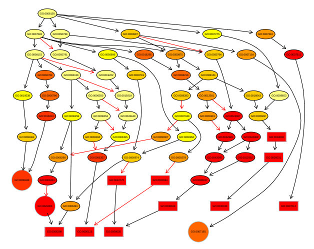

(c) top 15

Figure 5.15: *The subgraph induced by the most significant 3 GO terms found by method* classic.f. *This subgraph contains 40 nodes. There are 43 nodes in figure (a), 58 nodes in figure (b) and 59 in figure (c).*




(a) top 5


(b) top 10

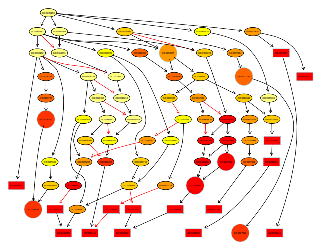
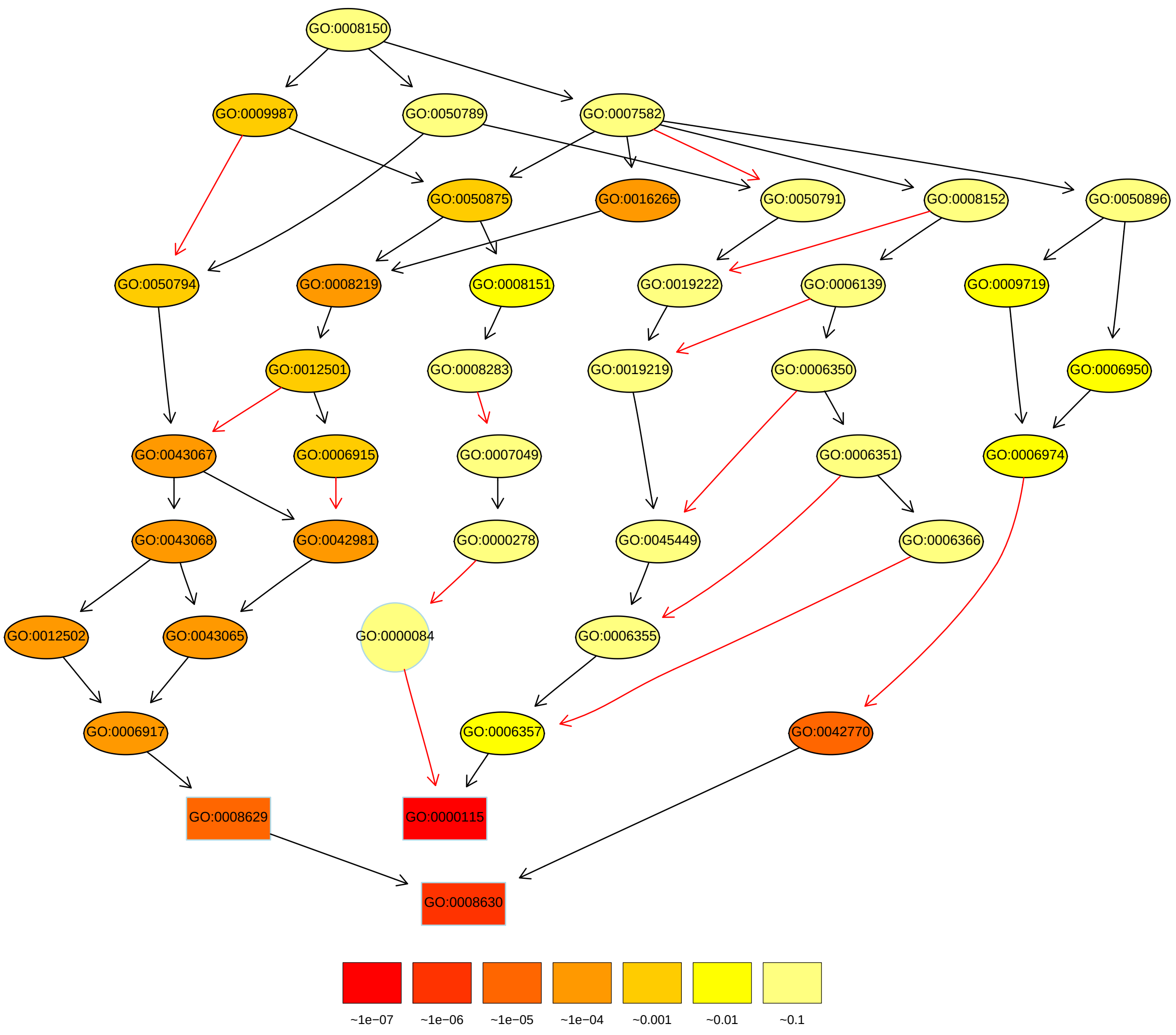
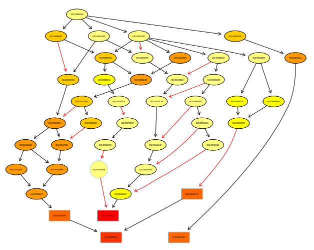
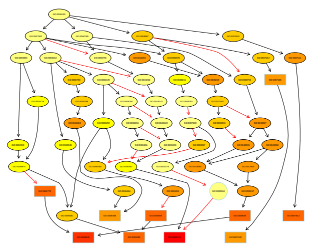

(c) top 15

Figure 5.16: *The subgraph induced by the most significant 3 GO terms found by method* classic.z. *This subgraph contains 46 nodes. There are 50 nodes in figure (a), 65 nodes in figure (b) and 72 in figure (c).*




(a) top 5


(b) top 10

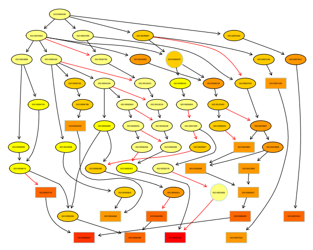
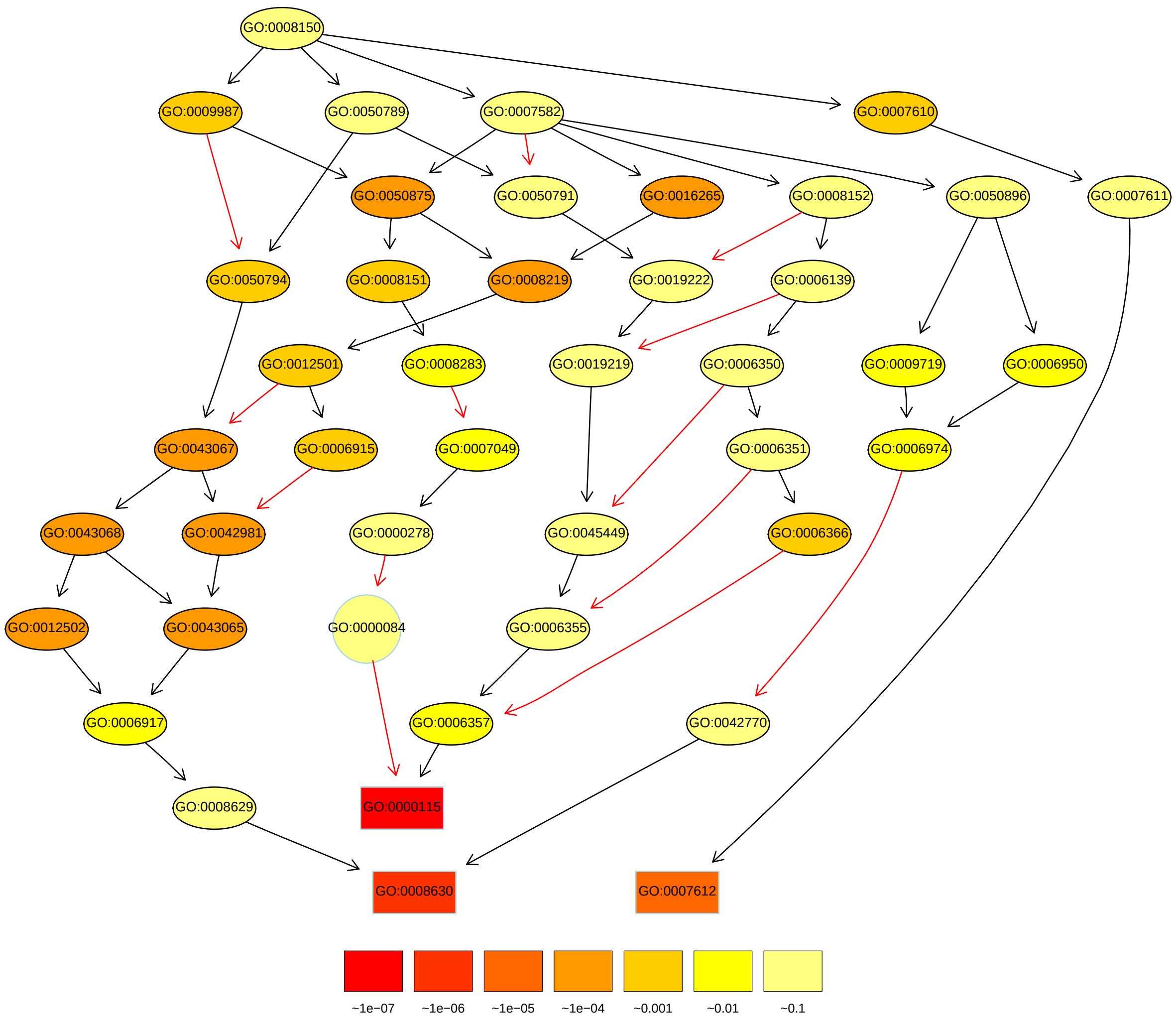
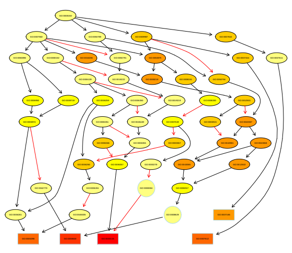
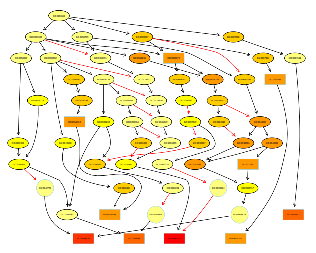

(c) top 15

Figure 5.17: *The subgraph induced by the most significant 3 GO terms found by method* elim. *This subgraph contains 40 nodes. There are 43 nodes in figure (a), 59 nodes in figure (b) and 59 in figure (c).*




(a) top 5


(b) top 10

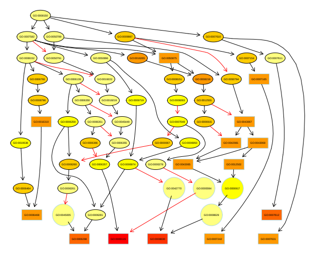
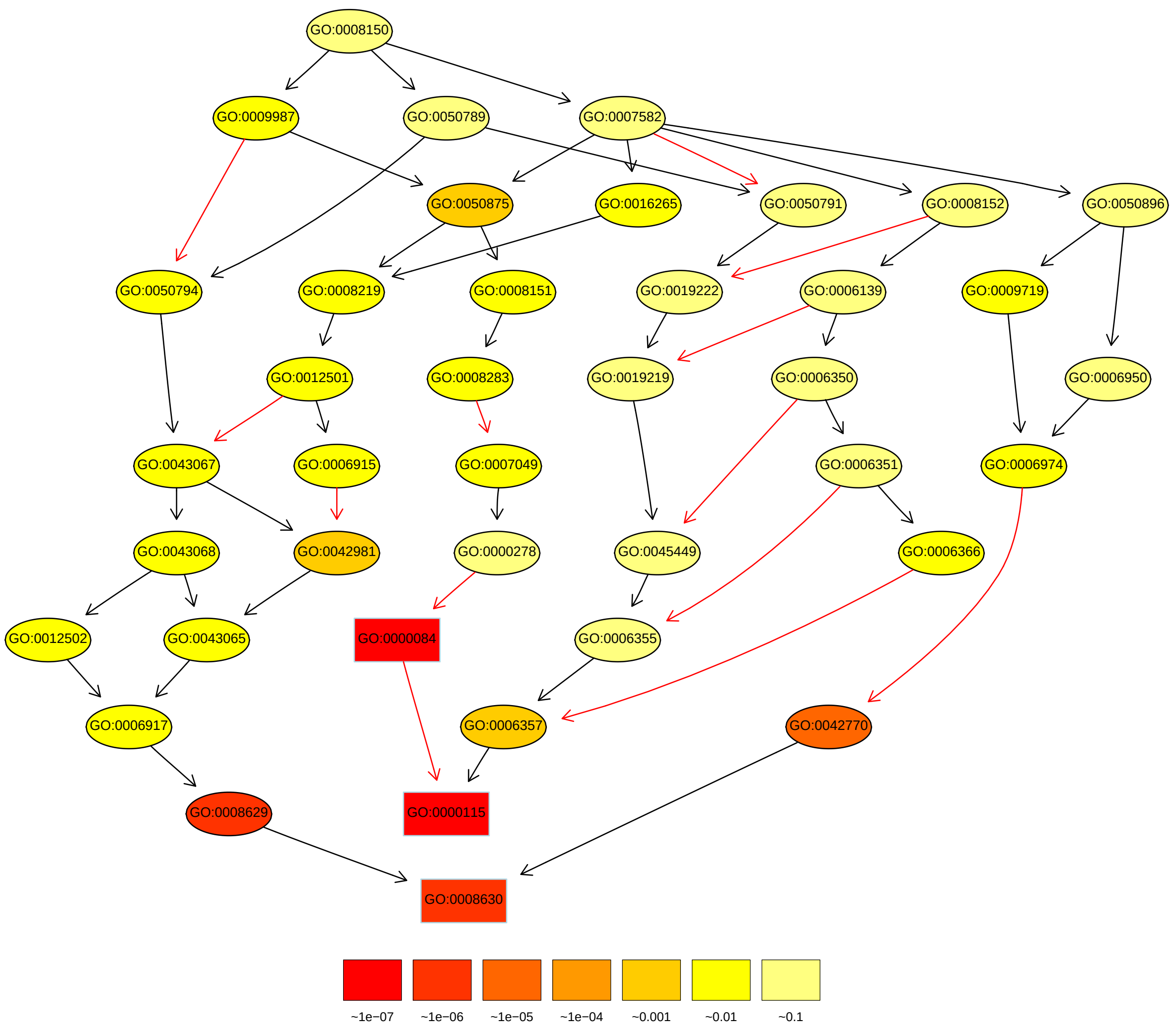
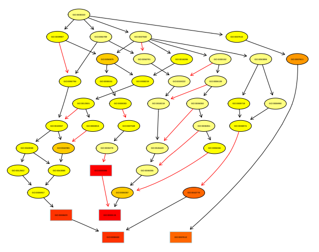
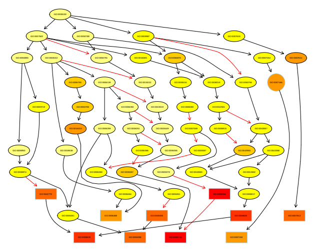

(c) top 15

Figure 5.18: *The subgraph induced by the most significant 3 GO terms found by method* readjust. *This subgraph contains 43 nodes. There are 52 nodes in figure (a), 59 nodes in figure (b) and 60 in figure (c).*




(a) top 5


(b) top 10

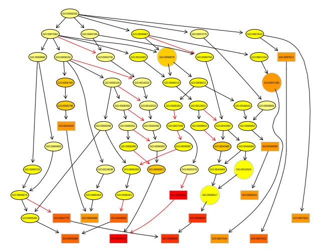
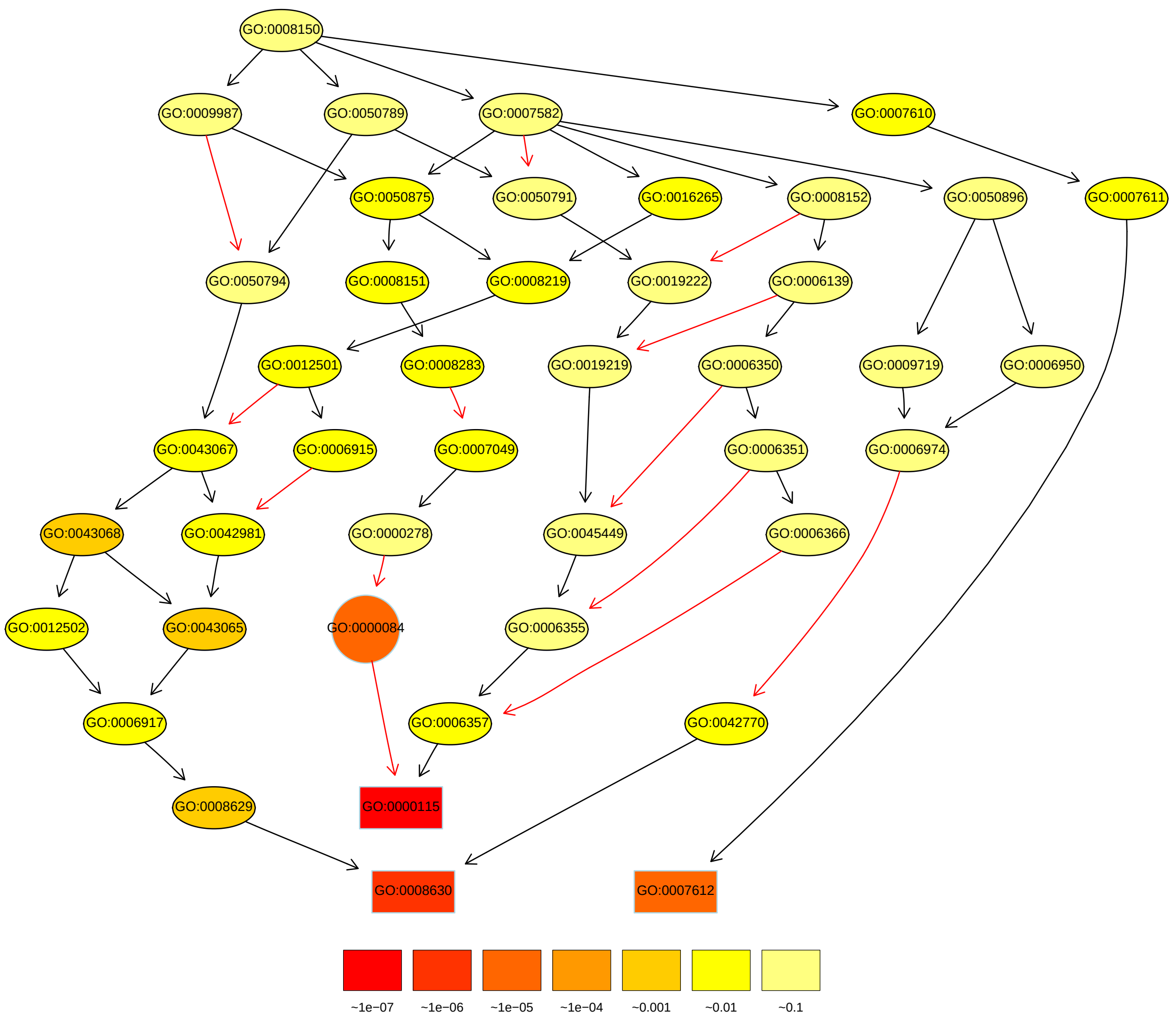
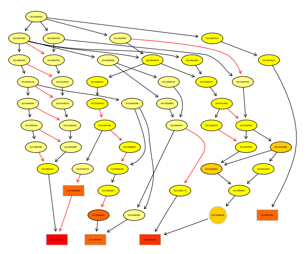
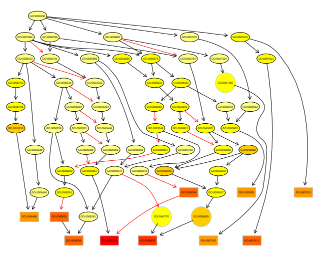

(c) top 15

Figure 5.19: *The subgraph induced by the most significant 3 GO terms found by method* weight*. The weighting function used is* sigRatioL( )*. This subgraph contains 40 nodes. There are 43 nodes in figure (a), 59 nodes in figure (b) and 66 in figure (c).*




(a) top 5


(b) top 10

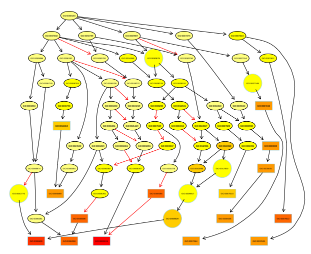
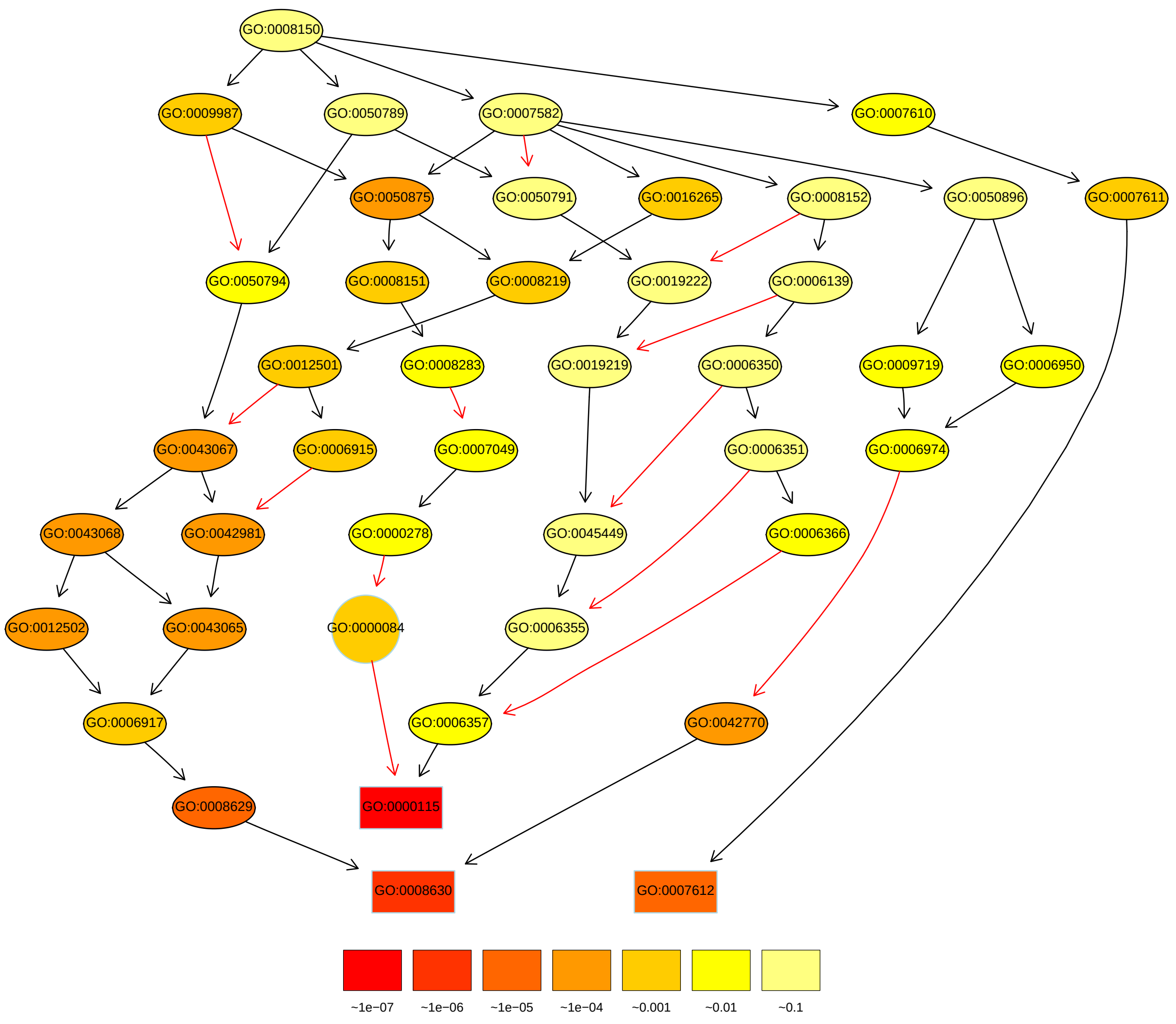
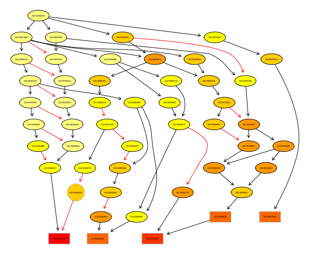
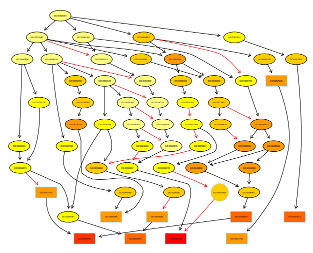

(c) top 15

Figure 5.20: *The subgraph induced by the most significant 3 GO terms found by method* weight. *The weighting function used is* sigRatio( ). *This subgraph contains 43 nodes. There are 50 nodes in figure (a), 65 nodes in figure (b) and 71 in figure (c).*




(a) top 5


(b) top 10

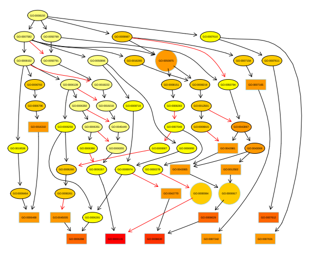

(c) top 15

Figure 5.21: *The subgraph induced by the most significant 3 GO terms found by method* all.M. *This subgraph contains 43 nodes. There are 50 nodes in figure (a), 59 nodes in figure (b) and 60 in figure (c).*

5.3.3 Advantages and disadvantages of scoring algorithms

In this section we discuss the advantages and disadvantages of the algorithms presented in Chapter 4 based on the results obtained from the two setups in Section 5.3.1 and Section 5.3.2.

Algorithm topo

Advantages. By comparing the graphs from Figure 5.5 and Figure 5.7 we can see that this method manages to eliminate some of the local dependences between the GO terms. For Setup 1 in the list of top 20 significant GO terms new nodes that are ranked lower by the classical approach are obtained. The graph induced by these 20 GO terms has 45 nodes, 9 more nodes than the graph obtained by the classic.f method. This expansion of the graph is useful since new areas in the graph are pointed out for investigation. For example, GO:0006468 and GO:0030097 give new paths in the graph that are not discovered by the classic.f method if the first 20 GO terms are considered.

Disadvantages. On the other hand the method depends on the `cutOff` parameter, see Algorithm 1. A large value for this parameter (close to 1) is considering too many nodes as significant during the running time, and the final result can be biased. By choosing a small value for this parameter the result is closer to the classic.f method; only few terms are considered significant and thus the p-values of only few terms are changed. There is no clear way in which the `cutOff` parameter should be chosen. In Section 5.4 we show how the algorithm performs for different cutoffs in a simulation setup.

The second problem with this algorithm is that a *significant - not-significant - significant* level effect appears. This is due to the fact that the significance of one node depends only on its direct children and not on the children of its children an so on. If the child of node x is considered significant than the genes from the child are removed from the node and the significance of node x is decreased. If the obtained p-value is above the cutoff, then when a parent of node x is processed node x is considered not-significant and no genes from x are removed from the parent. In this case the parent is found significant. This problem can be easily seen in Figure 5.7 for the path GO:0050896 $\longrightarrow$ GO:0009605 $\longrightarrow$ GO:0009607 $\longrightarrow$ GO:0006952 $\longrightarrow$ GO:0006955.

Algorithm elim

Advantages. This method was introduced in order to overcome the problems of the method topo. The level effect from method topo is not present. The main reason is that method elim has a global approach. When we decide to eliminate the genes from some child of node x, we eliminate them from all nodes of the upper.inducedGraph(x), see Algorithm 2. In this way we prefer more specific GO terms, thus we look for GO terms in the lower levels of the DAG. The graph induced by the first 20 GO terms reported by this method for Setup 1 has 49 nodes, 5 nodes more than topo, thus the significant nodes are more spread in the DAG, pointing to more areas for investigation. By inspecting the graph in Figure 5.8 we see that nodes very close to the root of the graph that were reported as significant by the classic.f method are completely discarded.

Disadvantages. Similar to the topo method the cutOff parameter is playing an important role in the results of the algorithm. With a very small value for this parameter the algorithm prefers very specific terms. This is not desirable since the GO term that best represents the list of interesting genes can be discarded if all its children are significant.

Another important disadvantage of the method is the simple strategy by which the genes are eliminated. The topology of the GO DAG is quite complex and the genes mapped to a node can come from different paths. This is not considered by the algorithm. Thus it can happen that we are removing too many genes.

The results of this algorithm need to be interpreted with caution. It is advisable to look at the parents of the reported GO terms for a more correct inference.

Also a disadvantage of this algorithm is its running time. As stated in Section 4.2, this algorithm is quadratic in the worst case in contrast with classic.f and topo which are linear.

Algorithm readjust

Advantages. The main advantage of this method is the way in which a cutoff is chosen. Since the classical p-values are computed in the first step, different multiple testing procedures can be used to adjust the p-values. Thus, we can decide with more confidence which nodes are significant. Using two p-values the level effect present in the topo method is removed. This can be seen in Figure 5.9. Note also that the first 20 GO terms are more spread in the DAG. From the results obtained it seems that this algorithm is close to the elim method. An advantage over the elim method is the linear running time of the algorithm.

Disadvantages. Using the adjusted p-values given by the classic.f method as evidence for the significance of the GO terms, the method is biased by the classic.f method.

Similar to the elim method this method is stopping too early. The nodes on the lower levels are advantaged over their ancestors, even if the ancestors are more significant. Thus, the issues discussed for elim also hold for this method.

Algorithm weight

Advantages. There are two important advantages for this method. The first is that there is no cutoff. The second is that the genes are not eliminated completely as before, but they are weighted.

By changing the weighting function different behaviors can be obtained. For example, we saw in Section 5.3.1 that a result close to classic.f can be obtained if the sigRatio.weak weighting function is used. On the other side, by giving very small weights we get more close to the elim method. In fact, by using the sigRatio weighting function the first 20 GO terms induce a graph with 69 nodes, a graph which is twice as big as the one obtained if the classic.f method is used. This means that more areas in the DAG are pointed out for investigated, see Figure 5.12 and Figure 5.20. Note also that the significance of the nodes from Figure 5.20 is reduced compared to the other methods.

Another advantage of Algorithm 4 is that it offers multiple strategies for updating the weights of the genes. For our experiments we used a simple strategy for recomputing the significance of the children of a node.

Disadvantages. Some problems also emerge for this method. The issue on the different paths from which the genes come is not considered by the method. It is also not clear what weighting function is better. More information than the simple ratio is desired to be included in the weights. The ratio is considering only the difference between the terms and not their magnitude. For example, $\frac{10^{-1}}{10^{-4}}$ equal $\frac{10^{-10}}{10^{-13}}$, but the nodes with a p-value of 10^{-10}, respectively 10^{-13}, should be considered more important.

Another problem is shown in Figure 5.14(c) in which the GO terms GO:0045005 and GO:0006298 are reported with the same p-value. In this case the weights to be assigned are 1, thus no weighting is performed. It is not clear what should be done in this case and if the algorithm should prefer more specific nodes or less specific nodes.

The quadratic running time of this method can also be considered a disadvantage.

5.4 Comparing the algorithms on simulated data

In Section 5.3 we analyzed the behavior of the algorithms presented in Chapter 4 on the **ALL** dataset. The problem with this analysis is the subjectivity with which the results are interpreted. In this section we present the performance of the algorithms on simulated data.

5.4.1 Simulation setup

When analyzing a real dataset like the **ALL** dataset, we need to assess if the GO terms that are found significant by an algorithm are the true significant ones. The problem with real data is that we do not know the truth. In a simulation setup we fix some GO terms to be the significant ones. The performance of the algorithm is then measured w.r.t. the number of truly identified nodes.

Next we describe how the data necessary for the input of the algorithms is compiled. As stated in Figure 1.1 all we need is a graph with the properties of the GO DAG, a list of items (genes) and a list of interesting items.

We want to stay as close as possible to the reality. Thus, we choose the GO DAG as the underlying graph. The list of items contains all the probes found on the HGU95aV2 Affymetrix chip that are annotated to the GO hierarchy. The BP hierarchy was chosen as the hierarchy. We could map only 9623 probes out of a total of 12625 probes. The resulting DAG has 2311 nodes and 3525 edges.

A set of nodes that fulfill certain criteria is randomly selected. This set of nodes is denoted as the *wanted nodes.* Our criterion was to select nodes with the number of mapped items in a certain interval. For example, if we are interested in nodes close to the root of the DAG, then we set an interval of $[100, 1000]$. We impose such a constraint since we want to overcome two problems. First, the small nodes are too specific and cannot synthesize the underlying biology, see (Gentleman, 2004c). On the other hand, nodes with too many items mapped to them are to general. To select the wanted nodes all the nodes that do not fulfill our criterion are discarded. From the remaining nodes the desired number of wanted nodes is selected with equal probability.

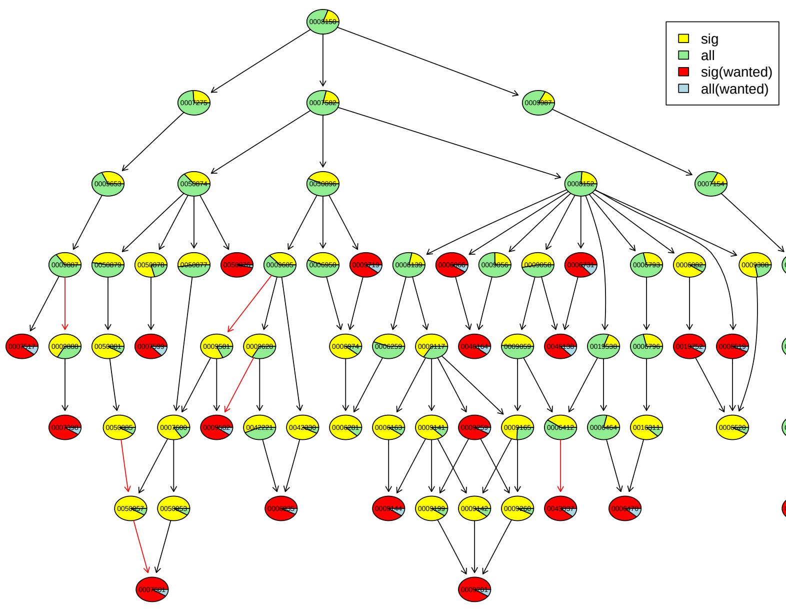
After selecting the wanted nodes the list of interesting items is compiled. From each selected node the list of items is retrieved. Then, the union of these items (with duplicates removed) is the list of interesting items. To model better the reality, some noise is introduced in the list of interesting items. A fraction, for example 10%, of interesting items is randomly selected and replaced with items that are not in the interesting list.

Having compiled the lists of all items and the list of interesting items the goal is to recover as best as possible the wanted nodes. To get an idea how the wanted nodes are distributed in the DAG and how the dependences between nodes play a role in the results, we plot in Figure 5.22 the subgraphs induced by the wanted nodes and the nodes found significant by the classic algorithm. The wanted nodes have numbers of items in the interval $[100, 500]$. We choose this interval for visualization reasons (for the interval $[10, 50]$ we obtain in average a subgraph with more than 500 nodes, thus hard to visualize).

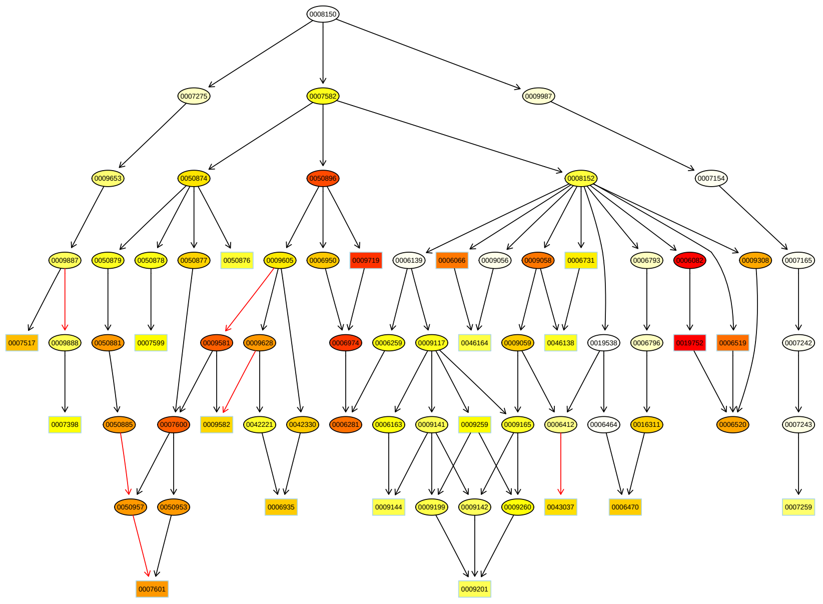
In Figure 5.22(a) the pie plots show the amount of interesting items mapped to the nodes, see Section 5.2.1. We can see that a wanted node propagates its significance to the neighbors. Trivially, all the children of a wanted node are more dense than its parent, see property 2 and 3 from Figure 1.1.

In Figure 5.22(b) the result of the classic algorithm is shown. The boxes represent the wanted nodes. We see that many wanted nodes are not considered significant (they are colored light yellow). In the left part of the graph we can see an area with many significant nodes (orange-red area).

In the next section we discuss the results obtained using this type of simulation. All tested algorithms are using Fisher's exact test to score the nodes for significance. The algorithms are the ones presented in Chapter 4. Different parameters are tested for each method. For the weight method all weighting functions described in Section 5.3.1 are used. We test different combination methods that report the average of the p-values. The methods are named with the initials of the methods used in the average. For example, CEW denotes the method that averages the p-values of classic, elim and weight algorithms.



(a) The wanted nodes



(b) The nodes obtained with the classic method

Figure 5.22: *The subgraphs induced by the wanted nodes in a simulation setup. There are 20 wanted nodes. The noise level is set to 10%. The red and blue nodes from figure (a) are the wanted nodes. In figure (b) the boxes represent the wanted nodes.*

5.4.2 Results

To assess the performance of the algorithms we used the following measure. For each method, the nodes are sorted in ascending order of their computed p-values. We fix a k and count how many wanted nodes are among the top k nodes

$$\mathsf{score}_k^0(\mathcal{M}) = |\mathsf{top}_k(\mathcal{M}) \cap \mathit{wanted}| \,.$$

$\mathsf{top}_k(\mathcal{M})$ denotes the set of first k nodes for method $\mathcal{M}$ and wanted denotes the set of wanted nodes. Methods that obtain a higher score are considered better.

To get more insight into how the methods account for the topology of the graph, the following scores are defined:

$$\mathsf{score}_k^1(\mathcal{M}) = |(\mathsf{top}_k(\mathcal{M}) \cup \mathsf{children}(\mathsf{top}_k(\mathcal{M})) \cup \mathsf{parents}(\mathsf{top}_k(\mathcal{M}))) \cap \mathit{wanted}| \,,$$
$$\mathsf{score}_k^{1p}(\mathcal{M}) = |(\mathsf{top}_k(\mathcal{M}) \cup \mathsf{parents}(\mathsf{top}_k(\mathcal{M}))) \cap \mathit{wanted}| \,.$$

Thus, score_k^{1p} looks at the first k nodes together with their parents. We compute these scores based on the observation that some algorithms are considering as significant more specific GO terms, see the discussion of the methods elim and readjust from Section 5.3.3. If for some method $\mathcal{M}$ the difference $\mathsf{score}_k^{1p}(\mathcal{M}) - \mathsf{score}_k^0(\mathcal{M})$ is large then method M is stopping too early, discarding the parents of some significant nodes. If, on the other hand, both scores are almost equal, then the first k nodes are close neighbors (the effect seen in Figure 5.5), and thus the method gives many false positive nodes.

k	Score	class	w.log	w.weak	w.ratio	topo	elim	readj	all.M	CWWIWs	EWWIWs	WWIWs	WIWs
	0	**9**	**15**	**13**	**18**	**13**	**20**	**20**	**21**	**15**	**21**	**17**	**18**
	1p	9	15	13	18	15	22	22	21	15	21	17	18
25	1	12	19	15	19	17	22	22	22	17	22	18	19
	2p	9	15	13	18	15	25	24	22	15	22	17	18
	2	19	23	20	21	25	25	25	25	22	25	20	22
	0	**22**	**27**	**24**	**31**	**25**	**31**	**29**	**33**	**30**	**33**	**31**	**31**
	1p	22	28	24	33	28	42	38	39	31	37	31	32
50	1	27	31	28	33	30	42	39	39	33	37	33	33
	2p	22	28	24	33	28	47	43	39	31	37	31	32
	2	31	34	33	35	36	47	45	42	37	39	36	35
	0	**29**	**36**	**31**	**44**	**33**	**32**	**30**	**40**	**39**	**42**	**43**	**43**
	1p	29	37	31	45	41	44	42	45	41	46	45	44
75	1	33	38	35	45	42	44	42	46	42	46	46	44
	2p	29	38	31	46	41	49	47	45	41	46	45	45
	2	37	41	38	46	43	49	47	46	43	47	47	46
	0	**36**	**44**	**39**	**47**	**38**	**34**	**31**	**42**	**47**	**46**	**46**	**47**
	1p	37	45	39	49	48	45	43	47	48	48	48	49
100	1	40	46	41	49	48	46	43	47	48	48	48	49
	2p	37	46	39	49	48	50	48	47	48	48	48	49
	2	43	47	42	49	48	50	49	47	48	48	48	49

Table 5.16: *The results for one run on simulated data with 50 wanted nodes, 10% noise level and 10 to 20 items mapped to the wanted nodes.*

Similarly, we define $\mathsf{score}_k^2(\mathcal{M})$ and $\mathsf{score}_k^{2p}(\mathcal{M})$ in which we look two levels away from the first k nodes.

Table 5.16 shows the results obtained for a simulation run. There are 50 wanted nodes. The number of items mapped to a node is between 10 and 20. There is 10% noise introduced in the list of interesting items. For each method all five scores are given for the different values of k. To get a better idea of the differences between the methods in Table 5.17 we give the percentage of wanted nodes found in top k nodes. The table gives only the findings for $\mathtt{score}_k^0$.

k	class	w.log	w.weak	w.ratio	topo	elim	readj	all.M	CWWIWs	EWWIWs	WWIWs	WIWs
25	0.18	0.30	0.26	0.36	0.26	0.40	0.40	0.42	0.30	0.42	0.34	0.36
50	0.44	0.54	0.48	0.62	0.50	0.62	0.58	0.66	0.60	0.66	0.62	0.62
75	0.58	0.72	0.62	0.88	0.66	0.64	0.60	0.80	0.78	0.84	0.86	0.86
100	0.72	0.88	0.78	0.94	0.76	0.68	0.62	0.84	0.94	0.92	0.92	0.94

Table 5.17: *The percentage of wanted nodes found for one run on simulated data with 50 wanted nodes, 10% noise level and 10 to 20 items mapped to the wanted nodes.*

We see that all proposed methods perform better than the classic method for small values of k. The elim method seems to be the best one in this case, beating the classic method with almost 20%. Averaging the reported p-values for different methods also helps. We obtain 11 more nodes than the classic method with the all.M method. With k increasing, the methods that eliminate items, see Section 4.2, are under-performing for $\mathtt{score}_k^0$. The weight method is always above the classic. In particular, we are more than 10% better than classic if the $\mathtt{sigRatio.log}$ weight function is used. On the other hand, we can see that the method elim is preferring more specific nodes as we known. Note that there is no difference between $\mathtt{score}_k^1$ and $\mathtt{score}_k^{1p}$ for this method. Also by looking at the parents and the grand-parents of the top 50 nodes we find 47 wanted nodes, 3 less than optimal. Thus, the elim algorithm successfully identifies the areas in the DAG where the wanted nodes are located, but fails to point at them. This is not the case for the classic algorithm. For this algorithm looking at the parents gives us no information in this case.

k	class	w.log	w.weak	w.ratio	topo	elim	readj	all.M	CWWIWs	EWWIWs	WWIWs	WIWs
25	0.11	0.26	0.15	0.28	0.18	0.34	0.31	0.29	0.24	0.31	0.28	0.29
50	0.29	0.51	0.35	0.56	0.37	0.55	0.49	0.53	0.48	0.57	0.52	0.56
75	0.45	0.71	0.52	0.76	0.53	0.62	0.56	0.71	0.67	0.76	0.73	0.76
100	0.62	0.84	0.67	0.79	0.64	0.67	0.61	0.83	0.81	0.87	0.84	0.84

Table 5.18: *Average percentage of wanted nodes found over 100 runs on simulated data with 50 wanted nodes, 10% noise level and 10 to 20 items mapped to the wanted nodes.*

To obtain more precise results, 100 simulations were performed. Table 5.19 and Table 5.18 present for each k the average scores. We see that the results are similar to the ones from Table 5.16. In average the weight.log method is 20% better than classic. The best result is obtained by the averaging method EWWIWs.

A more difficult dataset can be obtained if the noise in increased and nodes higher in the hierarchy are selected. Tables 5.20, 5.21 and 5.22 show the results for simulated data in which the 15 wanted nodes are chosen such that 100 to 200 items are mapped to them, and the list of interesting items contains 50% noise.

k	Score	class	w.log	w.weak	w.ratio	topo	elim	readj	all.M	CWWIWs	EWWIWs	WWIWs	WIWs
	0	**5.4**	**13.2**	**7.6**	**14.2**	**9.0**	**16.9**	**15.6**	**14.6**	**12.1**	**15.4**	**13.8**	**14.5**
	1p	5.5	13.5	7.6	14.5	11.2	22.9	21.3	17.5	12.3	17.2	14.0	14.8
25	1	12.2	18.6	13.7	20.0	15.1	23.1	21.8	21.1	18.3	21.7	19.7	20.2
	2p	5.5	13.6	7.6	14.5	11.2	23.9	22.7	17.8	12.3	17.4	14.1	14.8
	2	19.3	22.3	19.9	23.2	22.5	24.8	24.6	24.4	22.6	24.3	23.2	23.3
	0	**15**	**25**	**17**	**28**	**18**	**27**	**25**	**27**	**24**	**28**	**26**	**28**
	1p	15	26	17	29	24	40	36	31	25	31	27	29
50	1	23	32	24	35	29	41	37	35	32	36	34	36
	2p	15	26	17	29	24	43	39	31	25	31	27	29
	2	29	36	30	39	35	45	42	40	36	40	37	39
	0	**23**	**36**	**26**	**38**	**26**	**31**	**28**	**35**	**34**	**38**	**36**	**38**
	1p	23	36	26	39	36	44	42	40	34	40	37	39
75	1	30	42	32	46	39	45	43	44	41	45	44	46
	2p	23	36	26	40	36	48	46	40	35	41	38	39
	2	35	44	36	48	43	48	48	46	43	47	46	47
	0	**31**	**42**	**34**	**40**	**32**	**33**	**31**	**41**	**41**	**43**	**42**	**42**
	1p	31	43	34	41	44	46	44	45	41	45	43	43
100	1	36	47	38	48	46	46	45	48	46	48	47	48
	2p	31	43	34	42	44	49	48	46	41	45	43	43
	2	40	49	41	50	48	49	49	49	47	49	49	49

Table 5.19: *Average results over 100 runs on simulated data with 50 wanted nodes, 10% noise level and 10 to 20 items mapped to the wanted nodes.*

k	class	w.log	w.weak	w.ratio	topo	elim	readj	all.M	CWWIWs	EWWIWs	WWIWs	WIWs
10	0.24	0.32	0.25	0.38	0.28	0.40	0.38	0.36	0.30	0.35	0.31	0.30
25	0.58	0.75	0.62	0.76	0.61	0.63	0.56	0.77	0.71	0.76	0.72	0.71
50	0.94	0.95	0.94	0.81	0.78	0.72	0.65	0.97	0.96	0.97	0.96	0.96

Table 5.20: *Average percentage of wanted nodes found over 100 runs on simulated data with 15 wanted nodes, 50% noise level and 100 to 200 items mapped to the wanted nodes.*

k	Score	class	w.log	w.weak	w.ratio	topo	elim	readj	all.M	CWWIWs	EWWIWs	WWIWs	WIWs
	0	**3.6**	**4.8**	**3.8**	**5.6**	**4.3**	**5.9**	**5.7**	**5.4**	**4.5**	**5.2**	**4.6**	**4.5**
	1p	3.7	5.2	3.8	6.2	5.7	8.8	8.1	6.4	4.7	6.0	5.0	4.8
10	1	5.2	6.3	5.1	7.5	6.8	8.9	8.3	7.3	6.1	6.8	6.1	6.0
	2p	3.7	5.2	3.8	6.3	5.7	9.7	9.2	6.7	4.8	6.2	5.0	4.8
	2	6.4	7.0	6.2	8.3	8.2	9.9	9.5	8.3	7.0	7.7	7.0	6.9
	0	**8.8**	**11.2**	**9.3**	**11.5**	**9.2**	**9.4**	**8.4**	**11.5**	**10.7**	**11.3**	**10.9**	**10.7**
	1p	8.8	11.7	9.4	12.5	12.8	12.7	11.9	12.5	10.9	12.1	11.2	11.0
25	1	9.9	12.7	10.4	14.2	13.3	12.9	12.2	13.2	12.0	12.8	12.3	12.0
	2p	8.8	11.8	9.4	12.8	12.8	14.1	13.7	12.8	11.0	12.3	11.3	11.0
	2	10.8	13.2	11.0	14.8	13.6	14.3	14.1	13.7	12.5	13.3	12.7	12.5
	0	**14.0**	**14.2**	**14.1**	**12.2**	**11.6**	**10.8**	**9.8**	**14.6**	**14.4**	**14.6**	**14.4**	**14.4**
	1p	14.1	14.5	14.2	13.1	14.9	13.7	13.0	14.9	14.5	14.8	14.6	14.5
50	1	14.3	14.9	14.4	14.7	14.9	13.8	13.3	15.0	14.8	14.9	14.9	14.8
	2p	14.1	14.6	14.2	13.4	14.9	14.6	14.3	14.9	14.6	14.9	14.7	14.6
	2	14.5	15.0	14.5	15.0	14.9	14.7	14.7	15.0	14.9	15.0	14.9	14.9

Table 5.21: *Average results over 100 runs on simulated data with 15 wanted nodes, 50% noise level and 100 to 200 items mapped to the wanted nodes.*

k	Score	class	w.log	w.weak	w.ratio	topo	elim	readj	all.M	CWWIWs	EWWIWs	WWIWs	WIWs
	0	**4**	**4**	**4**	**5**	**4**	**5**	**5**	**7**	**4**	**7**	**4**	**4**
	1p	4	5	4	6	6	8	7	8	5	7	5	5
10	1	4	5	4	6	7	8	7	8	5	7	5	5
	2p	4	5	4	6	6	9	8	9	5	8	5	5
	2	4	5	4	6	7	9	8	9	5	8	5	5
	0	**9**	**11**	**9**	**14**	**8**	**12**	**10**	**11**	**11**	**11**	**11**	**11**
	1p	9	11	9	15	12	13	12	11	11	11	11	11
25	1	9	11	9	15	12	13	12	11	11	11	11	11
	2p	9	11	9	15	12	14	13	11	11	11	11	11
	2	10	11	9	15	12	14	13	11	11	11	11	11
	0	**13**	**15**	**14**	**14**	**13**	**13**	**13**	**15**	**15**	**15**	**15**	**15**
	1p	13	15	14	15	15	13	13	15	15	15	15	15
50	1	13	15	14	15	15	13	13	15	15	15	15	15
	2p	13	15	14	15	15	14	14	15	15	15	15	15
	2	14	15	14	15	15	14	15	15	15	15	15	15

Table 5.22: *The results for one run on simulated data with 15 wanted nodes, 50% noise level and 100 to 200 items mapped to the wanted nodes.*

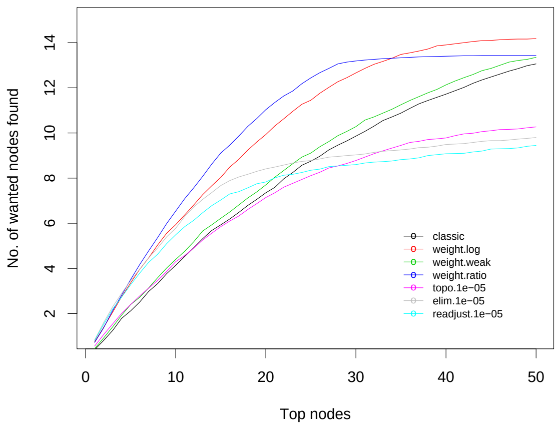
We tried different parameters for the simulation: the number of nodes, the number of items allowed for the wanted nodes, and the noise level. The results are similar to the one presented above.

Figure 5.23 and Figure 5.24 show how the average score is varying as a function of k. We see that for $\mathtt{score}^0$ the weight method is outperforming always the classic method.

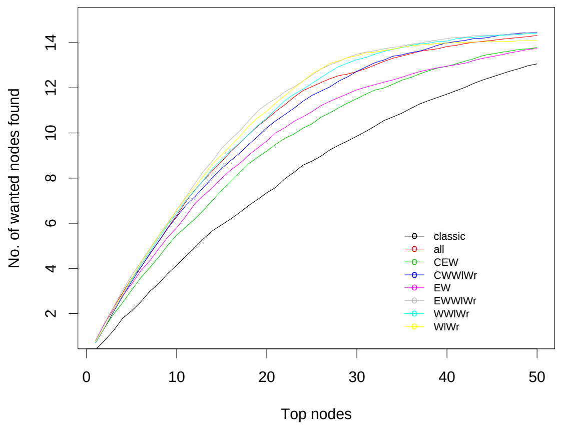
Figure 5.25 shows the influence of the parameters on the performance of the method. We observe that in average a very low cutoff gives better results. For the weight method it seems that the best choice is the sigRatioL weighting function. The sigRatio function is better for smaller values of k, but after a certain threshold it is dominated by the log ratio. With sigRatio.*weak* the weight method is close to the classic method, but it is never worse.

We are also interested to see how the score is affecting each method. In Figure 5.26 and Figure 5.27 the improvement due to a different score is shown for each method. The elim method is almost optimal if $\mathtt{score}^{2p}$ is used. For the classic method there is almost no improvement, when looking only at the parents. A small improvement is obtained by looking at the complete neighborhood.

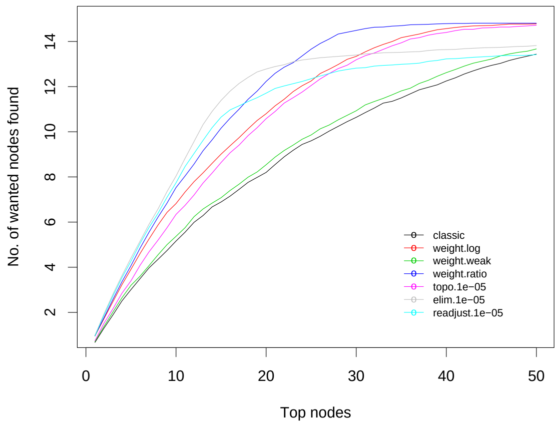
Finally, we are interested how the noise introduced in the list of interesting items affects the performance of the algorithms. In practice there is a considerably amount of noise in the data. We believe that a level of 50% noise in our case makes the problem harder. Figure 5.28 shows the method's performance in noisier instances. Note that the method most affected by the noise is the classic method.



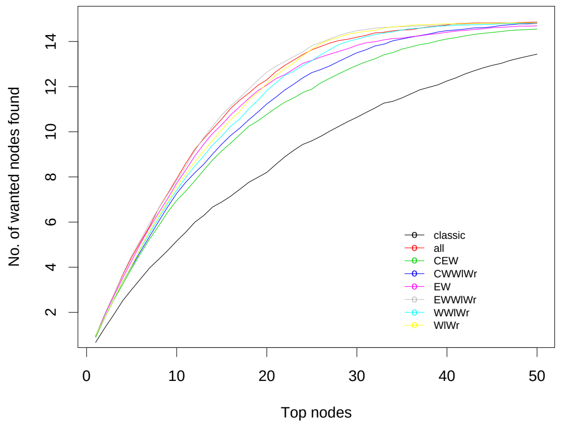
(a) score^0



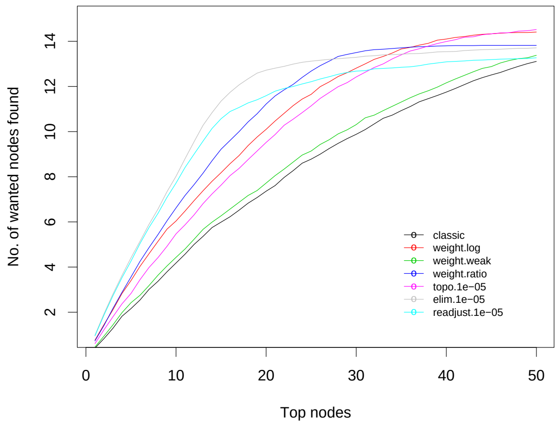
(b) score^0



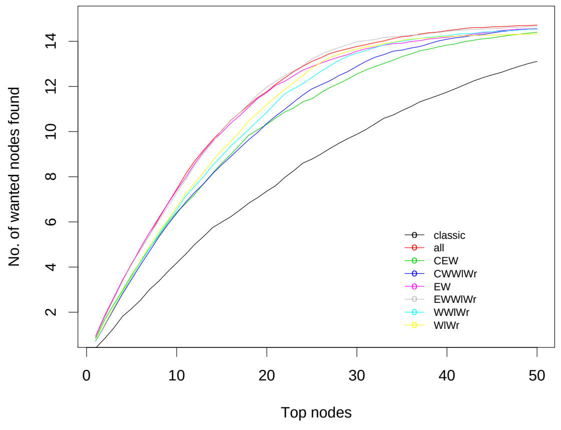
(c) score^1



(d) score^1

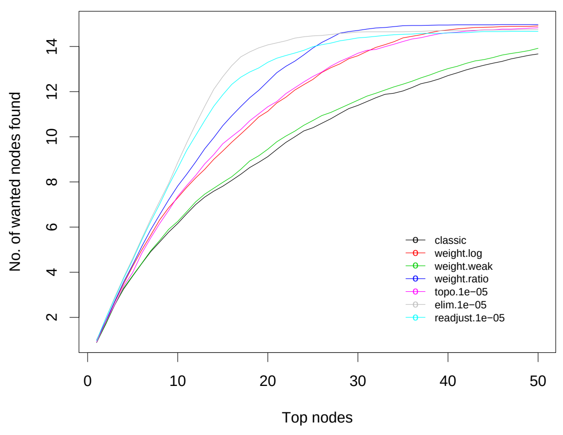


(e) score^{1p}

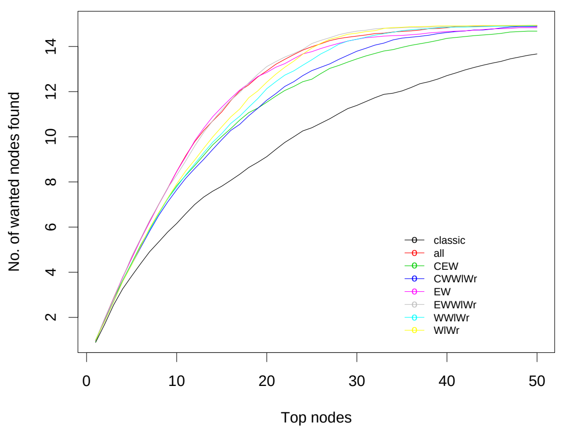


(f) score^{1p}

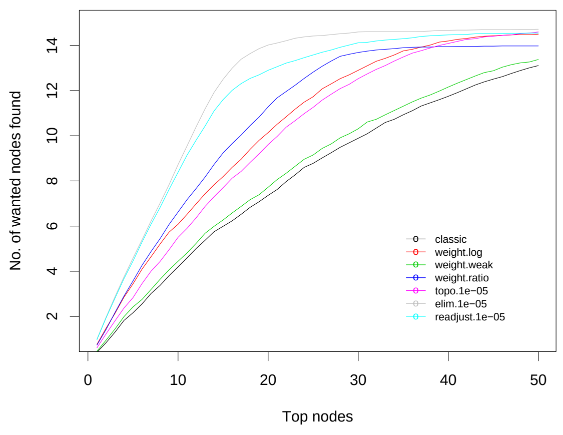
Figure 5.23: *The average performance of the algorithms for 100 simulation runs. There are 15 wanted nodes containing 10 to 20 items. The noise level is set to 10%*



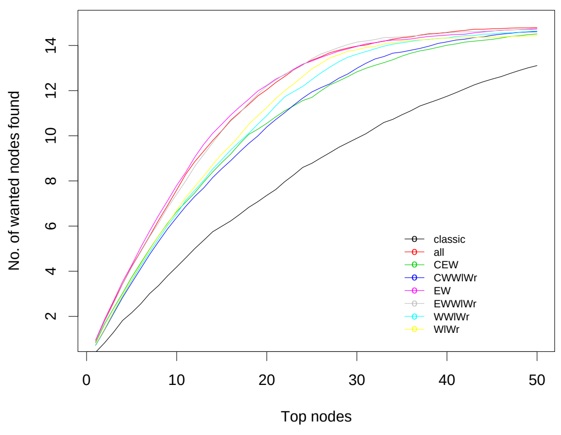
(a) $\mathtt{score}^2$



(b) $\mathtt{score}^2$

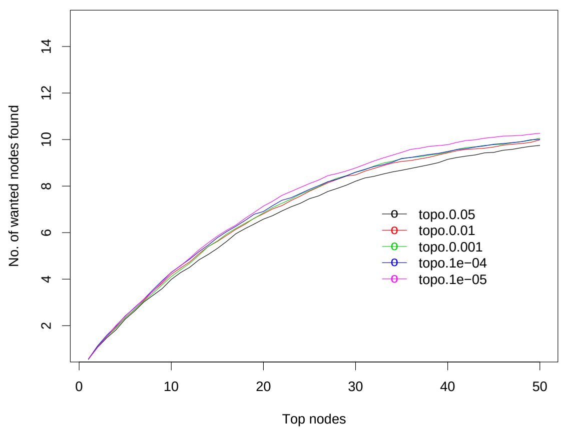


(c) $\mathtt{score}^{2p}$

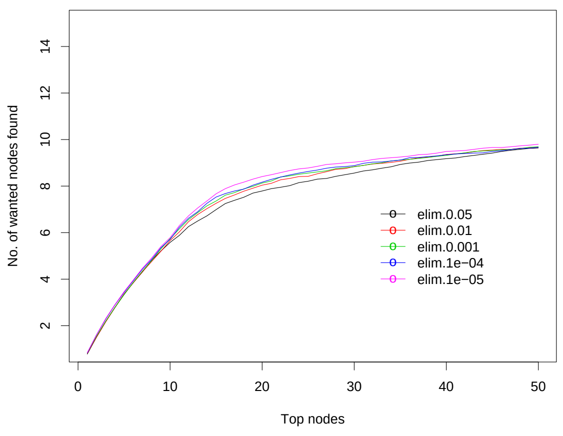


(d) $\mathtt{score}^{2p}$

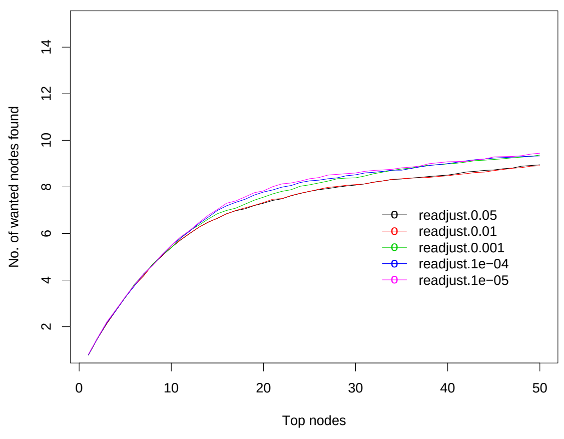
Figure 5.24: *The average performance of the algorithms for 100 simulation runs. There are 15 wanted nodes containing 10 to 20 items. The noise level is set to 10%*



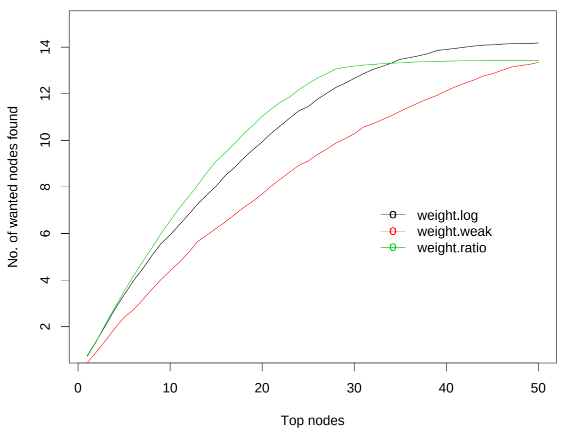
(a) topo method



(b) elim method

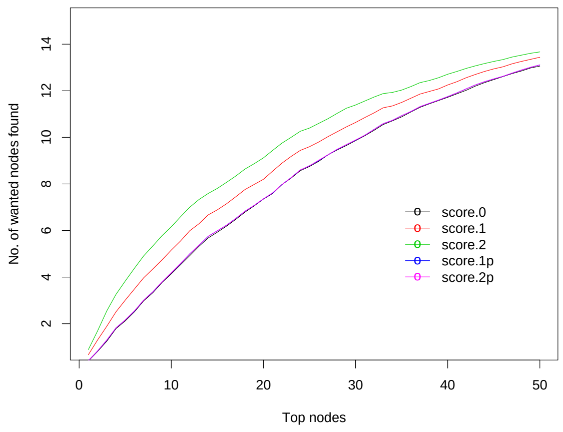


(c) readjust method

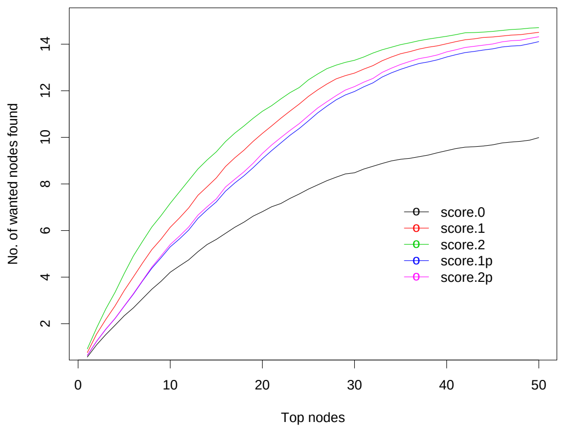


(d) weight method

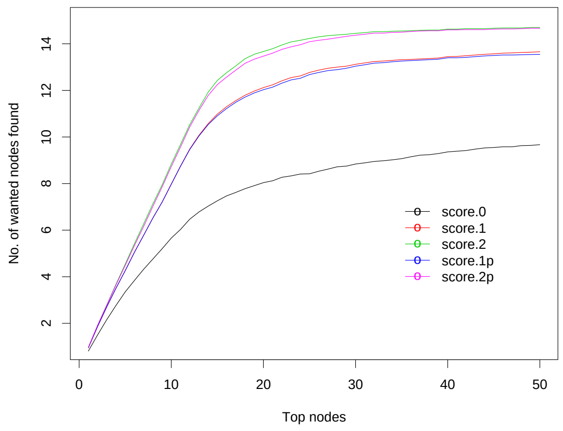
Figure 5.25: *The effect of the parameters on the performance of the algorithms.*



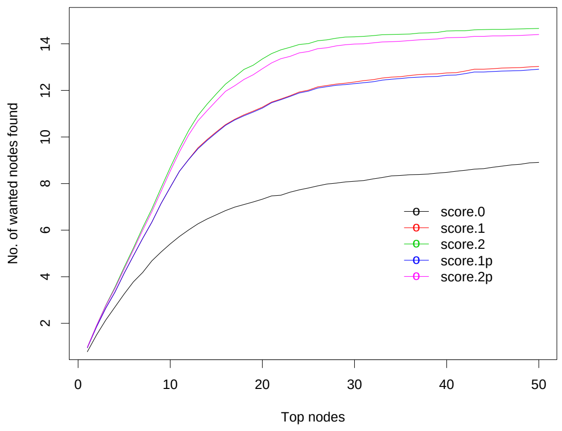
(a) classic method



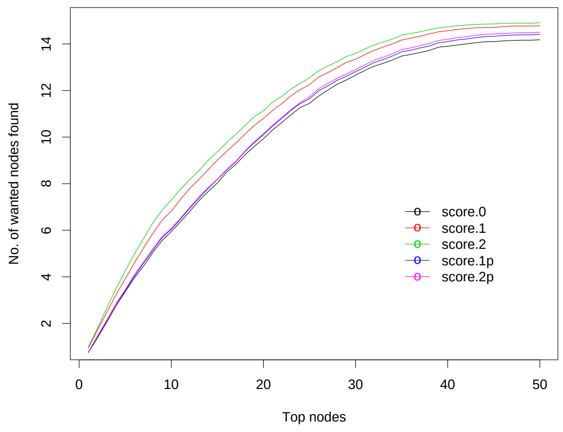
(b) topo method with $\mathsf{cutOff} = 0.01$



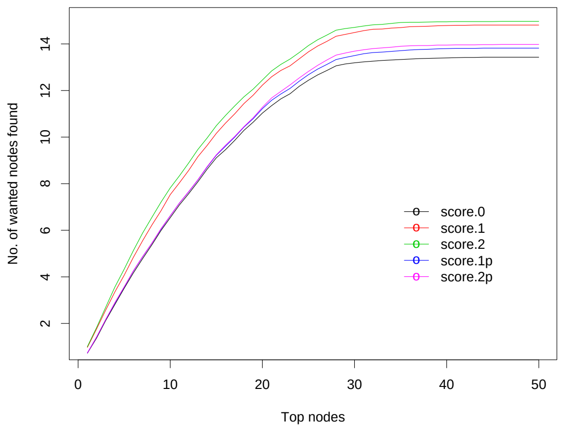
(c) elim method with $\mathsf{cutOff} = 0.01$



(d) readjust method with $\mathsf{cutOff} = 0.01$

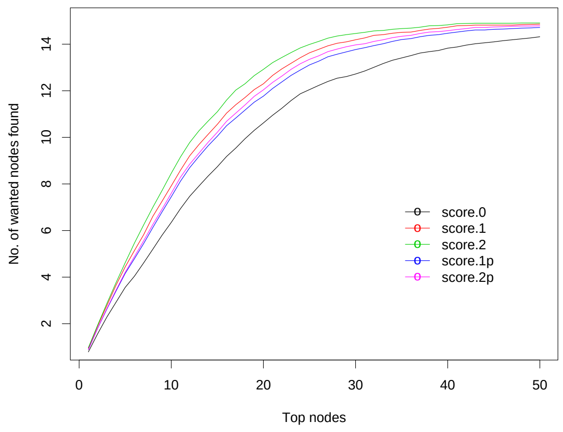


(e) weight.log method

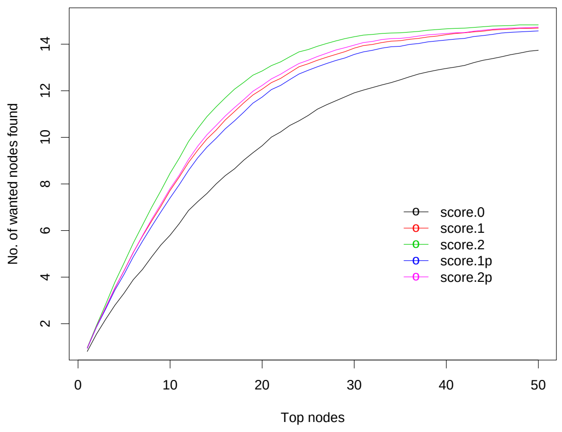


(f) weight.ratio method

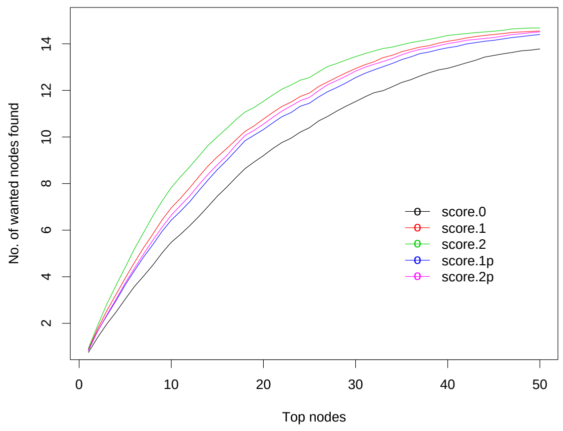
Figure 5.26: *The impact of the scores on the methods (part I)*



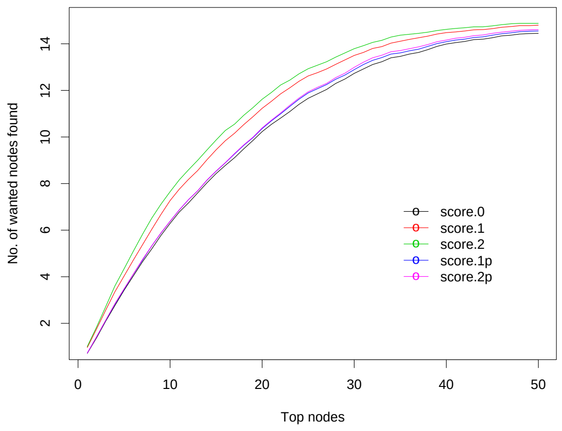
(a) all.M method



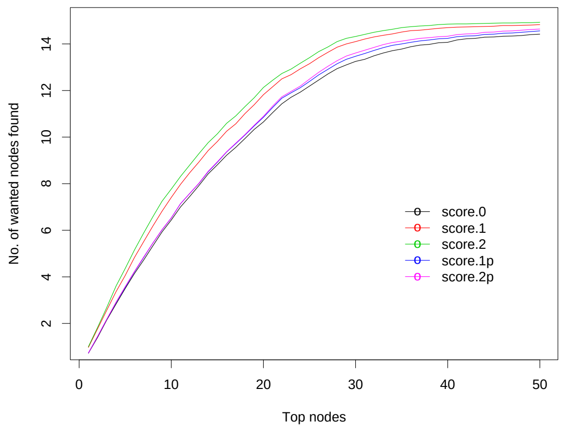
(b) EW average



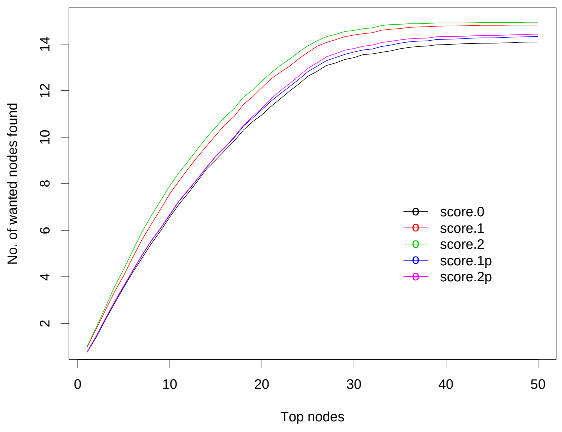
(c) CEW average



(d) CWWlWs average

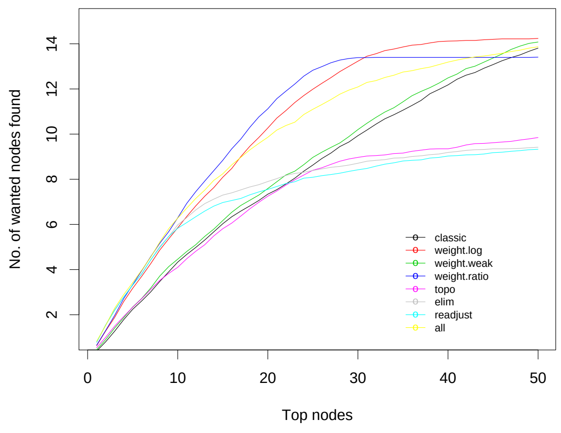


(e) WWlWs average

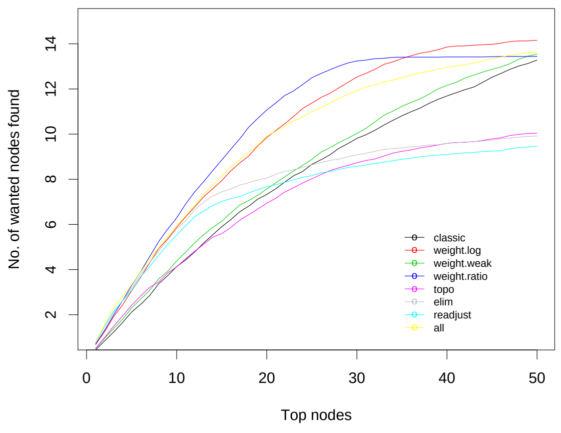


(f) WlWs average

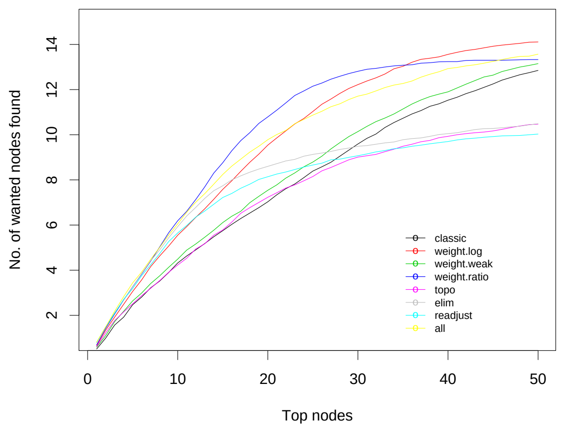
Figure 5.27: *The impact of the scores on the methods (part II)*



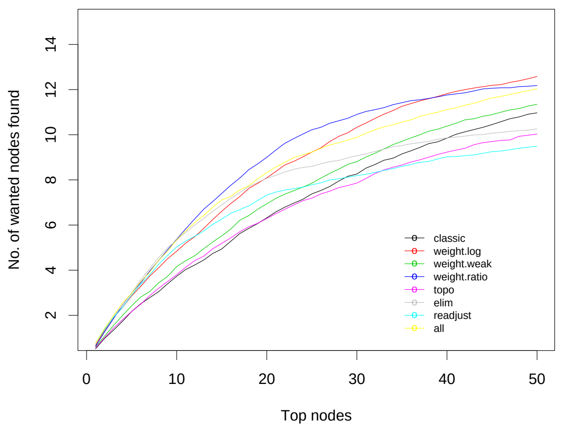
(a) No noise



(b) 10% noise



(c) 30% noise



(d) 50% noise

Figure 5.28: *The effect of the noise introduced in the list of interesting genes. There are 15 wanted nodes with containing 10 to 20 items. Plotted are the average scores over 100 simulations.*

Chapter 6

Conclusions and future work

In this thesis we analyze the problem of finding enriched functional groups of genes based on gene expression data. Identifying the most relevant functional groups for an experiment gives the researcher more insight into the underlying biology. Current methods score predefined functional groups without taking into account the relations between these groups. If the Gene Ontology is used to form functional groups, then more biological knowledge is used in the analysis if the algorithms account for the relationships between the GO terms.

6.1 Conclusions

Our research shows that accounting for the dependences between functional groups is not a trivial task. We proposed several heuristics for integrating the DAG structure of the GO in testing for group enrichment. The key idea is to compute the significance of a GO term based on its neighborhood. Different strategies for this are presented in Chapter 4. The experimental results from Chapter 5 show that we manage to improve over current methods.

Another difficulty in such an analysis is the evaluation of the results. There is no clear measure to tell which method performs better on a real dataset. The evaluation is done by the researcher and thus contains a certain amount of subjectivity. To eliminate this subjectivity we evaluate the methods on simulated data. However there are also some problems with simulated data. Given the complicated topology of the GO and the way the genes are annotated to the GO terms, the wanted nodes induce other significant nodes (possibly 'more significant' nodes). Secondly, adding noise can be misleading. The genes from some wanted nodes can be replaced with other genes, thus making the nodes not significant any more. Although simulated data are not perfectly representing real data, we believe that simulated data are a big challenge. Our methods seem to be more robust to noisier data than the current methods.

We should be aware of the fact that it is hard to beat the classical approach in which each node is scored independently. The reason is that this simple approach is controlling the false negative rate. This means that more nodes are reported as significant but there are very few truly significant nodes that are undiscovered by the method. Our aim was to reduce the false positive rate. Algorithm 4 manages to reduce this rate and at the same time keeps the false negative rate low. On the other hand Algorithm 2 further reduces the false positive rate, but the false negative rate increases.

Thus a compromise must be made. If the researcher is interested in finding the important areas in the graph, then Algorithm 2 is to be preferred. Investigating the neighborhood of the obtained nodes can significantly improve the result of the inference. For this, visualization tools as the ones presented in Section 5.2.1 can be very useful.

There are some issues that are not considered by all presented algorithms. The most important issue is that the algorithms look only at the counts in a node. Without any assumption on the list of interesting genes there is no other information that we can use. If the interesting genes are differentially expressed genes than the method should use this knowledge. By ignoring the distribution of the genes in the node, nodes having the same amounts of genes mapped to them are receiving the same score. For example suppose we are in the setup of analyzing differentially expressed genes and suppose that there are two nodes A and B, with 100 genes mapped to them. For both of them we can map 10 interesting genes. For node A the 10 genes are the most significant genes found on the microarray. For node B the 10 genes are the genes with the smallest significance in the list of interesting genes. It seems natural to say that node A is more significant than node B. By looking only at the counts both nodes are equally significant.

All in all we sow that integrating the dependences between the nodes is enhancing the inference. Moreover, we can combine the results of multiple methods, even further improving the result of the analysis.

6.2 Future work

In this section we briefly outline the future direction we want to take related to the topics discussed in this thesis.

We will focus on implementing more sophisticated weighting schema for Algorithm 4. A better understanding of the dependences between the nodes can be useful in this direction. Given that the same gene can be annotated at different GO terms in the GO hierarchy we must take into account the path of the nodes at which the genes were originally annotated. By considering this, the weighting can be more precise. Another direction is to make the algorithms capture more global behavior. In our experiments Algorithm 4 was using a simple function recomputeSig( ). Tuning this function can improve the results.

We want to use other statistical tests, in addition to Fisher's Exact test, especially non-parametric tests that account for the distribution of the genes in a GO term. The Kolmogorov-Smirnov test seems to be a feasible choice. Trying to find a theoretical model for this type of inference is our primary goal.

Apply the methods to more microarray datasets other than **ALL** can give us more insight into the problems of each algorithm and thus help to improve the algorithms.

The algorithms presented in this thesis are not limited to microarray data. As stated in Figure 1.1, any other type of (biological) data that can be mapped to some ontology can be scored using these methods. Thus, we are interested in applying the methods in different areas. The result of the algorithms is a score for each node in the graph. These scores can be used as input for other types of analysis.

We also plan to develop the R software that was used in this thesis. We want to provide an easy and flexible tool for the analysis of the GO and other ontologies. Visualization tools that simplify the interpretation of the results are very important for such analysis. Using interactive tools can improve the analysis of the results, and at the same time the development of the algorithms.

Bibliography

Al-Shahrour, F., Díaz-Uriarte, R., and Dopazo, J. (2004). FatiGO: a web tool for finding significant associations of Gene Ontology terms with groups of genes. *Bioinformatics*, 20:578–580.

Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, M. J., Davis, A. P., Dolinski, K., Dwight, S. S., Eppig, J. T., Harris, M. A., Hill, D. P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J. C., Richardson, J. E., Ringwald, M., Rubin, G. M., and Sherlock, G. (2000). Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25. http://genetics.nature.com.

Bairoch, A. and Apweiler, R. (2000). The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000. *Nucleic Acids*, 28:45–48.

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.

Beissbarth, T. and Speed, T. (2004). GOstat: Find statistically overrepresented Gene Ontologies within a group of genes. *Bioinformatics*, 1(1):1–2.

Benjamini, Y. and Hochberg, Y. (1995). Controling the False Discovery Rate - a new powerful approach to multiple testing. *JRSS B*, 57:289–300.

Benjamini, Y. and Yekutieli, D. (2001). The control of the false discovery rate in multiple testing under dependency. *Annals of Statistics*, 29(4):1165–1188.

Boyle, E. I., Weng, S., Gollub, J., Jin, H., Botstein, D., Cherry, M., and Sherlock, G. (2004). GO::TermFinder–open source software for accessing Gene Ontology information and finding significantly enriched Gene Ontology terms associated with a list of genes. *Bioinformatics*, 20(18):3710–3715. Applications Note.

Brazma, A. (2003). ArrayExpress-a public repository for microarray gene expression data at the EBI. *Nucleic Acids Research*, 31(1):68–71.

Brazma, A., Robinson, A., Cameron, G., and Ashburner, M. (2000). One-stop shop for microarray data. *Nature*, 403:699–700. www.nature.com.

Breitling, R., Amtmann, A., and Herzyk, P. (2004). Graph-based iterative Group Analysis enhances microarray interpretation. *BMC Bioinformatics*, 5.

Brown, P. O. and Botstein, D. (1999). Exploring the new world of the genome with DNA microarrays. *Natural Genetics*, 21:33–37.

Camon, E., Magrane, M., Barrell, D., Lee, V., Dimmer, E., Binns, D., Maslen, J., Harte, N., Lopez, R., and Apweiler, R. (2004). The Gene Ontology Annotation (GOA) database: sharing knowledge in Uniprot with Gene Ontology. *Nucleic Acids Research*, 32:D262–D266.

Casella, G. and Berger, R. L. (2001). *Statistical Inference.* Duxbury, second edition.

Cheng, Y. and Church, G. (2000). Biclustering of gene expression data. pages 93–103. ISMB'00, AAAI Press.

Chiaretti, S., Li, X., Gentleman, R., Vitale, A., Vignetti, M., Mandelli, F., Ritz, J., and Foa, R. (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.

Consortium, G. O. (2001). Creating the Gene Ontology Resource: Design and Implementation. *Genome Research*, 11:14251433. Cold Spring Harbor Laboratory Press.

Consortium, G. O. (2004). The Gene Ontology (GO) database and informatics resource. *Nucleic Acids Research*, 32:D258–D261. Oxford University Press.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2003). *Introduction to Algorithms.* The MIT Press, second edition.

Doniger, S. W., Salomonis, N., Dahlquist, K. D., Vranzian, K., Lawlor, S. C., and Conklin, B. R. (2003). MAPPFinder: using Gene Ontology and GenMAPP to create a global gene-expression profile from microarray data. *Genome Biology*, 4(1):R7.

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

Dubitzky, W., Garnzow, M., Downes, S. C., and Berrar, D. P. (2003). Introduction to Microarray Data Analysis. In Berrar, D. P., Dubitzky, W., and Garanzow, M., editors, *A Practical Approach to Microarray Data Analysis*, pages 1–46. Kluwer Academic Publishers.

Dudoit, S. and Fridlyand, J. (2003). Introduction to Classification in Microarray Experiments. In Berrar, D. P., Dubitzky, W., and Garanzow, M., editors, *A Practical Approach to Microarray Data Analysis*, pages 1–46. Kluwer Academic Publishers.

Dudoit, S., Shaffer, J. P., and Boldrick, J. C. (2003a). Multiple Hypothesis Testing in Microarray Experiments. *Statistical Science*, 18(1):71–103.

Dudoit, S., van der Laan, M. J., and Pollard, K. S. (2003b). Multiple Testing. Part I. Single-Step Procedures for Control of General Type I Error Rates. Technical Report 138, U.C. Berkeley Division of Biostatistics Working Paper Series.

Dudoit, S. and Yang, Y. H. (2003). Bioconductor R packages for exploratory analysis and normalization of cDNA microarray data. In Parmigiani, G., Garrett, E. S., Irizarry, R. A., and Zeger, S. L., editors, *The Analysis of Gene Expression Data: Methods and Software*. Springer, New York. (To appear).

Edgar, R., Domrachev, M., and Lash, E. A. (2002). Gene Expression Omnibus: NCBI gene expression and hybridization array data repository. *Nucleic Acids Research*, 30(1):207–210.

Friedman, N. and Kaminski, N. (2002). Statistical methods for analyzing gene expression data for cancer research. *Ernst Schering Res Found Workshop*, 38:109–131.

Ge, Y., Dudoit, S., and Speed, T. P. (2003). Resampling-based multiple testing for microarray data analysis. Technical Report 633, University of California, Berkeley.

Gentleman, R. (2004a). Basic GO Usage. Bioconductor Vignettes.

Gentleman, R. (2004b). Some Perspectives on Statistical Computing. Unpublished Manuscript.

Gentleman, R. (2004c). Using GO for Statistical Analyses. Bioconductor Vignettes.

Gentleman, R. (2004d). Visualizing and Distances Using GO. Bioconductor Vignettes.

Gentleman, R. and Carey, V. (2002). Bioconductor. *R News*, 2(1):11–16.

Gentleman, R. and Carey, V. (2003). Visualization and annotation of genomic experiments. In Parmigiani, G., Garrett, E. S., Irizarry, R. A., and Zeger, S. L., editors, *The Analysis of Gene Expression Data: Methods and Software*. Springer, New York. (To appear).

Gentleman, R. and Ihaka, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5:299–314. www://www.r-project.org.

Gentleman, R. and Lang, T. D. (2004). Statistical Analyses and Reproducible Research. Unpublished Manuscript.

Gentleman, R. and Whalen, E. (2004). How To use the graph package. Bioconductor Vignettes.

Gentry, J. (2004). HowTo render a graph using Rgraphviz. Bioconductor Vignettes.

Golub, T. R., Slonim, D. K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J. P., Coller, H., Loh, M. L., Downing, J. R., Caligiuri, M. A., Bloomfield, C. D., and Lander, E. S. (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286:531–537.

Gruber, T. R. (1993). A Translation Approach to Portable Ontology Specifications. Technical report, Knowledge Systems Laboratory, KNOWLEDGE SYSTEMS LABORATORY, Computer Science Department, Standford University, Standford, California 94305.

Hastie, T., Tibishirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Texts in Statistics. Springer-Verlag, New York, fourth edition.

Hennig, S., Groth, D., and Lehrach, H. (2003). Automated Gene Ontology annotation for anonymous sequence data. *Nucleic Acids Research*, 31(13):3712–3715.

Huber, W., von Heydebreck, A., Sültmann, H., Poustka, A., and Vingron, M. (2002). Variance Stablization Applied to Microarray Data Calibration and to Quantification of Differential Expression. *Bioinformatics*, 18 Suppl. 1:S96–S104.

Irizarry, R. A., Hobbs, B., Collin, F., Beazer-Barclay, D. Y., Scherf, U., and Speed, T. P. (2003). Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics*, 4:249–264.

Jones, D. M. and Paton, R. C. (1999). Toward principles for the representation of hierarchical knowledge in formal ontologies. *Data Knowl. Eng.*, 31(2):99–113.

Krajewski, P. and Bocianowski, J. (2002). Statistical methods for microarray assays. *Applied Genetics*, 43(3):269–278.

Kremer, S. S. (2001). Ontologies for Molecular Biology. *Linkoping Electronic Articles in Computer and Information Science*, 6(21). http://www.ep.liu.se/ea/cis/2001/021/.

Lee, S. G., Hur, J. U., and Kim, Y. S. (2004). A graph-theoretic modeling on GO space for biological interpretation of gene clusters. *Bioinformatics*, 20(3):381–388.

Lehmann, E. L. (1986). *Testing Statistical Hypotheses.* Springer Texts in Statistics. Springer-Verlang, New York, second edition.

Lord, P. W., Stevens, R. D., Brass, A., and Goble, C. A. (2003). Investigating semantic similarity measures across the Gene Ontology: the relationship between sequence and annotation. *Bioinformatics*, 19(10):1275–1283.

Mootha, V. K., Lidgren, C. M., Eriksson, K.-F., Subramanian, A., Sihag, S., Lehar, J., Puigserver, P., Carlsson, E., Ridderstrale, M., Laurila, E., Houstis, N., Daly, M. J., Patterson, N., Mersirov, J. P., Golub, T. R., Tamayo, P., Spiegelman, B., Lander, E. S., Hirschhorn, J. N., Altshuler, D., and Groop, L. C. (2003). PGC-1α-responsive genes involved in oxidative phosphorylation are coordinately downregulated in human diabetes. *Nature Genetics*, 34(3):267–273.

Park, T. (2003). Evaluation of normalization methods for microarray data. *BMC Bioinformatics*, 4(1).

Quackenbush, J. (2002). Microarray data normalization and transformation. *Natural Genetics*, 32:496–501.

Rahnenführer, J., Domingues, F., Maydt, J., and Lengauer, T. (2004). Calculating the statistical significance of changes in pathway activity from gene expression data. *Statistical Applications in Genetics and Molecular Biology*, 3(1):Article 16.

Raychaudhuri, S., Chang, J. T., Imam, F., and Altman, R. B. (2003). The computational analysis of scientific literature to define and recognize gene expression clusters. *Nucleic Acids Research*, 31(15):4553–4560. Oxford University Press.

Raychaudhuri, S., Chang, J. T., Sutphin, P. D., and Altman, R. (2002). Associating Genes with Gene Ontology Codes Using a Maximum Entropy Analysis of Biomedical Literature. *Genome Research*, 12:203–214. www.genome.org.

Setubal, J. and Meidanis, J. (1997). *Introduction to computational molecular biology*. PWS Publishing Company.

Smith, D. A., editor (1997). *Oxford Dictionary of Biochemistry and Molecular Biology*. Oxford University Press.

Speed, T., editor (2003). *Statistical analysis of gene expression microarray data*. Interdisciplinary Statistics. Chapman & Hall.

Staab, S. and Studer, R. (2004). *Handbook on Ontologies*. Springer Verlag, Heidelberg.

Tanay, A., Sharan, R., and Shamir, R. (2002). Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, 18(1):S136–S144.

Tobler, J. B. (2002). Evaluating machine learning approaches for aiding probe selection for gene-expression arrays. *Bioinformatics*, 18:S164–S171.

von Heydebreck, A., Huber, W., and Gentleman, R. (2004). Differential Expression with the Bioconductor Project. Technical report, The Berkeley Electronic Press, http://www.bepress.com/bioconductor/paper7. Bioconductor Project Working Papers.

von Heydebreck, A., Huber, W., Poustka, A., and Vingron, M. (2001). Identifying splits with clear separation: a new class discovery method for gene expression data. *Bioinformatics*, 17 Suppl. 1:S107–114. ISMB 2001.

Westfall, P. H. and Young, S. S. (1993). *Resampling-based multiple testing: Examples and methods for* p*-value adjustment*. John Wiley & Sons, New York.

Yang, Y. H. and Speed, T. (2002). Design issues for cDNA microarray experiments. *Natural Genetics*, 3(8):579–588.

Young, A., Whitehouse, N., Cho, J., and Shaw, C. (2005). OntologyTraverser: an R package for GO analysis. *Bioinformatics*, 21(2):275–276. Applications Note.

Zeeberg, B. R., Feng, W., Wang, G., Wnag, M. D., Fojo, A. T., Sunshine, M., Narasimhan, S., Kane, D. W., Reinhold, W. C., Lababidi, S., Bussey, K. J., Riss, J., Barrett, C., and Weinstein, J. N. (2003). GoMiner: a resource for biological interpretation of genomic and proteomic data. *Genome Biology*, 4(4):R28.