Gene Ontology as a tool for the systematic analysis of large-scale gene-expression data [1]

Masterarbeit

im Aufbaustudiengang Bioinformatik
der Technischen Fachhochschule Berlin
zur Erlangung des akademischen Grades eines
Master of science in applied Bioinformatics
vorgelegt von

Stefan Bentink

02/2003

February 15, 2003

[1] Gene Ontology als Werkzeug zur systematischen Gliederung von Microarray-Daten

Acknowlegements

This work was written as my masters thesis in applied bioinformatics at the Technische Fachhochschule Berlin (TFH). The research was carried out at the Department of Computational Molecular Biology (CMB) at the Max Planck Institute for Molecular Genetics Berlin. I want to thank Prof. Dr. Martin Vingron for this opportunity. He is head of the CMB and director of the MPI for Molecular Genetics. It was a great experience to work in the CMB.

Dr. Rainer Spang, head of the Computational Diagnostics group at the CMB, supervised the research on my thesis. I thank him very much for providing me a very exciting research topic and for supporting me in all aspects of my work. Prof. Dr. Ina Koch, head of the post-gradual and continuing study in bioinformatics at the TFH, has encouraged me to start my work at the CMB. I want thank her for this advice and for supervising my work as second expert.

I thank everybody at the CMB and especially at the Computational Diagnostics group for providing me a very nice, interesting and educational time. I could collect experiences in many aspects. I thank my colleagues for their interest in my research and their help.

Abstract

Approach: Technologies like micro- or macroarrays are the source of various problems concerning analysis and evaluation of the datasets obtained. In the present thesis a functional classification provided by the Gene Ontology is used to group large-scale gene-expression. The problem was: How can the expression levels of all genes in a functional group be combined to a single number (score), and how can this score be tested for differential gene expression of the GO-node? Two different statistical tests are discussed to find an accumulation of induced genes in a GO-node on the one hand. On the other hand, we test for a contamination of GO-node with genes that display moderately increased or decreased expression levels. The tests are applied on GO-scores measuring the level of differential gene-expression in GO-nodes. Two different methods are proposed to calculate a GO-score.

Results: The tests are used to compare expression data derived from two different types of human breast cancer. The first dataset comes from the Estrogen receptor positive type of breast cancer, the second one comes from the Estrogen receptor negative type. The most important finding is, that the method provides insight, that can not be found by gene-wise screens, since the GO-scores are mainly driven by genes which do not indicate statistical significance in a gene-wise multiple testing context. One interesting group is the GO-term mitosis. Differential expression of the genes in this group reflects the differences in the proliferative potential of these breast cancer types. To enlarge the practical use, a graphical user-interface has been implemented which allows the molecular biologist for browsing and structuring results of microarray experiments.

Contents

Chapter 1

Introduction

Since the whole genome sequences of several organisms are available, the parallel execution of expression analysis for multiple genes in a single experiment has become an important technology in molecular biology. The most prominent implementations for large-scale analysis of gene-expression are cDNA-macro- or microarrays and the DNA-chip technology. All these methods measure the mRNA levels for many genes at once, according to the following principle: A labeled mRNA is captured by single-stranded, complementary DNA-probes immobilized on a spot of an array or a DNA-chip. The amount of the mRNA hybridized to the specific spot can be quantified by detecting the amount of label and therefore the amount of the corresponding mRNA. While adding several hundred or thousand different DNA-probes to the arrays or chips, the parallel analysis of many genes can be performed. A DNA-chip from Affymetrix contains short oligo-nucleotides synthesized directly on a class wafer by a proprietary technology [12][22]. A macro- and microarray contains separately synthesized oligo-nucleotides spotted to either a nylon-membrane [11] or a glass-slide [4]. The radioactive labeling used in conjunction with nylon-membranes and the fluorescent labeling combined with Affymetrix DNA-chips are examples for multiple-slide experiments. The comparison of the gene-expression in one tissue-type to the gene-expression in another tissue-type requires two separate samples. The glass-slide microarray technology is a single-slide method. The mRNA from the two different tissue-types are labeled with different fluorescent-dyes (red and green). These probes are hybridized to the same microarray. The ratio of the red and green intensity is a direct measure for differential gene-expression of a particular gene.

The analysis of large-scale gene-expression data raise numerous computational and statistical questions. Starting from the output of a scanner which reads the labeling intensities, image-processing is the first problem while analyzing the data. The data have to be normalized, to compare the expression values of the genes on a single array to each other or to that from another array. Image-processing and normalization can be summarized by the term pre-processing (see [10] for a review). The result from pre-processing is a list of expression values or ratios for every single gene. These values can be used to identify those genes, differentially expressed in different tissue-types. For

that, a statistical test is required to assess the significance of differential expression. A method, based on a gene-wise two sample t-test, is introduced by Dudoit et al. [7]. This test raises an important problem. The significance analysis of microarray-data is a multiple testing problem. Assume that for each gene a statistical test for differential expression is conducted. If one fixes a gene-wise significance level of e.g. $\alpha = 0.05$, on average one in every 20 genes that are actually not differentially expressed will show a p-value below α just by chance. According to the large number of genes represented on a microarray, this may lead to a large number of false positive calls [10]. Different methods are suggested to adjust the p-values according to the multiple-testing problem. For example, multiplying the p-values by n, with n denoting the number of genes represented on the microarray, is known as Bonferroni correction. A less conservative way used by Dudoit et al. [7] is the step-wise adjustment of p-values due to Westfall and Young [16]. Both methods may miss genes, which are differentially expressed. A way to decide whether or not a specific expression difference indicates systematic de-regulation is to look at the functional context of the corresponding gene. One would expect a gene acting in a specific biological process to be de-regulated, if the other members of the whole process are de-regulated, too. Another problem raised by the significance analysis of single genes is, that even if the multiple testing problem can be solved adequately one will often end up with a long list of significantly de-regulated genes. Examining every single gene manually is time-consuming and difficult to carry out. A benefit of grouping genes according to specific biological properties is, that the interpretation of the data is simplified. The approach allows for identifying functional groups particularly attached by differential gene-expression.

The literature describes several methods for grouping the gene-expression data into functional groups. The first attempt linking biological knowledge to large-scale gene-expression data was made by Fellenberg and Mewes [8]. They compare the results from clustering to known metabolic pathways. This method uses un-supervised clustering and compares the results in a second step with biological knowledge. A complementary way to find interesting pathways or functional correlated groups of genes is following the opposite direction: Starting from a known functional related group one examines how well this group is supported by expression data. Several approaches follow this way for analyzing large-scale gene-expression data. Zien et al. [18] introduce an approach to check whether or not several possible glycolysis pathways fit to a time series of 8 microarray-experiments representing the diauxic shift of Saccharomyces cerevisiae [3]. They construct the theoretical pathways from open-reading-frames coding enzymes of the glycolysis and propose three methods to score differential gene-expression in the given pathways. These methods are based on p-value like score indicating the level of differential gene expression at a given time-point (t_i) compared to the reference time-point (t_0). They introduce a gene-score from the mean of the negative logarithms of the sample-by-sample p-values and propose a score for a pathway which is the overall mean of the gene-scores in this pathway. They call it conspicuous score. In a second approach they score, how well the differential expression of the genes in the same pathway is correlated over the time-series. The third score is a combination of their conspicuous and correlation score. They assess the significance of their scores

by comparing them to scores computed for 10000 randomly composed pathways. Zien et al. conclude from their results that the conspicuous score best addresses the overall changes in the gene-expression of a specific pathway. In contrast, they claim that the correlation score seems to identify those cases, if the genes belonging to the same pathway are simultaneously activated. The result of the combined score is the same as those produced by the conspicuous score. Zien et al. suggest, that a dominating conspicuous term in the combined scoring function is responsible for this result.

Zien et al. use their approach to examine how well theoretically constructed pathways fit to the "real world". They are looking only at a single biological process, the glycolysis. However, molecular biologist are interested in extracting a biological processes particularly attached by differential gene-expression from a large set of biologically related groups. The Gene Ontology [2][23] provides a structure that organizes genes into biologically related groups according to three different criteria. It classifies genes and their products due to either the biological process in which they are acting, the molecular function they are able to conduct or the cellular component where they can be found. The Gene Ontology consortium aims to provide a unified vocabulary to describe genes. For that reason, the GO-database is a hierarchical ordered set of terms for describing genes. The descriptions which form the nodes of the hierarchical graph-structure increase in detail as one descends down the hierarchy. The root-node of the graph is the term Gene Ontology. The first level of organization contains the three organizing criteria described above (biological process, molecular function and cellular component). Contributors of the GO-database can annotate genes to the different terms or nodes. If a gene is annotated to a GO-node, it is a member of the ancestors of the node, too. A child-node of the graph can be either a part of or an instance of its parent-nodes. A hexokinase for example is a part of the biological process called glycolysis. However, it is an instance of the molecular function called kinase. The Gene Ontology database is not only a tool that allows for viewing a genome in a well structured way. It provides the possibility to group large-scale gene-expression data into biologically related groups.

An approach making use of the Gene Ontology for analyzing microarray-experiments is that of Pavlidis et al. [13]. In their publication they propose three different methods for scoring differential gene-expression in groups of functionally related genes. They use datasets derived from samples of the brain of different mouse-strains, different samples of human leukemia and yeast-samples derived from several growth-conditions. In the case of the yeast-samples Pavlidis et al. use a yeast-specific database called MIPS yeast catalog [24] to structure the expression data. The MIPS catalog is similar to the Gene Ontology database, however it is specific for yeast. They use the Gene Ontology to perform the analysis of the leukemia and brain data. The first scoring-method they introduce measures how well the genes in a specific GO- or MIPS-group cluster together. The score is the average pair-wise correlation between the genes in the same functional group. The second scoring method they propose reflects the statistical significance of the expression pattern of each gene with respect to the experimental design. For that, Pavlidis et al. calculate a p-value for each gene by

applying the analysis of variance-method (ANOVA) on the gene-specific expression values over the samples. They calculate the so-called experiment score by adding the negative logarithms of the ANOVA-p-values of genes which belong to the same GO- or MIPS-group. The last scoring-method Pavlidis et al. call learnability-score. They calculate a k-nearest neighbor classifier for each GO- or MIPS-group. The score they propose is a p-value derived from a leave-one-out cross-validated error rate. Pavlidis et al. assess the significance of their scores by comparing them to scores calculated for 500.000 randomly constituted functional groups of each size. They claim, that the learnability and the correlation score seems to identify GO-or MIPS-groups containing "housekeeping"-genes. In contrast, the experiment score is suggested to be the method most suitable for identifying those functional groups which reflect the specific biological properties of the different samples.

A second approach using the Gene Ontology to analyze large-scale gene-expression is to search for over-representation of particular GO-nodes in a list of genes. This list may contain either significantly de-regulated genes or genes identified by un-supervised clustering-methods. The FatiGO web-tool of Rámon Díaz-Uriarte [19] for example proposes to assign a biological property to a gene-cluster by searching the most frequent GO-node in the cluster. The MAPPFinder application of Doniger et al. [5] identifies GO-nodes by counting the number of GO-specific genes which are significantly de-regulated. They calculate the significance of the single genes in a previous step by a gene-wise analysis. Doniger et al. introduce a z-score by comparing the real number of significant genes in a functional group to those which is expected by chance. The approaches using gene-lists include only genes identified by a un-supervised clustering or significance analysis applied to the un-grouped microarray. That's why, they may miss the influence of genes which can only be identified in concert by a Gene Ontology driven significance analysis, respectively (e.g. correlation and conspicuous score of Zien et al. [18]). Despite this disadvantages, FatiGO and MAPPFinder are useful visualization tools, which allow for quickly inspecting results.

Previous publications provide several approaches to score differential or correlated expression in groups of functional related genes. The power of a Gene Ontology driven approach is, that slightly de-regulated genes are able to form a significant score as group, while the single genes do not reach a significant level of differential expression. For that reason, these approaches are able to identify de-regulated genes that cannot be found by gene-wise scans. We implement two different scoring-methods to identify GO-nodes. We work with the sum of the negative logarithms of p-values proposed by Zien et al. and Pavlidis et al. as the first GO-score. The second GO-score we propose is similar to the test of Kolmogorv and Smirnov combined with Tukey's higher criticism score [14]. The higher criticism deals with a situation where there are many test of significance (in our case of differential gene-expression) and one is interested in rejecting the joint null hypothesis. Using the different scoring methods, we aim to identify GO-nodes with a significant score without containing any significantly de-regulated genes from a gene-wise screen. The approaches described in the literature obtain a test-statistic for the scores by randomizing the annotation of

the genes to the functional groups. We call this method accumulation test, because it tests for an accumulation of low p-values in a GO-node. Additionally, we randomize the annotation of the microarray-samples to the breast-cancer types. We call this new method contamination test, because it tests for a contamination of a GO-node with differentially expressed genes.

We use expression-data derived from 49 samples of of human breast-cancer and provided by West et al. [15]. Half of the microarray samples have been prepared from estrogen receptor positive (ER+) tumor-cells. The remaining samples have been prepared from estrogen receptor negative tumor (ER−) cells. Expression of the estrogen receptor is important as predictive factor for response to endocrine therapy, for example with tamoxifen. Patients with a ER+-tumors have sightly better survival rates, because they respond to endocrine therapy [9]. We choose the data-set, because it provides us expression-data from two well defined tissue-types. An essential part of this thesis is the comparison of results from the accumulation to that from the contamination test. A prototype of a GO-browser implemented in Java supports the analysis of our results.

Chapter 2

Theory

The Gene Ontology provides a functional classification of genes in a hierarchical way. The root node called Gene Ontology contains all genes annotated to the Gene Ontology. Following the graph towards its branches, the functional groups become smaller and more specific. We are interested in finding levels of differential gene-expression in specific functional groups, represented by a GO-node. For that reason, the genes represented on the microarray have to be assigned to GO-nodes. Methods are required to score a whole node for differential gene-expression. In addition we need significance tests for these scores.

2.1 Representation of the data

A major problem is, that the annotation of genes to GO-nodes doesn't reflect the graph structure of the ontology. The Gene Ontology consortium [23] allows to annotate a gene to any GO-node in any level of the graph. However, a child-node in the Gene Ontology graph is defined to be a member of its parent-nodes. A gene annotated to a child-node has to be annotated to the corresponding parent-node, too. The Gene Ontology consortium doesn't set this rule for people contributing an annotation. We implement a routine copying the "nested" annotation of child-nodes to its parent-nodes. Figure 2.1 gives an example for a possible annotation. In this picture the GO-node GO:1 has no direct annotation. But, according to the graph-structure of the Gene Ontology, GO:1 contains all the genes shown in the example. Figure 2.2 demonstrates a recursive algorithm which copies the annotation of child-nodes to its parent-nodes. It is based on a graph-traversing algorithm in post-order.

Figure 2.3 demonstrates the annotation after algorithm 2.2 has been performed on the data shown in figure 2.1. The data in figure 2.3 are represented by a matrix X of normalized expression-values, with k rows corresponding to the genes annotated to GO-nodes and $n = n_1 + n_2$ columns corresponding to the n_1 samples of the first tissue-type and n_2 of the second tissue-type.

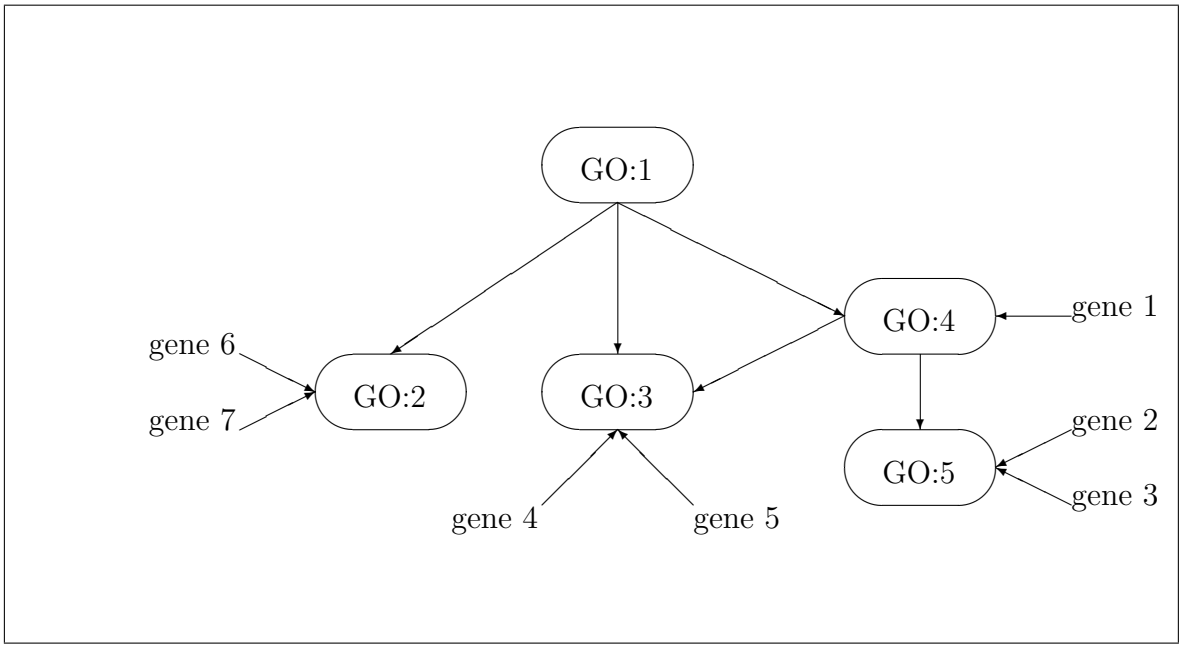

Figure 2.1: Structure of the Gene Ontology with genes annotated to several GO-nodes. Note that child-nodes are members of its parent-nodes. So, the genes annotated to child-nodes belong to the parent-nodes, too (e.g. gene 2 and gene 3 belong to the GO-nodes 5, 4 and 1).

Data : *currentNode*: the GO-node to be annotated
Result : *annotation*: complete annotation directly and downwards to *currentNode*

function *collectSubAnnotation*(*currentNode*)
    annotation ← empty list
    children ← children of *currentNode*
    foreach *children* **do**
        currentChild ← the current child-node
        annotation ← *collectSubAnnotation*(*currentChild*) !Recursion!
        annotation ← genes annotated directly to *currentNode*
    return *annotation*

Figure 2.2: Algorithm to copy the annotation of child-terms to parent-terms

2.2 Scoring

We consider two different scores, to assess differential gene-expression in a GO-node. Both scores are based on a two-sample t-statistic and a gene-wise p-value calculated from the t-distribution. The first scoring function [18][13] sums the negative logarithms of the p-values for the genes in the same GO-node. The second approach is based on Tukey's higher criticism principle [14]. It can be viewed as a Kolmogorov-Smirnov test (KS-test) on the distribution of p-values. We introduce this new scoring

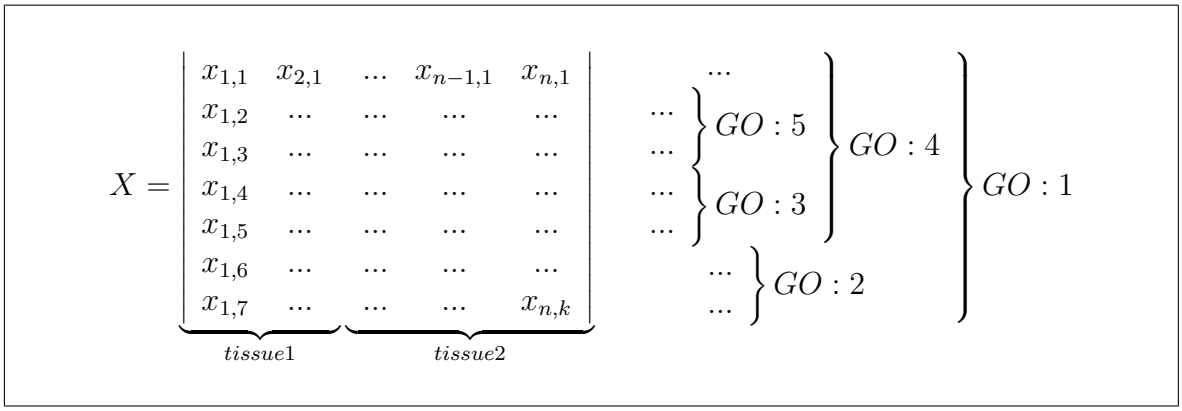

Figure 2.3: Representation of the microarray-data annotated to the Gene Ontology database. The k rows of the matrix X contain the expression values corresponding to the genes. The genes are annotated to different GO-nodes. This is indicated by the brackets to the right of the matrix. The n columns of X contain the expression-values corresponding to different microarray-samples.

method as an alternative to the sum of logarithms. In some cases it is more sensitive than the approach used by Zien and Pavlidis.

2.2.1 T-score

Let H_j denote the null hypothesis of no differential gene-expression between the two tissue-types for a fixed gene $j, j = 1, ..., k$. The alternative representing an individual gene is two-sided. For gene j, the t-score is

$$t_j = \frac{\overline{x}_{n_2} - \overline{x}_{n_1}}{\sqrt{\frac{\frac{1}{n_1}+\frac{1}{n_2}}{n_1+n_2-2}\left(\sum_{i=n_1+1}^{n}(x_{i,j} - \overline{x}_{b_j})^2 + \sum_{i=1}^{n_1}(x_{i,j} - \overline{x}_{a_j})^2\right)}} \tag{2.1}$$

where $\overline{x}_{n_1}$ and $\overline{x}_{n_2}$ denote the average expression level of gene j in the n_1 samples of the first tissue-type and n_2 samples of the second tissue-type. Furthermore, $x_{i,j}$ denotes the expression value of gene j in sample i. The t-score for gene j is denoted by t_j.

Large absolute t-scores suggest that the corresponding genes have different expression levels in the two cancer types. A p-value is calculated from the t-distribution. Note that the t-score only follows a t-distribution in the case of normality of the expression values. In a microarray-experiment this is not necessarily the case. That's why, we do not see the p-value as a real p-value but as part of a scoring function. The p-value is useful for the current calculations for another reason. It normalizes t-values to a number between 0 and 0.5. For normally, independently distributed expression values, one would expect a uniform distribution of the p-values. This is not necessarily the case, but again we will not use these assumptions on the distribution to compute a score from it. Another reason for calculating the p-value-like scores is, that we want

to calculate the sum of logarithms-score. The formula for the p-values obtained from the t-distribution reads as follows:

$$p_j = \begin{cases} 1 - cdf_{df}(t_j) & t \geq 0 \\ cdf_{df}(t_j) & t < 0 \end{cases} \tag{2.2}$$

cdf: cumulative t-distribution function
df: degrees of freedom (number of samples minus 2)

The result so far is a set of p-values per GO-node. The problem is to find scoring methods to summarize them and obtain a score for the node.

2.2.2 Sum of negative logarithms as score

The first scoring function we use was suggested by Zien et al. [18] and Pavlidis et al.[13]. As a cumulative measure Zien et al. calculate the negative sum of logarithms of p-values belonging to the same metabolic pathway. Pavlidis et al. suggest the same function to score differential gene-expression in nodes of the Gene Ontology. Here, we adapt this approach. Following them we define:

$$S_{go\text{-}node} = \sum_{i=0}^{n} \log 2p_i \tag{2.3}$$

Note, the logarithm of one is zero. For that reason the p-values are modified by multiplying them by two. This results in a smaller influence on the score from p-values around 0.5 ($\Rightarrow 2 * p \approx 1$). As a consequence the score is mostly driven by low p-values which suggest differential gene-expression. The summation corresponds to some assumption of independence of genes. This assumption is certainly not correct. However, the sum of logarithms is again seen as part of the score and not as a reliable probability for a false positive error. We use permutations to estimate the distribution of the scores and to assess their significance (see 2.3).

2.2.3 Scoring according to Kolmogorov and Smirnov

The Kolmogorov-Smirnov-test is a statistical method to decide whether or not an empirical distribution fits to a presumed theoretical one. In the case that a theoretical description holds true, the maximal deviation between observed distributions and the theoretical one is scattered according to the KS-distribution. The Kolmogorov-Smirnov function itself is independent of the particular shape of the distribution tested. Here, we apply this property not on the distribution of the data but in a meta-analysis on the distribution of p-values belonging to the same GO-node. This approach is similar to Tukey's higher criticism score [14] which deals with a situation where there are many tests of significance (in our case of differential gene-expression) and one is interested in rejecting the joint null hypothesis (no differential expression of the genes in the same GO-node). The work of D. Donoho and J. Jin [6] gives a detailed

description on the higher criticism in the case of independent test. Donoho and Jin assume, that under the joint null hypothesis the p-values sorted in an increasing order are uniformly distributed. The alternative assumes that there are more low p-values. Here, we also assume a uniform distribution of p-values in the same GO-node. If there is an accumulation of low p-values in a GO-node, the maximum distance to the uniform distribution increases. Hence, this distance is a quantity for differential gene-expression. We use it as GO-score. The actual calculation of the score is implemented as follows: The inspected GO-node contains n p-values. $P(n)$ is defined as the sorted array of the p-values multiplied by two (2.2.1). The uniform distribution is modeled by the array $F(n) = \frac{1}{n}, \frac{1}{n-1}, ..., \frac{1}{2}, 1$. In analogy to the KS-test the score is calculated as:

$$S_{go\text{-}node} = max|P(n) - F(n)| \tag{2.4}$$

Note, the assumptions on the p-values, their distribution and their independence not necessarily hold true. Like for the sum of logarithms score (see 2.2.2), we check the significance of the KS-score against a distribution simulated by permutations (see 2.3).

2.3 Testing the scores

Section 2.2 describes the calculation of two different scores for differential gene-expression in a GO-node. Since we are interested in finding GO-nodes with more differential gene-expression, a test is required to assess the significance of the GO-scores. But, what does more differential gene-expression mean? One answer could focus on the distribution of de-regulated genes. Here we do not question that there are differentially expressed genes at all. What we test for is that these genes do not fall randomly into GO-nodes, but accumulate in the tested GO-node. Another answer does not take the presents of de-regulated genes as a given fact but tests for them de novo, GO-node per GO-node. Note, several moderately induced genes can cause a GO-node to be tested positive, also none of these genes is significant in a gene-wise test. We derive two tests from both definitions of differential gene-expression of a GO-node. We call them **accumulation test** and **contamination test**. The accumulation test proposed by Zien et al. and Pavlidis at al. [18][13] checks for the null hypothesis that there are differentially expressed genes falling randomly into different GO-nodes. The contamination test checks for the null hypothesis, that there is no differential gene-expression in a GO-group. We use permutations to estimate the corresponding test-statistics. Permutations are performed either on the rows of the data matrix X (see figure 2.3), to randomize the composition of the GO-nodes (accumulation test), or on the columns, reflecting the hypothesis of no differential gene-expression at all (contamination test).

The figures 2.4 and 2.5 give a detailed description of the algorithms for calculating the GO-scores and for assessing their significance. There is an important difference between the two algorithms. The algorithm for the accumulation test requires the

Data : Number of permutations: B
n GO-nodes: go_n
Matrix of expression data: $X_{i,j}$ with j rows (genes g_j)
For each GO-node go_n:
R_n: indices of the rows (genes) of $X_{i,j}$ annotated to go_n

Result : p-value for every GO-node $prob_n$

function $performAccumulationTest()$
- **foreach** g_j **do**
  - compute t-statistics t_j
  - compute p-values from t-distribution p_j
- **foreach** go_n **do**
  - compute GO-score S_n from a the subset of p_j with $j \in R_n$
  - $count_n \leftarrow 0$
- **for** $b \leftarrow 1$ ***to*** B **do**
  - permute p_j
    (p-values now maped to random GO-groups)
  - **foreach** go_n **do**
    - compute GO-score U_n from a the subset of p_j with $j \in R_n$
    - **if** $U_n \geq S_n$ **then**
      - $count_n \leftarrow count_n + 1$
- **foreach** go_n **do**
  - $prob_n \leftarrow count_n / B$
- **return** $prob_n$

Figure 2.4: Algorithm to perform the **accumulation test.**

calculation of t-statistics and p-value only once (see 2.2.1). The result is an array p_j of p-values which are ordered like the rows of the data-matrix X. The algorithm performs the permutation not on X but on the array p_j, because the single-gene p-values remain unchanged in the accumulation test, with respect to the permutations. In contrast, the algorithm for the contamination test performs the permutation on the columns of X. It calculates a new p-value for every gene and every permutation. The contamination test algorithm is more time consuming. The calculation of GO-scores is the same, except that the scores are calculated from p-values randomly assigned to GO-nodes on the one hand and from p-values obtained by random assignments of the microarray-samples to tissue-types on the other hand. The algorithms calculate a measure of significance for differential gene-expression of GO-nodes by counting the number of simulated GO-scores (U_n) which are equal or larger than the original GO-scores (S_n). This number relates to the probability of a false positive error. We define the p-value p_n for the GO-score S_n and the corresponding test as the number of simulated scores U_n reaching the original score divided by the number B

Data : Number of permutations: B
n GO-nodes: go_n
Matrix of expression data: $X_{i,j}$ with j rows (genes g_j)
For each GO-node go_n:
R_n: indices of the rows (genes) of $X_{i,j}$ annotated to go_n

Result : p-value for every GO-node $prob_n$

function $performContaminationTest()$
- **foreach** g_j **do**
  - compute t-statistics t_j
  - compute p-values from t-distribution p_j
- **foreach** go_n **do**
  - compute GO-score S_n from a the subset of p_j with $j \in R_n$
  - $count_n \leftarrow 0$
- **for** $b \leftarrow 1$ ***to*** B **do**
  - permute columns i of the data matrix $X_{i,j}$
    (Expression values are now maped to random tissue-types.
    For that reason, the t-scores have to be calculated again:)
  - **foreach** g_j **do**
    - compute t-statistics t_j
    - compute p-values from t-distribution p_j
  - **foreach** go_n **do**
    - compute GO-score U_n from a the subset of p_j with $j \in R_n$
    - **if** $U_n \geq S_n$ **then**
      - $count_n \leftarrow count_n + 1$
- **foreach** go_n **do**
  - $prob_n \leftarrow count_n / B$
- **return** $prob_n$

Figure 2.5: Algorithm to perform the **contamination test.**

of permutations ($p_n = \frac{\#(S_n \leq U_n)}{B}$). This time, we interpret p_n as real p-values in full meaning of the word. Given the null-hypotheses holds true, p_n is the probability to obtain equal or more evidence for accumulation or contamination of GO-node n, than we observe in the score S_n.

The distinction between both testing methods might be confusing at first. But, have a look at the large GO-groups, including the root-node of the Gene Ontology. The contamination test randomizes the assignment of different microarray-samples to the corresponding tissue-types. In the case of differential gene-expression, the gene-wise p-values, calculated for the original assignment of the samples to the tissue-types, tend to be low. In this case, the root-node will always be tested positive by the contamination test, because it contains differentially expressed genes. If the child-nodes inherit

equal portions of differentially expressed genes from the root-node, they will be tested positive by the contamination test, too. Otherwise, if the differential gene-expression is restricted to a particular sub-branch of the Gene Ontology, the contamination test will identify this sub-branch. This is different with the accumulation test. The large GO-nodes contain many genes. The root-node represents the most extreme case. It contains each of the gene having a GO-annotation. Hence, its composition cannot be randomized. The simulated scores U_{root} are equal to the original GO-score S_{root}. The condition $S_{root} \leq U_{root}$ holds true for each simulated score. Hence, the accumulation test will always be negative for the root-node with $p_{root} = 1$. The situation changes, if the GO-nodes become smaller and do not contain all genes having a GO-annotation. There will be simulated scores exceeding the original score and those doing not. Hence, the accumulation and the contamination test are different methods for analyzing GO-specific differential gene-expression.

Chapter 3

Implementation

This chapter deals with the implementation of the statistical analysis. We use the programming language Java [20], that provides us several advantages: The object orientation of Java allows for implementing the Gene Ontology graph in a clear and well structured way. The core application programming interface (API, [20]) of Java provides classes supporting the implementation and visualization of tree like structures given by the Gene Ontology. But, the most important advantage of Java is its independence of the system used to run a program.

We have to solve different problems concerning the implementation of the Gene Ontology approach. We have to represent the GO-graph in a appropriate data structure. We must perform the annotation of the genes on the DNA-chip to the GO-nodes. Furthermore, we have to implement the analysis and evaluation of the gene-expression data according to chapter 2. Last, we want to present the results in a graphical user interface. A complete documentation of our implementation is available under [25].

3.1 The Gene Ontology graph

The first step is to store the graph structure of the GO-database. A graph consists of nodes connected by edges. The Gene Ontology is a directed acyclic graph (DAG) which we can characterize by the following properties: The edges represent directed connections between parent- and child-nodes. A child-node can have more than one parent-node – a property distinguishing a DAG from a tree. While following the graph towards the direction defined by its edges, we can't get back to a node we have already visited. That's what is meant by acyclic. A root-node has no parent-node, so that no in-edge points to it. A leaf-node has no child-nodes or no out-edges, respectively. The Gene Ontology graph contains just one root-node and multiple leaf-nodes.

Our implementation of the Gene Ontology graph follows an object-oriented approach. Objects representing the edges of the graph connect objects representing the nodes. This kind of programming allows for writing the code in a modular way. An object

is characterized by fields storing its data and by methods accessing and changing the content of the fields. In Java objects are implemented by class-files. An object itself is an instance of a class. For example, if a class `GOnode` contains the field `term` describing the GO-node, and the variable `go` stores an instance of the class `GOnode`, then the command `go.term` will return the value of `term`. Another way to access and change the value of `term` could be realized by methods like `setTerm()` and `getTerm()`. These methods would be accessible by the commands `go.setTerm(newName)` and `newName = go.getTerm()`. According to the example, the core implementation of the Gene Ontology graph consists of the classes `GOnode` representing the GO-nodes and `GOedge` representing the edges.

3.1.1 The Java-class `GOnode`

A node of the Gene Ontology itself is characterized by a term describing it and an unique identifier. The position of the node in the Gene Ontology graph is defined by a set of in-edges connecting the node to its parent-nodes and out-edges connecting the node to its child-nodes. We put all these information into the class `GOnode`. Additionally, the class contains optional fields describing additional properties of a GO-node and methods accessing and manipulating all the information stored in it:

Fields of the class `GOnode`:

- The string-variable `term` storing the term describing the node.
- An array of strings we call `goIds[]` storing the unique identifier characterizing every GO-node. It is accessible by the first index of the array (`goIds[0]`). The following elements of the array contain old identifiers, documenting the history of changes of the Gene Ontology.
- An array of strings we call `attributes[]` storing additional information on the node like cross-references to other databases or synonyms of the term describing the node. This field is optional.
- A string-variable we call `group`, to store the name of the sub-branch of the Gene Ontology to which the current node belongs. This is one of the terms `molecular-function`, `biological-process` or `cellular-component` (see chapter 1). This field is optional.
- A hash-table we call `parents`, mapping the identifiers (`goIds[0]`) of the parents of a specific GO-node to objects of the class `GOedge` (see 3.1.2). This field connects a node to its parents.
- A hash-table we call `children`: Similar to the field `parents`, but maps a node to its children, respectively.

- A hash-table we call `directAnnotation`, mapping identifiers for the probes of a microarray (image-ids) to gene-names. This field stores the direct annotation of a GO-node to a microarray (see figure 2.1).
- A hash-table we call `allSubAnnotation`: Similar to `directAnnotation`, but contains, additionally to the direct annotation, the whole set of genes annotated downwards in the Gene Ontology (see figure 2.1).
- A boolean value we call `isInteresting`, storing the information, if a node is significant, according to a GO-score calculated for differential gene-expression (see 2.2).
- A boolean value we call `hasInterestingChildren`, storing the information, if one of the nodes downwards from this node in the Gene Ontology is significant according to section 2.2.

Methods of the class `GOnode`

Except of `isInteresting` and `hasInterestingChildren`, all the fields in the class `GOnode` can be accessed indirectly by methods, we call for example `addId(newId)` or `getId(index)`. This way of accessing and manipulating the fields enables the implementation of additional operations necessary in case a value of a field is changed. For example, if a gene is added to `directAnnotation`, this field is checked for redundancy first. Beside the methods for accessing and manipulating the fields, the class contains the implementation of the algorithm 2.2 which collects the annotation downwards a node (see sections 2.1 and 3.3 for further information). The online documentation of our complete implementation [25] gives an overview over the methods implemented in the class `GOnode`.

3.1.2 The Java-class `GOedge`

The class `GOedge` connects the nodes of the Gene Ontology. It contains three fields, one storing the parent-node (source of an edge), one the child-node (target of an edge) and one storing a literal coding the relationship between a parent- and a child-node (see chapter 1). Similarly to the class `GOnode`, the fields can be accessed indirectly by methods (see online-documentation for details). Only the indirect connection of the nodes of the GO-graph by objects of the class `GOedge` allows for storing the relationship between the parent- and child-nodes. For that reason, we implement the class `GOnode`.

3.1.3 Example

The following example demonstrates the implementation of the data-structure we choose to store the Gene Ontology. The GO-node "axon guidance" is characterized

by its term and two different GO-identifiers. The current identifier is stored at the first position of the array `goIds[]`. The second identifier documents a change in the history of the Gene Ontology database. The node contains only one attribute describing a synonym for the term. It belongs to the Gene Ontology sub-branch "biological process". The node has two parent- and no child-nodes. It is a leaf-node. There are several genes annotated to the node. Since "axon guidance" is a leaf-node, references stored in the hash-table `directAnnotation` are the same than those stored in the hash-table `allSubAnnotation`. Note, the hash-table `allSubAnnotation` stores the references to all genes annotated directly and downwards in the Gene Ontology. The following text-boxes show the contents of the GOnode-object "axon guidance", two string-objects storing the names of the annotated genes and two objects of the type GOedge which connect the node "axon guidance" to its parent-nodes.

GO-node "axon guidance:" Note, the fields `parents`, `children`, `directAnnotation` and `allSubAnnotation` are hash-tables mapping separate objects to key-values. The connection of the parent- and child-nodes is realized by mapping the GO-identifiers of the parent- and child-nodes to separate objects of the type GO-edge (hash-tables `parents, children`). Similarly, the names of annotated genes are mapped to a unique identifier for each gene on a microarray.

GOnode axon guidance:

- `term = "axon guidance";`
- `goIds[] = {"GO:0007411" ; "GO:0008040"};`
- `attributes[] = {"synonym:axon growth cone guidance"};`
- `group = "biological_process";`
- `parents = {"GO:0007409"`$\leftarrow$*in-edge1* `; "GO:0008037"`$\leftarrow$*in-edge2*`};`
- `children = {};`
- `directAnnotation = {"U28369_at"`$\leftarrow$*gene1* `; "M73239_s_at"`$\leftarrow$*gene2*`};`
- `allSubAnnotation = {"U28369_at"`$\leftarrow$*gene1* `; "M73239_s_at"`$\leftarrow$*gene2*`};`

Genes annotated to the GO-node "axon guidance". The genes are referenced by the hash-tables `directAnnotation` and `allSubAnnotation`.

- *gene1* `= "sema domain, immunoglobulin domain";`
- *gene2* `= "hepatocyte growth factor(hepapoietin A; ...)";`

The edge connecting the node "axon guidance" to its parent-node "axonogenesis". Note, the character '<' defines, that the node "axon guidance" is a **part of** the node "axonogenesis".

In-edge1:

- `parent = GOnode axonogenesis;`
- `child = GOnode axon-guidance;`
- `relation = '<';`

The edge connecting the node "axon guidance" to its parent-node "cell recognition". Note, the character '%' defines, that the node "axon guidance" is an **instance of** the node "cell recognition".

In-edge2:

- `parent = GOnode cell recognition;`
- `child = GOnode axon-guidance;`
- `relation = '%';`

3.2 Construction of the GO graph

Section 3.1 gives an overview over the data-structure we choose to store the GO-graph. In this section we describe the implementation of a parser which reads the GO-graph from text-files and constructs the data-structure.

We download the current GO-version of the from the web page of the Gene Ontology Consortium [23]. It is provided in form of three flat-files, one for every sub-branch of the ontology (see chapter 1). The following text shows an excerpt of the file storing the sub-branch `molecular-function`:

```
$Gene_Ontology ; GO:0003673
 <molecular_function ; GO:0003674
  %anti-toxin ; GO:0015643
   %lipoprotein anti-toxin ; GO:0015644
  %anticoagulant ; GO:0008435
  %antifreeze ; GO:0016172
   %ice nucleation inhibitor ; GO:0016173
  %antioxidant ; GO:0016209
```

Every line in the text-file represents a GO-node. The character at the beginning of a line codes the relationship of a child-node to its parent- node (% instance of, < part of, $ root (see chapter 1)). This literal is followed by the term describing the node and its unique identifier. The indentation of the line indicates the level of the node in the GO-graph. In the shown example, the terms `anti-toxin`, `anticoagulant`, `antifreeze` and `antioxidant` are instances of the term `molecular-function` (one of the three major-branches of the GO). The node `molecular-function` itself is a

```
Data   : text-file containing the Gene Ontology
Result : hash-table with objects of GO-nodes
         mapped to unique GO-identifiers and connected by edges

function parseOntology(text-file)
    hash-table ontology ← empty hash-table
    stack currentPath ← empty stack
    foreach line of the text-file do
        currentGoId ← unique GO-identifier from the current line
        if onology contains key currentGoId then
            newNode ← get object from ontology mapped to currentGoId
        else
            newNode ← create new GOnode-object from the current line
            ontology ← newNode mapped to the key currentGoId
        indentation ← indentation of the current line (space-characters)
        currentPath ← trim to size of indentation, by removing elements from the
        top
        newNodesParent ← top of currentPath
        create edge between the newNodesParent and newNode
        currentPath ← put newNode on top
    return ontology
```

Figure 3.1: Algorithm implemented by a parser which reads text-files containing the Gene Ontology. It reads the text-file line by line and creates a new GO-node for every row. A stack stores the GO-nodes which form the path from the root to the new GO-node. The algorithm trims the stack-size to the length of the indentation of the current row. This is performed by removing GO-nodes from the top of the stack. For that reason, the GO-node on the top of the stack always represents the parent of the current GO-node. After cross-linking the parent- and child-node, the new GO-node is put to the top of the stack.

part of the node `Gene-Ontology` which represents the root. A line of the file can contain further information on a GO-node. The GO-web page provides a detailed description on this. We implement a parser that loads the graph-structure into the memory of a computer. Basically this parser constructs line by line an object of the type `GOnode`. For having direct access to the GO-nodes, a hash-table maps the created node-objects to their unique identifier. The parser-algorithm is shown in figure 3.1. After parsing the flat-files, the complete Gene Ontology graph is stored in memory. In a next step, we have to perform the annotation to the probes of a microarray.

3.3 Annotation

3.3.1 Performing the annotation

An essential step is the appropriate GO-annotation for the Affymetrix DNA-chips, since we use data obtained by Affymetrix DNA-chips. Fortunately Affymetrix provides free access to its proprietary databases [22]. This includes the possibility to perform batch-queries. Hence, we can download GO-annotations for microarray-data from the web. Available are files that list the direct annotation of genes to GO-identifiers. For example a transcription factor is annotated to the GO-node transcription factor but not to the corresponding parent-nodes. However, the Gene-Ontology defines child-nodes as members of its parent-nodes (see figure 2.1). To get all the genes that belong to a specific GO-node, the annotation downwards from this node has to be collected. For that purposes we use an implementation of the algorithm described in section 2.1 and figure 2.2. The implementation is accessible as a method of the class `GOnode` (see online-documentation [25]).

3.3.2 Thinning out the GO-graph

We do not have an annotation of a gene represented on the microarray to every GO-node. For that reason, we remove all GO-nodes from the GO-graph, which are neither annotated directly, nor containing a child-node with a gene annotated to it. The implementation of the class `GOnode` contains a recursive method, that looks up for annotation of its children, of the children of the children and so on. Starting from the root of the GO-graph, all the sub-branches are removed, that do not contain genes from the microarray. We call this method `removeNotAnnotatedChildren` (see online-documentation [25]).

3.4 Calculating and testing the GO-scores

One major implementation question is, how to deal with the large data-sets. A solution has been found in form of a Java-API provided by Wolfgang Hoschek [21]. A set of packages belonging to the so called Colt-API allow for creating large matrices and performing complex calculations on them in Java. A main feature of the API is, that views of a matrix can be created by selecting specific rows and columns. These views are new matrices consisting of the selected rows and columns. While creating a view, the values of the matrix remain untouched. They are not copied to another location in memory. The selection returns a reference to the original values. For that reason, a view is rather a new object of the type matrix, but another way to look at the original data. This way of dealing with large data-sets saves time and computing power and is well established in applications like Matlab or R. Its availability in Java

provides us with the possibility to build analysis-tools independent of the platform and any mathematical software. Additionally, the code uses some statistical functions also taken from the Colt-API.

3.4.1 Data-selection

Figure 2.3 demonstrates the representation of gene-expression data. The rows of the matrix X correspond to genes and the columns correspond to different microarray-samples. The expression-values of the genes annotated to a GO-node, can be obtained by the selection of the rows corresponding to the GO-node. The microarray-samples of a specific tissue-type can be obtained by the selection of the corresponding columns of X. The Colt-API provides the Java-class `DoubleMatrix2D` which allows for storing two-dimensional arrays. The class contains the method `viewSelection (int[] rowIndices, int[] columnIndices)`. This method returns a new instance of the class `DoubleMatrix2D`, representing a virtual view on the original data (see above). The rows of the new virtual matrix are those given by the parameter `rowIndices` and the columns are those of the parameter `columnIndices`. For example, the parameter `rowIndices = {4, 3, 2, 1, 1}` returns a new matrix with the rows in reversed order and a second copy of the row with the index 1.

3.4.2 Calculating gene-wise p-values

The viewSelection-method of the Colt-API (see 3.4.1) allows for selecting the columns of X corresponding to the two tissue-types and storing them separately in new matrices X_1 and X_2. Note, permutation of the columns of X results in a new assignment to either the matrix X_1 or matrix X_2. Our implementation of the equations 2.1 and 2.2 calculates a p-value p_j, using row j of matrix X_1 as sample-set 1 and the same row of matrix X_2 as sample set 2. The Java-class `TwoSampleT` implements a classical t-test (see online-documentation [25]). We need a quantile-table of the t-distribution, to calculate a p-value from a t-value. We use a class from the Colt-API called `Probability`. It contains the method `studentT` returning the value of the cumulative t-distribution function for a given t-value and a defined number of degrees of freedom. The implementation of equation 2.1, which calculates the t-value, uses the Colt-class `Descriptive`. This class provides methods to calculate a mean-value, its standard deviation and its variance. The result of calculating the p-values is an array p_j which is ordered like the rows of the data-matrix X.

3.4.3 Calculating the GO-scores

A score for a GO-node is calculated from the p-values of the genes annotated to the GO-node. We get the p-values by selecting them from the array p_j (see above). An

array i_n of integer-values is created for every GO-node. This array contains the row-index for each gene annotated to the GO-node. We create a new array p_n containing the GO-specific p-values using the viewSelection-method (see 3.4.1) with i_n as parameter. From these node-specific p-values, we calculate the scores according to the equations 2.3 and 2.4. Note, permutation of p_j results in a new assignment of the p-values to GO-nodes.

3.4.4 Performing the permutation-tests

We use two different methods to test GO-scores for significance, the accumulation and the contamination test (see section 2.3). The tests are implemented according to the algorithms 2.4 and 2.5. Remember, the accumulation test needs permutation of the array p_j containing a p-value for every gene. The contamination test needs permutation of the columns of the data-matrix X. We implement these permutations as follows:

- Create a random sequence of integer-values ranging from the first to the last index of the matrix which will be permuted (see Algorithm 3.2).
- Select the rows or the columns of the matrix, according to the random sequence of integer-values.
- Take this selection as new data-matrix.

Note, algorithm 3.2 requires a random generator, generating uniformly distributed integer-values. In contrast to the core Java-API, the Colt-API provides several classes implementing such a random generator. We use the Colt-class `Uniform`. Its method `nextIntFromTo (min,max)` applied on an instance of `Uniform` returns a random integer-value between `min` and `max`.

3.5 Visualization of the GO-graph

Since we want to present the results of our Gene Ontology driven microarray-analyzes in a graphical user interface, we have implemented a prototype for a Gene Ontology browser. Figure 3.3 demonstrates its features. The browser-window contains three different frames. The left frame visualizes the hierarchical structure of the Gene Ontology. The lower-right frame contains a table, listing the genes falling into a particular GO-node. This list can be obtained by double-clicking a GO-node in the hierarchy. The upper-right frame contains a detailed description of a particular gene from the gene-list. This information can be obtained by double-clicking a gene in the lower-right table. Additionally, the results from the calculation of the GO-scores are visualized. So, the table on the lower-right contains for each gene the t- and

Data : min: the lowest value of the set of integer-values required
max: the largest value of the set of integer-values required
A random generator returning uniformly distributed random integer-values
Result : v_n: Array containing n integer-values ranging from min to max in a random order

function *integerRangePermutation(min,max)*
  $n \leftarrow (max - min + 1)$
  $b_n \leftarrow$ array of the length n
  $v_n \leftarrow$ array of the length n
  for $j \leftarrow 1$ ***to*** n **do**
    $b_j \leftarrow min$
    $min \leftarrow (min + 1)$
  for $i \leftarrow 1$ ***to*** n **do**
    generate random integer $g, g = 0, 1, ..., (n - i - 1)$
    $rand \leftarrow (i + g)$
    $v_i \leftarrow b_{rand}$
    $b_{rand} \leftarrow b_i$
    $b_i \leftarrow v_n$
  return v_n

Figure 3.2: Algorithm that generates a random permutation of integer-values ranging from min to max.

p-score calculated according to the equations 2.1 and 2.2. The significance of either the accumulation or the contamination test is color-coded in the hierarchy-view. GO-nodes with a significant score are colored in red. The level of the significance is indicated by a continuous scale from red to green. While browsing the results, one might be interested in the question, if a particular GO-node has child-nodes with a significant score. This information is provided by a small file-icon which is added to a GO-node in the hierarchy-view, if it has significant child-nodes.

The implementation of the GO-browser makes use of an important feature of the Java-API. The class `JTree` automatically creates a file-system like view of hierarchical structures. The structure must be provided by an interface called `TreeModel`. Note, a Java-interface is a class-like file which only defines names and parameters of methods. A class implementing an interface has to contain the concrete code for these methods. The interface `TreeModel` defines the methods, required to visualize hierarchical structures. These are for example the methods `getChild(Object node, int index)`, `getParent(Object node)` and `getRoot()`. So, our GO-browser contains a class which implements the interface `TreeModel`. We call this class `GeneOntology`. It transforms our representation of the Gene Ontology to the representation required for visualization by the class `JTree`. The core composition of the Gene Ontology browser contains the following classes (see online-documentation [25]):

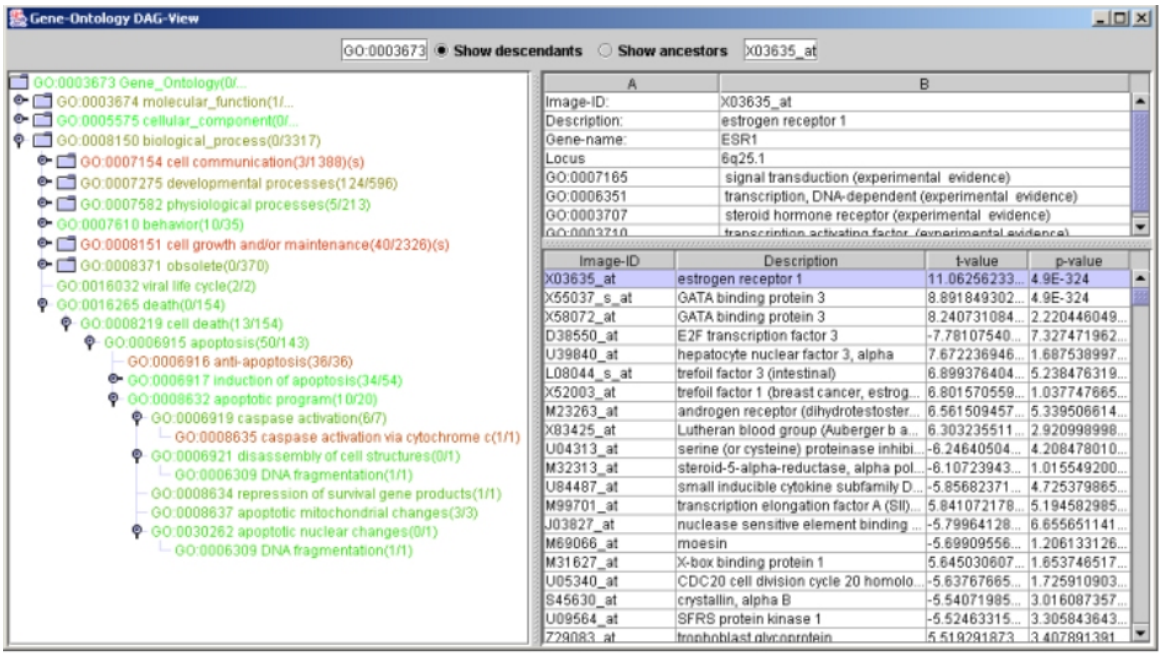

Figure 3.3: *Prototype of a Gene Ontology browser.* The left frame visualizes the Gene Ontology. The significance of the GO-nodes according to the accumulation or the contamination test is color-coded by a continuous red/green scale (red: significant). GO-nodes containing children with a significant score are marked by a small file-icon. The upper-right frame gives detailed information on a particular gene. The lower-right frame shows a table with genes that fall into a particular GO-node.

- `class GOnode` and `class GOedge`: Representation and connection of the GO-nodes.
- `class GeneOntologyTree extends JTree`: Visualization of the Gene Ontology. The class extends the class `JTree`. It inherits all methods and fields from the class `JTree`.
- `class GeneOntology implements TreeModel`: Provides the graph-structure for the `GeneOntologyTree` by implementing the interface `TreeModel`

Chapter 4

Results

For testing our Gene Ontology approach, we analyze expression-data derived from two different types of human breast-cancer using Affymetrix Human GeneFL genechip DNA arrays. We compare 25 estrogen receptor positive (ER+) tumor-samples to 24 estrogen receptor negative (ER−) ones (data provided by [15]). There are significant differences between the two types of breast-cancer concerning their response to endocrine therapy, for example with tamoxifen interrupting the function of the estrogen receptor. There is active research on the role of the estrogen receptor in human breast-cancer. Many genes are known to be differentially expressed in ER+ and ER− tissues[9]. We hope add to the picture by structuring these genes according to the Gene Ontology and to possibly reveal additional subtle differences by GO-node based scoring. We use the scoring- and testing-methods described in chapter 2. The current chapter presents a comparative analysis of the accumulation- and contamination-test as well as the two scoring methods.

4.1 Annotation

The Affymetrix Human GeneFL genechip DNA array has 7129 genes represented on it. We obtain a GO-annotation for 4081 of the genes from the Affymetrix web-page [22]. 2641 GO-nodes contain at least one GO-annotated gene after copying the annotation of the child-nodes to their parent-nodes (see section 2.1). We perform the following calculations only on the subset of 4081 annotated genes. So, we have a data-matrix X containing $n = 49$ columns each representing a microarray-sample and $k = 4081$ rows, each representing a gene.

4.2 Analysis

We calculate a p-value-like score for each of the 4081 genes, according to section 2.2.1. We us these p-values to obtain sum of logarithms- and KS-score for each of

the 2641 annotated GO-nodes. We repeat the calculation 1000 times with random permutations of either the rows (accumulation test) or the columns (contamination test) of the data-matrix X. The result is a set of 1000 random scores for each kind of scoring and permutation. We consider a GO-score to be significant if it is exceeded by less than 5 percent of the corresponding random scores. This corresponds to a significance level of 0.05.

4.3 Comparing the tests

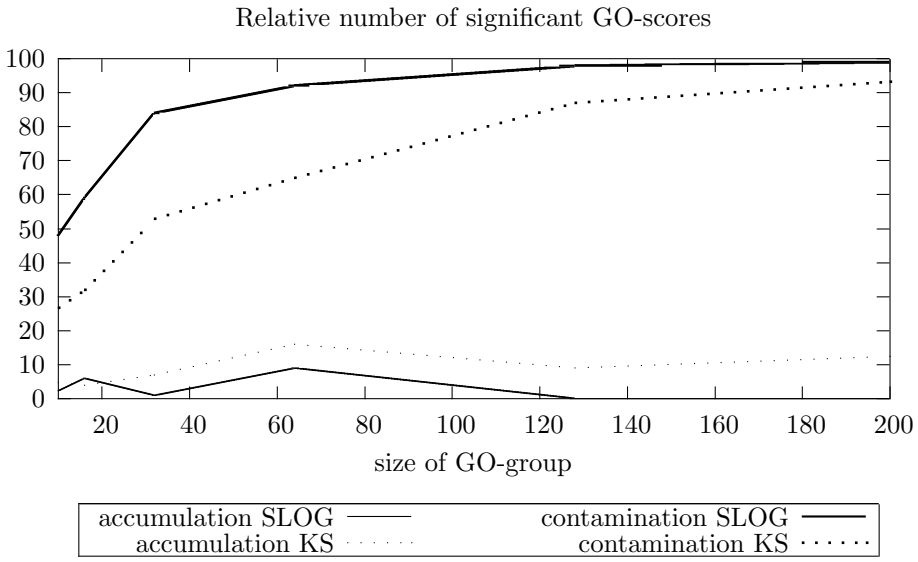

Figure 4.1: *Relative number of significant GO-groups in percent.* The plot shows the results from the accumulation and the contamination test applied to the KS-score (KS) and the sum of logarithms-score (SLOG). The null-hypothesizes are rejected at a p-value larger 0.05. Only the results for GO-groups containing more than 10 and less than 200 genes are shown.

The contamination test strongly depends on the size of the GO-nodes and returns much more significant nodes than the accumulation test. Figure 4.1 should demonstrates the different results. It shows the relative number of significant GO-nodes according to the different scoring-methods and tests, plotted against the size of the GO-nodes. We expect such a result (see section 2.3). The contamination test checks for each GO-node separately, whether there are differentially expressed genes. The larger the GO-node is, the more de-regulated genes are found in it. If a child-node is significant, its corresponding parent-nodes are expected to be significant, too. The root-node is always positive, if there is any difference in gene-regulation at all. The accumulation test checks whether the known differentially expressed genes accumulate in a GO-node. Figure 4.2 visualizes that the results of the tests depends on the size of the inspected GO-groups. It shows two windows of our Gene Ontology browser.

The left frame of both windows provides a tree-like view on the Gene Ontology. It shows a complete path of the Gene Ontology leading from the root-node to the node "apoptotic program". The text beside each node is colored. This color codes the significance of each node. GO-nodes with a significant score are red, those which are not have a green color. The level of the significance is indicated by a continuous scale from red to green. The upper window visualizes the result of the accumulation test, the lower that of the contamination test. The accumulation test returns few significant GO-nodes in different levels of the hierarchy. The contamination test returns significant GO-nodes mainly in the higher levels. Following the graph by its branches, the number of significant GO-nodes decreases. The GO-nodes downwards the "apoptotic program"-node contain no differentially expressed genes at all. The whole branch is unaffected by differential gene-expression. Hence, the contamination test discovers, if branches of the GO-graph are affected by differential gene-expression. This is a particular property of the contamination test. The identification of new drug-targets maybe a possible application of this property. For example, the response to endocrine therapy is more successful in patients suffering from the ER(+) breast-cancer type [9]. This can be explained by the differential expression of the estrogen receptor which is target of the therapy. The treatment of a biological process that is equally regulated may provide an appropriate therapy effecting on both subtypes of human breast-cancer. A candidate might be for example the process "apoptotic program" containing 20 genes which are 20 possible new drug-targets.

4.4 The scores and the single genes

4.4.1 Excluding the significant genes

Our GO-approach allows for discovering genes that are differentially expressed as functional group. However, does this provide new insight or is it just the summary of the results we can also obtain from gene-wise screening? Are the scores mainly driven by genes which can also be found by a significance analysis applied to single genes? We address this question and exclude all the genes from our analysis which are differentially expressed at a level, that could be detected by a gene-wise procedure.

We use the gene-wise p-values (see section 2.2.1) for a rough single-gene significance analysis, although they should not be interpreted as meaningful p-values. We obtain a list roughly representing the rank-order of differential expression by sorting the genes by their corresponding p-values. The single-gene null-hypothesis is rejected at a significance level of $\alpha = 0.05$. Since the microarray-analysis represents a multiple-testing problem, we adjust this significance level according to Bonferroni to $\alpha^* = \alpha/n$ with n denoting the number of genes to be analyzed (4081 in our case). We consider genes as significantly de-regulated, if their p-value is lower or equal than α^*. According to this condition 68 of the 4081 GO-annotated genes are differentially expressed. We compute new scores modified by excluding the significant genes from our analysis.

Figure 4.2: *The path of the Gene Ontology graph leading to the node "apoptotic program".* The p-value according to either the accumulation test (top) or the contamination test (bottom) applied on the sum of logarithms score is coded by a red-green ratio (red low p-value, green large p-value).

First, we apply the accumulation test on modified KS-scores. 81.5 percent of the significant GO-nodes we obtain by this new analysis are identical to those identified without modification of the score. The contamination test applied on modified and not modified KS-scores returns 82.3 percent identical GO-nodes. Hence, the KS-score is

independent of genes which can also be found by gene-wise-screens. The GO-approach can identify slightly de-regulated genes.

4.4.2 Interesting GO-nodes

We use our GO-browser to inspect the GO-scores manually and to identify interesting GO-nodes according to the breast-cancer data-set. We can show that our GO-approach provides insight, that can not be found by gene-wise screens. The GO-node "complement component" (unique GO-identifier: GO:0003811) is a very interesting example, because it contains no gene with a significant single-gene p-value (see table 4.1). But, the KS-score for this GO-node reaches significance according to the accumulation and the contamination test. Hence, our GO-approach can filter useful information from microarray-experiments. This information maybe completely concealed from gene-wise screens. The GO-node "complement component" contains genes important in the immune system. Currently, we have no possible biological interpretation for the significance of this GO-node.

We can identify another interesting GO-node called "mitosis" (see table 4.2). It contains 33 genes responsible for the regulation and the performance of the cell-division. Only one of these genes can be detected by gene-wise screens. The score for the "mitosis"-node remains significant, even if this gene is removed from the analysis. The identification of the GO-node can be explained biologically. The ER(−)-type is more aggressive according to the proliferation of the tumor-cells [9]. So, one would expect a differential expression of the mitosis-genes. We use the descriptions of the 10 genes listed in table 4.2 to query the PubMed database [26]. The query for the PLK-gene returns reference to a publication of Wolf et al. [17]. Wolf et al. could show by immunohistochemistry, that the expression of the PLK-gene (polo-like kinase) is different in ER(+)- and ER(−)-cells. We can show the differential expression of the PLK-gene, too. But, we can show this only by using the GO-approach. A gene-wise microarray-analysis cannot identify the differential expression of the PLK-gene.

4.5 The scoring methods

We propose two different methods to combine the expression levels in a GO-node to a single number. These are the sum of logarithms- and the KS-score. The results which we obtain from the methods are similar but they do not exactly return the same GO-nodes. Currently, we cannot explain the reason for the different results. However, we can fix from excluding the significant genes, that the KS-score seems to be more robust against strongly de-regulated genes. Additionally, it produces more significant scores, if we use the accumulation test.

Image-ID	**Bonferroni adjusted** p**-value** $p^* = p * n$	**gene description**
J15702_at	0.29	B-factor, properdin
M84526_at	0.63	D component of complement (adipsin)
J04080_at	36.82	complement component 1, s subcomponent
M16973_at	186.02	complement component 8, bet polypeptide
M83652_s_at	261.76	properdin P factor, complement
M13232_s_at	264.09	coagulation factor VII
M14058_at	309.25	complement component 1, r subcomponent
X02176_s_at	360.92	complement component 9
K02766_at	1645.78	complement component 9
M65134_at	2441.36	complement component 5
J03507_at	3570.63	complement component 7

Table 4.1: List of genes falling into the the GO-node "complement component". The genes are ordered by their level of differential expression. The second column contains the Bonferroni-adjusted gene-wise p-value $p^* = p * n$ with $n = 4081$ denoting the number of spots on the microarray.

Image-ID	**Bonferroni adjusted** p**-value** $p^* = p * n$	**gene description**
M86699_at	0.001	TTK TTK protein kinase
U30872_at	1.114	CENP-F kinetochore protein mRNA
U63743_at	1.19	Mitotic centromere-associated kinesin mRNA
U01038_at	2.28	PLK mRNA
Z15005_at	2.86	CENPE Centromere protein E (312kD)
S78187_at	7.18	M-PHASE INDUCER PHOSPHATASE 2
X89109_s_at	14.12	MacMarcks mRNA
X51688_at	34.89	CCNA Cyclin A
D21262_at	44.89	KIAA0035 gene, partial cds
U49070_at	92.23	Peptidyl-prolyl isomerase and ...
...	...	...

Table 4.2: List of genes falling into the the GO-node "mitosis". The genes are ordered by their level of differential expression. The second column contains the Bonferroni-adjusted gene-wise p-value $p^* = p * n$ with $n = 4081$ denoting the number of spots on the microarray. Only 10 out of 33 genes are shown.

Chapter 5

Discussion

In the present thesis we approach large-scale gene-expression data from a higher level of organization. We use Gene Ontology providing a hierarchical, functional classification of genes. Since we want to combine the expression levels of genes in the same GO-node to a single number, we propose two different scoring-methods. We check their significance using two different tests. We use a data-set derived from two different classes of human breast-cancer, the estrogen receptor positive and estrogen receptor negative class. We can show by three examples, that the GO-approach provides insight, that can not be found by gene-wise screens of microarray-data (GO-nodes apoptotic program, complement component and mitosis). We implement a prototype for a Java-application which allows for browsing the Gene Ontology and supports the manual inspection of microarray-data.

We introduce a scoring-method which is based on a Kolmogorv-Smirnov test applied on the distribution of p-values. This score is similar to Tukey's higher criticism score dealing with the multiple-testing problem [14]. The analysis of microarray-data is a multiple-testing problem. So, we propose the application of the higher criticism-based KS-score to be an alternative method to the sum of logarithms-score proposed by Zien et al. and Pavlidis et al. [18][13]. We suggest two different null hypothesizes to assess the significance of the scores. In the first null hypothesis we assume, that there are differentially expressed genes falling randomly into different GO-nodes. We propose the accumulation test to check for this hypothesis. This test uses permutations of the rows of the data matrix randomly assigning the gene-wise expression values to GO-nodes. We compute the scores for these simulated GO-nodes to obtain a test statistic. This way to test the scores is identical to those proposed by Zien et al. and Pavlidis et al. [18][13]. In the second null hypothesis we separately assume for each GO-node that there are no differentially expressed genes in it. We propose the contamination test to check for this hypothesis. This test uses random permutations of the columns of the data matrix to obtain a test statistic. The permutations randomly assign tissue-samples to tissue-classes. We compute for each permutation gene-wise p-values and use them to calculate GO-scores. Dudoit et al. [7] propose permutations randomizing the class-assignment to perform a gene-wise significance analysis of microarray-data.

We modify this gene-wise screen by adding the GO-scoring step. Hence, we obtain a new test statistic different from that proposed by Zien et al. and Pavlidis et al.. We obtain it by a modification of the gene-wise significance analysis suggested by Dudoit et al..

We aim to find GO-nodes containing differentially expressed genes. For that reason, we propose two different statistical tests to identify the interesting GO-nodes. This raises an important biological question. What is the difference between the GO-nodes identified by the accumulation test and those identified by the contamination test. We show that this question is not only a formal, but can provide different insights into the biology. The accumulation test provides few GO-nodes with a significant score. This supports the manual inspection of single-genes expression data. Additionally, it provides genes that may be missed by gene-wise screens, because of multiple-testing. The GO-nodes "complement component" and "mitosis" are examples demonstrating, that the GO-approach can identify more differentially expressed genes than gene-wise screens. Their GO-score results only from the differential expression of genes, that cannot be identified by a gene-wise significance analysis. The contamination test identifies many interesting GO-nodes. A manual inspection of each of these nodes is hard. But, there is another benefit resulting from the contamination test. It provides a very interesting view on the expression data supported by the graph-structure of the Gene Ontology. The root-node is always positive, the smaller the GO-nodes are, the less significant ones can be identified. This property of the test allows for generating a hypothesis on the question, which branches of the Gene Ontology are mainly affected by differential gene-expression. The difference of the breast-cancer samples for example does not equally affect all the sub-branches of the Gene Ontology. We can show, that the node "apoptotic program", its children, grandchildren and so on are not differentially expressed. So, the contamination test allows insight, that supports for example the identification of whole processes that may be the target for new approaches in therapy. Both, the result of the accumulation and the contamination test is supported by the graphical user interface we implement for this purposes.

Biology is sometimes said to be a "knowledge based" rather than "axiom based" discipline [1]. The current work tries to combine the biological knowledge with the statistical analysis of microarray-data. It does not claim being the perfect solution to the problem. But, the analysis of large-scale gene-expression data requires an understanding of complex statistical methods on the one hand and a competent biological expertise on the other hand. Our Gene Ontology approach makes the microarray-data accessible in a form that highly supports a manual inspection by a molecular-biologist. The results from our method remain to be validated by additional molecular-biological experiments. But, the hypothesizes for these experiments can be generated much more effective.

References

[1] P.B. Baker, A. Brass, S. Bechhofer, C. Goble, N. Paton, and R. Stevens. TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. An Overview. In *ISMB98*, pages 25–34, Menlow Park, CA, June 28-July 1 1998. AAAI.

[2] The Gene Ontology Consortium. Gene ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[3] J.L. DeRisi, V.R. Iyer, and P.O Brown. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science*, 278:680–685, 1997.

[4] D.J.Duggan, M. Bittner, Y. Chen, P. Meltzer, and J.M. Trent. Expression profiling using cDNA microarrays. *Nature Genetics*, 21:10–14, 1999.

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

[6] D. Donoho and J. Jin. Higher Criticism for Detecting Sparse Heterogeneous Mixtures. Technical report, Department of Statistics, Stanford University, 2002.

[7] S. Dudoit, Y.H. Yang, T.P. Speed, and M.J. Callow. Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments. *Statisitca Sinica*, 12:111–139, 2002.

[8] M. Fellenberg and H.W. Mewes. Interpreting Clusters of Gene Expression Profiles in Terms of Metabolic Pathways. In *Proceedings of the German Conference on Bioinformatics*, 1999. Poster.

[9] J.M. Gross and D. Yee. How does the estrogen receptor work? *Breast Cancer Research*, 4:62–64, 2002.

[10] W. Huber, A.v. Heydebreck, and M. Vingron. Analysis of microarray gene expression data. To appear, 2003.

[11] G.G. Lennon and H. Lehrach. Hybridization analyses of arrayed cDNA libraries. *Trends in Genetics*, 10:314–317, 1991.

[12] R.J. Lipshutz, S.P. Fodor, T.R. Gingeras, and D.J. Lockhart. High density synthetic oligonucleotide arrays. *Nature Genetics*, 21:20–24, 1999.

[13] P. Pavlidis, D.L. Lewis, and W.S. Noble. Exploring gene expression data with class scores. *Proceedings of the Pacific Symposium on Biocomputing*, pages 474–485, 2002.

[14] J.W. Tukey. T13 N: The Higher Criticism. Course notes, Princeton University, 1976.

[15] M. West, C. Blanchette, H. Dressman, E. Huang, S. Ishida, R. Spang, H. Zuzan, J.A. Olson, J. R. Marks, and J.R. Nevins. Predicting the clinical status of human breast cancer by using gene expression profiles. *Proc. Natl. Acad. Sci. USA*, 98:11462–11467, 2001.

[16] P.H. Westfall and S.S. Young. Resampling-based multiple testing: examples and methods for p-value adjustment. *Whiley*, 1999.

[17] G. Wolf, R. Hildebrand, C. Schwar, R. Grobholz, M. Kaufmann, H.J. Stutte, K. Strebhardt, and U. Bleyl. Polo-like kinase: a novel marker of proliferation: correlation with estrogen-receptor expression in human breast cancer. *Pathol. Res. Prac.*, 196(11):753–759, 2000.

[18] A. Zien, R. Küffner, R. Zimmer, and T. Lengauer. Analysis of Gene Expression Data with Pathway Scores. In Russ Altman et al., editors, *ISMB00*, pages 407–417, La Jolla, CA, August 2000. AAAI.

[19] http://bioinfo.cnio.es/cgi-bin/tools/fatigo/fatigo.cgi. FatiGO: GO-based microarray analysis from Rámon Díaz-Uriarte.

[20] http://java.sun.com. Java web-page from Sun Microsystems.

[21] http://nicewww.cern.ch/∼hoschek/colt/index.htm. Colt-Application progamming interface from W. Hoscheck.

[22] http://www.affymetrix.com. Affymetrix web-page.

[23] http://www.geneontology.org. Gene Ontology web-page.

[24] http://www.mips.biochem.mpg.de/proj/yeast/. MIPS yeast functional catalog.

[25] http://www.molgen.mpg.de/∼bentink/results. Documentation of the Java-classes implemented in this work.

[26] http://www.ncbi.nlm.nih.gov/pubmed/. National Library of Medicine's medline and pre-medline database.