Large scale processing of microarray data
a diploma thesis

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Information and Communication Systems Research Group
Institute of Information Systems
Swiss Federal Institute of Technology
Zurich

Large Scale Processing of Microarray Data

Advisors:
Prof. Dr. Gustavo Alonso
Win Bausch

A diploma thesis by:
Reto Schaeppi
Lindenstrasse 33
8008 Zurich
Switzerland
sreto@student.ethz.ch
+41 (0)43 499 06 26

June 03, 2002

Acknowledgements

First, I want to thank Prof. Gustavo Alonso for providing me the opportunity to write my diploma thesis in his research group though my roots originates at the Department of Biology. The time in your institute has been full of intellectually stimulating experience and challenges.

The next big "Thank you" goes to my supervisor Win Bausch. He has invested a lot of time while teaching me the necessary computer skills to achieve this thesis as well as to learn me the language of a computer scientist. Many of the ideas discussed in our numberless conversations have resulted in this work.

I am also grateful to Cesare Pautasso, who has always kept patience while answering my questions.

Working in an inspiring atmosphere, I enjoyed my time at the Information and Communication Systems Research Group during the past year.

Zurich, June 2002

Contents

List of Tables

List of Figures

1. Introduction

In the last decade various projects, such as the Human Genome Project, have been launched to clone, map and sequence the genome of different organisms. The genome of several organisms, like Homo Sapiens or mouse, has already been determined and is publicly available. The ultimate goal of these efforts is the understanding of how an organisms genome acts as blueprint for defining its vital functions. Within the Human Genome Project, the researchers have catalogued over 4 million expressed sequence tags (ESTs) [1]. These ESTs correspond to a not yet determined number of unique human genes. Recent estimations range between 25'000 and 40'000 [2, 3]. The obtained sequence information, will eventually give rise to a better understanding of a genes function, the molecular events responsible for the ability of a cell to form an organism. It will be also crucial to discover the cause of diseases as well as to understand the phenomenon of aging. Finally, all this will lead to a increased efficiency in the development of new drug candidates. Just recently, functional genomics became a major tool to address the issues mentioned above, combining experimental and computational methods to analyze sequence information. Typically, these methods only yield reliable results, if performed on huge amounts of data.

For instance, the ability to identify genes by proceeding from a known protein to its chromosomal counterparts constitutes a highly complex research issue due to the complicated three-dimensional structures of proteins. The analysis of gene expression, at the mRNA level, may also result in functional information, because mRNA is coding for the proteins in an organism. For this purpose, two methods are currently popular: (1) Aligning unknown sequences against well known ones and (2) exploring the function of a gene by determining its pattern of expression. The first approach has led to the discovery of a wide variety of sequence motifs encoding structural domains, such as DNA-binding and nucleotide-binding domains, thus providing clues to gene function. The second approach considers changes in gene expression, evoked by a gene, to determine its function.

Traditional methods in this field work on a one gene in one experiment basis, implying that the throughput is very limited and getting the whole picture of a gene's function requires dozens of experiments. Thanks to the microarray technology, a paradigm shifts to one genome in one experiment is within reach. This technology allows researchers to monitor the whole genome on a single chip in a single experiment, so that they can take a snap-shot of the interactions among thousands of genes simultaneously - a dramatic increase in throughput.

This opportunity to carry out thousands of traditional experiments in one microarray experiment, represents a challenge for the biological researcher, mostly considering his computer skills: (1) One microarray initially creates approximately 50 megabytes of data. Since a typical experiment is carried out with ten to sixty microarrays in average, efficient data analysis requires more than the commonly used single desktop PC, (2) until now, biological research did not require sophisticated computer science skills in terms of programming. This results in the need of a closer interdisciplinary work between biological and computer scientists, which has not yet been fully established and therefore remains as a critical factor to allow efficient usage on DNA microarrays.

The microarray data analysis is based on various algorithms like *data normalization*, *background substraction* as well as *testing for differentially-expressed genes*. These algorithms are often combined into a software suite (Affymetrix Microarray Suite^{TM}[4], GeneData Expressionist^{TM}) [5]. These suites typically support explorative analysis of a microarray experiment and they only provide limited automatization capabilities of analysis process. In addition, data management tools are required for the increasing amounts of data (Stanford Microarray Database [6], ArrayExpress [7], GeneExpress [8]). Since different microarray technologies are currently available, standardization of the data is also a requirement of importance. There are various standardization projects being pursued by the Microarray Gene Expression Data Group (MGED) [9].

As a result of these efforts the focus in the microarray area is becoming to shift from data generation to data analysis [10]. Many tools are not suitable for large scale analysis. This thesis will give insight into some of the possibilities a process support system such as *BioOpera* provides to solve some issues involved. We focus on the data analysis part as a main goal. Once the analysis protocol has been developed, it can be fully automated and parallelized using *BioOpera*. The ability to automatically produce a clustered expression matrix represents a valuable and time saving addition compared to other tools.

2. Gene Expression Profiling

2.1. Expression Profiling Based on Microarrays

There are various methods available for detecting gene expression levels: northern blots [11], S1 nuclease protection [12], differential display [13], sequencing of cDNA libraries [14, 15] and serial analysis of gene expression [16] (SAGE). However, to take full advantage of the large and rapidly increasing body of genome sequence information, new technologies have been developed. Among the most powerful and versatile tools for functional genomics are high-density arrays of oligonucleotide (short DNA fragments) or complementary DNAs, which will be described in the following subsection. While the present study is focused on spotted DNA microarrays [17, 18], the techniques mentioned are generally applicable to expression data generated by oligonucleotide arrays [19], *Affymetrix GeneChips^{TM}* [20] or SAGE [16].

2.1.1. Microarray Technology

The underlying principle of DNA microarray technology is the spontaneous biochemical base-pairing process of complementary base pairs, called hybridization. An array is an orderly arrangement of known cDNA sequences or oligonucleotides. It provides a medium for matching known DNA (probe sequences) and unknown, fluorescently[1] labeled DNA or RNA samples (target sequences). The labeled target sequences allow a quantitative measurement of their abundance in the sample, i.e. tissue, cells, blood etc., being investigated. An array experiment can be created by hand or make use of robotics to fully automate the process of hybridization or deposition of the target sequences. In general, arrays are described as macroarrays or microarrays, the difference being the size of the sample spots which contain the probe sequences. On the chip, target sequences are organized in so-called spots. Each of these representing a gene. Macroarrays contain sample spot sizes of about 300 microns or larger. The sample spot sizes in microarrays are typically less than 200 microns in diameter and these arrays

[1]The labeling strategy depends on the used microarray technology

usually contains millions of spots. Microarrays require specialized high-speed robotics for manufacturing and imaging equipment (scanner) for measuring the raw intensity data for each spot. Generally, the chips are manufactured on glass but sometimes on nylon substrates, for which probes with known identity are used to determine complementary binding, thus allowing massively parallel gene expression and gene discovery studies. Details on spotted DNA microarrays and oligonucleotide arrays can be found in [21] and [22], respectively. Mainly, there are two variants of the DNA microarray technology, in terms of the properties of arrayed DNA probe sequence with known identity:

1. Spotted cDNA microarrays

    Probe cDNA (500∼5,000 bases long) is immobilized to a solid surface such as a microscope glass using robot spotting and exposed to a set of labeled targets. Usually Cye3-dUTP and Cye5-dUTP are used to label the target samples. Expression Arrays containing up to 8000 genes are printed onto a 2 x 4 cm glass slide with a probe diameter of 75 - 100μm and a 150μm distance between probes. Today, technological advances in printing and scanning technology allows to the manufacturing of microarrays that contain the entire human genome, i.e. over 40'000 complete cDNA sequences. The probe sequences to be printed on the array are usually selected from databases such as UniGene [23] and GeneBank [24] and then physically produced by extracting the plasmid DNA from the cDNA clones found in a cDNA library and then amplifying them through Polymerase Chain Reaction (PCR). Figure 2.1a shows a spotted cDNA array after hybridization of labeled samples and fluorescence detection. The image has been colored to indicate the relative number of yeast transcripts present in two different growth condition (red: high in condition 1, low in condition 2; green: high in condition 2, low in condition 1; yellow: high in both conditions; black: low in both conditions). Approximately one nanogram of material is deposited on the array and consists of double-stranded DNA probes used for each gene or EST. After hybridization and scanning, the quantitative fluorescence image, along with the known identity of the probes, is used to asses the 'presence' or 'absence' of a particular molecule (such as a transcript), and its relative abundance in one or more samples. Because the cDNA at each physical location (or address) is well described, and the recognition rules that govern hybridization are well characterized, the signal intensity at each position of the microarray gives a quantitative measurement of one single target sequence with known identity. The signal intensities obtained by spotted cDNA arrays give the relative concentration (ratio) of a given transcript in two different samples (derived from competitive, two-color hybridizations). Messenger RNAs present at a few copies (relative abundance of ∼1:100'000 or less) to thousands of copies per mammalian cell, can be detected, and changes as subtle as a factor of 1.3 to 2 can be reliably detected, if replicate experiments are performed. Figure 2.1b describes the two-color hybridization strategy often used with cDNA microarrays. cDNA from two different conditions or one single condition and its control are labeled with two different fluorescent dyes, as mentioned above allowing a direct and quantitative comparison after scanning simultaneously at two different wavelengths.

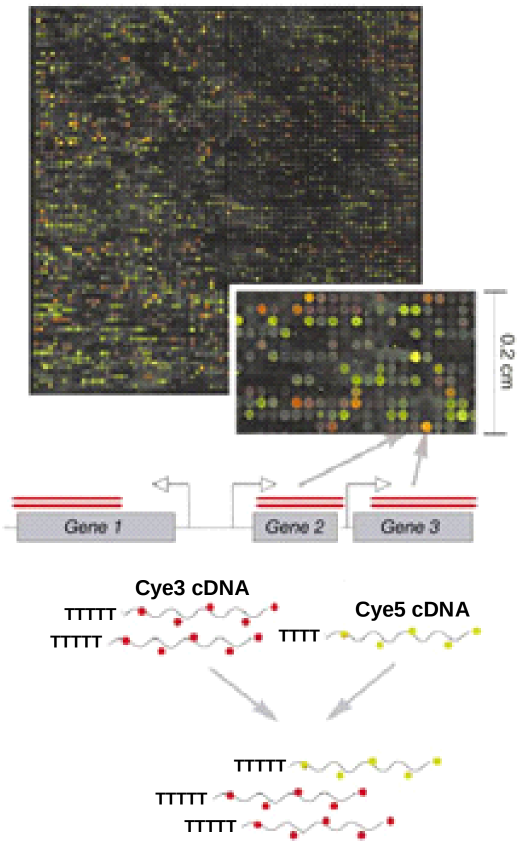


Figure 2.1.: **a**: A spotted cDNA array after hybridization of labeled samples and fluorescence detection. **b**: Two-color hybridization strategy often used with cDNA microarrys. (Modified from Lockhart, D.J. and Winzeler, E.A., Nature 405, 2000)

2. Oligonucleotide microarrays

   Oligonucleotides (20∼80-mer oligos) are exposed to labeled sample cDNA (targets), hybridized, and the identity/abundance of complementary sequences are determined. Often, the target sample DNA is labeled by using a biotin derivative. The probes are synthesized either in situ (on-chip) or by conventional synthesis followed by on-chip immobilization. *Affymetrix, Inc.*, the most popular vendor of this kind of array, uses photolithography and solid-phase chemistry to produce their GeneChips^{TM}. Their Human Genome U133 Set [25] contains for example more than 45'000 probe sets with a feature size of $18\mu m$ corresponding to 33'000 human genes. Many companies are manufacturing oligonucleotide based chips using alternative in-situ synthesis, such as piezoelectric (ink-jet). Figure 2.2a shows an oligonucleotide microarray. Similar to the spotted DNA microarray, the image has been colored to indicate the relative number of yeast transcripts present in two different growth conditions (red: high in condition 1, low in condition 2; green: high in condition 2, low in condition 1; yellow: high in both conditions; black: low in both conditions). Approximately 10^7 copies of each selected oligonucleotide are synthesized base by base in hundreds of thousands of different $24\mu m$ x $24\mu m$ areas on a $1.28cm$ x $1.28cm$ glass surface. After hybridization and scanning, the quantitative fluorescence image along with the known identity of the probes is used to asses the 'presence' or 'absence' of a particular molecule (such as a transcript), and its relative abundance in one or more samples. Because the oligonucleotides at each physical location (or address) is well described, and the recognition rules that govern hybridization are well characterized, the signal intensity at each position of the microarray gives a quantitative measurement of one single target sequence with known identity. Although oligonucleotide probes vary systematically in their hybridization efficiency, quantitative estimates of the number of transcripts per cell can be obtained directly by averaging the signal from multiple probes. Therefore, a gene is visually represented by multiple oligonucleotides. Figure 2.2b shows different methods for preparing the labeled material for measurements required for gene expression profiling.

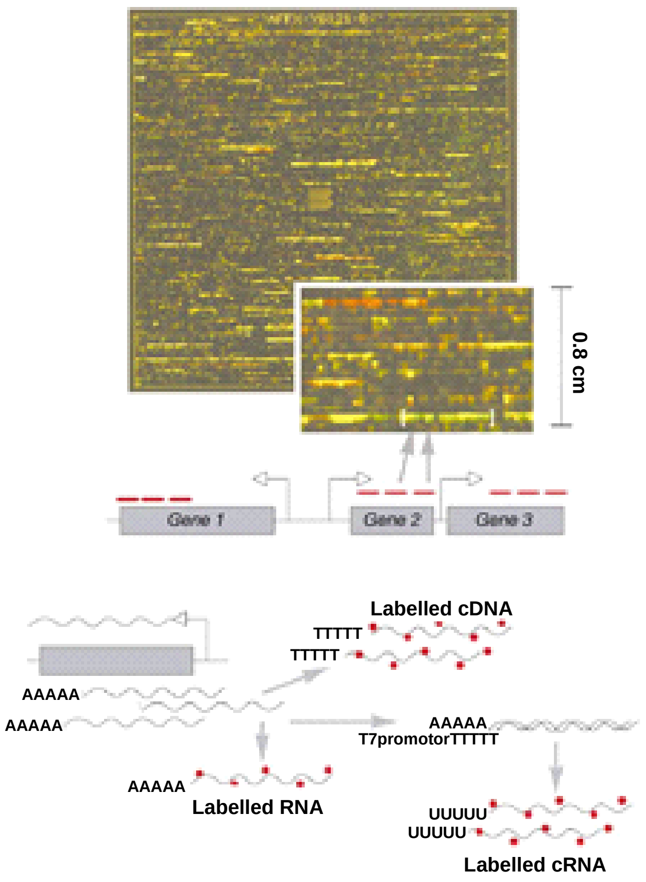


Figure 2.2.: **a**: An oligonucleotide array. **b**: Different methods for preparing labeled materials. Modified from Lockhart, D.J. and Winzeler, E.A., Nature 405, 2000)

2.1.2. From Raw Data to a Clustered Expression Matrix

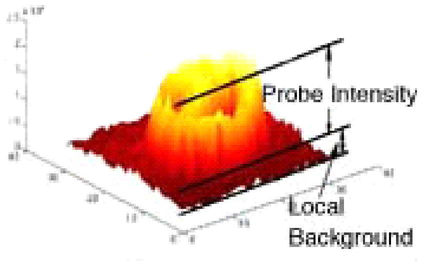


Figure 2.3.: Three-dimensional representation of the intensity of one single spot

After processing the required microarrays, the resulting images have to be analyzed to identify the arrayed spots and to measure the fluorescence intensities for each element (Figure 2.3). During hybridization some of the probe mRNA will attach to the array, even when there is no cDNA available. This is known as "background intensity". Within the foreground region the mRNA can hybridize to the target cDNA or to the glass itself. These two effects are assumed to be additive (Figure 2.4). Therefore, the extracted raw spot intensities have to be background corrected [26].

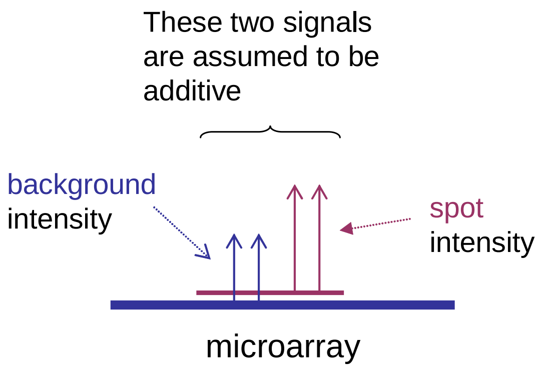


Figure 2.4.: The idea of background substraction

Differences in labelling and detection efficiencies for the fluorescent labels, as well as differences in the quantity of initial mRNA from the two samples examined in the assays can cause a shift in the average ratio of Cye5 to Cye3. Data normalization adjusts for these differences. The intensities must be rescaled before an experiment can be properly analyzed (Figure 2.5). Currently, there are different normalization algorithms used for preprocessing microarray data, which are described in [27, 28, 29, 30, 31].

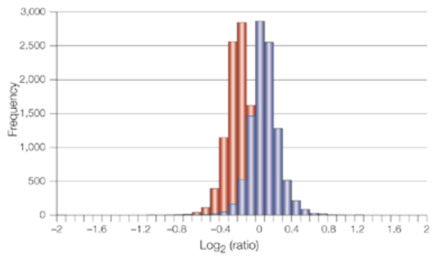


Figure 2.5.: The measured Cye5 intensity is generally less than the measured Cye3 intensity and therefore, in the histogram before data normalization (red), the average intensity is biased to the left of zero. Data normalization adjusts this shift in the average ratio and is shown in the blue histogram. (Figure Quackenbush, J., Nature Reviews Genetics 2, 2001)

After normalization, the data for each gene is reported as an expression ratio or as the logarithm of the expression ratio. The expression ratio is the background corrected, normalized value of the expression level for a particular gene in the query sample divided by the corresponding background corrected, normalized value of the control. The advantage of taking the logarithm of the expression ratio is that a gene upregulated by a factor of 2 has a $log_2(ratio)$ of 1, whereas a gene downregulated by the same factor has a $log_2(ratio)$ of -1, and a gene expressed at a constant level (with a ratio of 1) has a $log_2(ratio)$ of 0.

The next step in the microarray data analysis is identifying differentially-expressed genes within each microarray slide. The test for differentially-expressed genes directly compares a series of repeated measurements of the two dye intensities for each gene. This test uses a statistical model to describe multiplicative and additive errors influencing an array experiment, where model parameters are estimated from observed intensities for genes using the maximum-likelihood method. A generalized likelihood ratio test is performed for each gene to determine whether, under the model, these intensities are significantly different [32]. Those genes which are not differentially-expressed are excluded from the cluster analysis.

A powerful application of transcriptional profiling is the study of patterns of gene expression in order to survey cellular responses and conditions. To this end, the cells of interest are sampled at regular time intervals or after different treatments, each sample being analyzed using a separate microarray. The simplest way to identify genes of potential interest is to search for those that are consistently either up- or downregulated. This is a strong indication that these genes are part of the same pathway. To that end, a simple statistical analysis of gene-expression levels will suffice. Identifying patterns of gene expression and grouping genes into expression classes provides greater insight into their biological function and relevance. There exists a large group of statistical methods, generally referred to as *cluster analysis*. Various clustering techniques have been applied to the identification of patterns in gene expression data:

1. Unsupervised methods

    - Hierarchical clustering [33]

        Hierarchical clustering has the advantage that it is simple and the result can be easily visualized. It has become one of the most widely used techniques for the analysis of gene expression data. Hierarchical clustering is an agglomerative approach in which single expression profiles are joined to form groups, which are further joined until the process has been carried to completion, forming a single hierarchical tree.

    - k-means clustering [34]

        If there is advanced knowledge about the number of clusters that should be represented in the data, k-means clustering is an alternative to hierarchical methods. In k-means clustering, objects are partitioned into a fixed number (k) of clusters, such that the clusters are internally similar but externally dissimilar; no dendrograms are produced but one could use hierarchical techniques on each of the data partitions after they are constructed.

    - Self-organizing maps (SOM) [35]

        A SOM is a neural network based divisive clustering approach. A SOM assigns genes to a series of partitions on the basis of the similarity of their expression vectors to reference vectors that are defined for each partition. It is the process of defining these reference vectors that distinguishes SOMs from k-means clustering.

2. Supervised methods

    - Support vector machine (SVM) [36]

        SVMs use a training set in which genes known to be related by, for example, functions are provided as positive examples and genes known not to be members of that class are negative examples. These are combined into a set of training examples that is used by the SVM to learn to distinguish between members and non-members of the class on the basis of expression data.

- Artificial neural network (ANN) [37, 38]

  An ANN is an artificial intelligence tool that identifies arbitrary non-linear multiparametric discriminant functions directly from the experimental data. This machine learning technique can be roughly described as a universal algebraic function that will distinguish signal from noise directly from experimental data.

In supervised learning, pairs of inputs and outputs are given, and transfer functions are modified (by updating weights) to minimize the classification error. In unsupervised learning, in contrast, inputs and optimization criteria are given, and weights are modified, on the basis of outputs weights.

2.2. Microarray Data Analysis Pipeline

The *Institute for Systems Biology*, a non-profit research institute located in Seattle (WA, USA), is using different tools for their DNA microarray data processing. These tools are arranged in a pipeline[2]:

dapple Performs the image processing. Locates and quantifies DNA spots in a microarray image and outputs the raw intensity data.

preprocess Performs background substraction, normalization, and lookup on raw intensity data. Also provides rudimentary gene expression ratio for each gene.

mergeReps Combines the data from multiple preprocessed files and computes the average expression ratio of each gene over the replicate measurements.

VERA *Variability and ERror Assessment.* Estimates the error model parameters from replicated, preprocessed experiments.

SAM *Significance of Array Measurement.* Uses the error model to improve the accuracy of the expression ratio and to assign a value to each gene, indicating the likelihood that the gene is differentially-expressed between two conditions.

mergeConds Creates a gene expression matrix from multiple conditions.

[2]For a detailed description of this *pipeline*, please refer to [39]

For our purpose, we have adapted this analysis pipeline. Before the proper data analysis begins, the material, hybridized later on the DNA chips, must be prepared. For this purpose, the researcher treats cell populations under different conditions and retains untreated cells as controls. As the treatments are finished, the mRNA has to be extracted from the cell populations and transformed to cDNA by reverse transcription. For each condition, the researcher extracts the mRNA to be analyzed. Each treatment condition and its control are different fluorescently labeled and mixed together. Figure 2.6 shows the hybridization step, where the mixture of mRNA is applied onto the DNA chips. To improve the significance of the following data analysis, the microarray chip for one condition is often replicated (usually up to 4 times). After washing, the arrays are scanned at two different wavelengths to detect the relative transcript abundance for each condition. Figure 2.7 shows the resulting monochrome images (for Cye3 and Cye5) produced by the scanner.

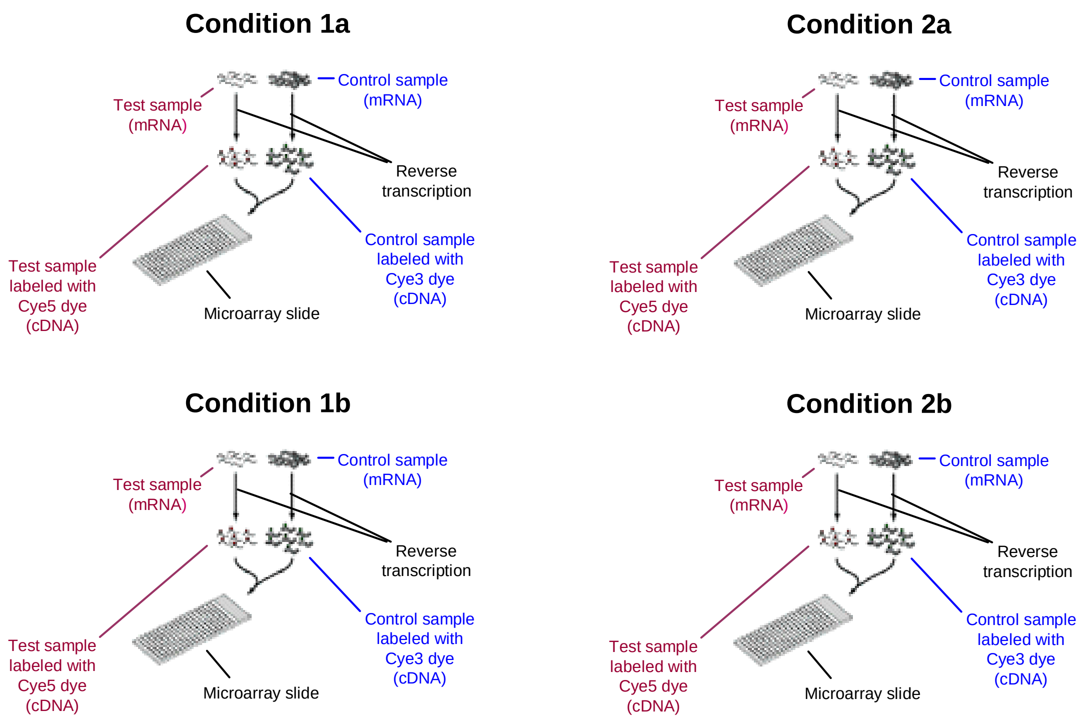


Figure 2.6.: Hybridization of the cDNA to the microarray slide. Each of the two conditions is replicated twice (C1a, C1b, C2a, C2b)

After those preparations in the wet lab, the proper data analysis can be started. First, the generated images have to be computationally post processed. The monochrome images are pseudo-colored based on their labels and merged into a single

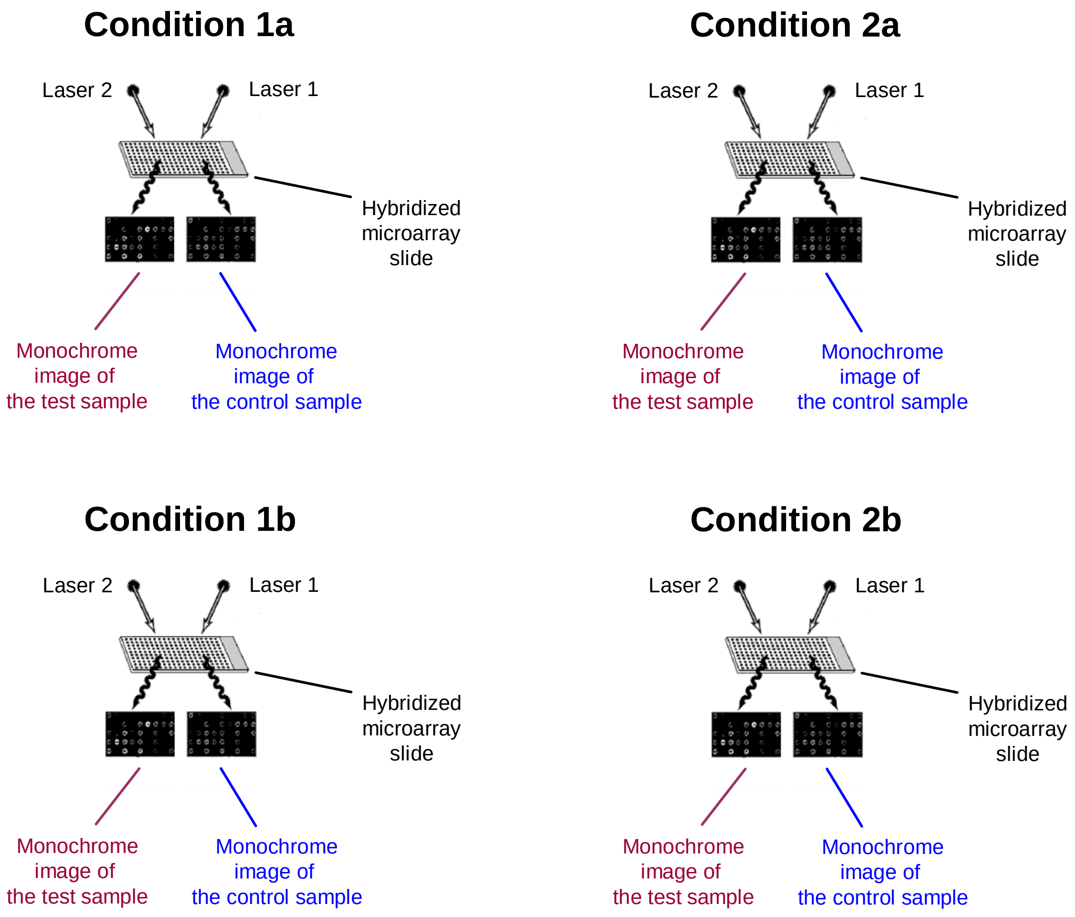
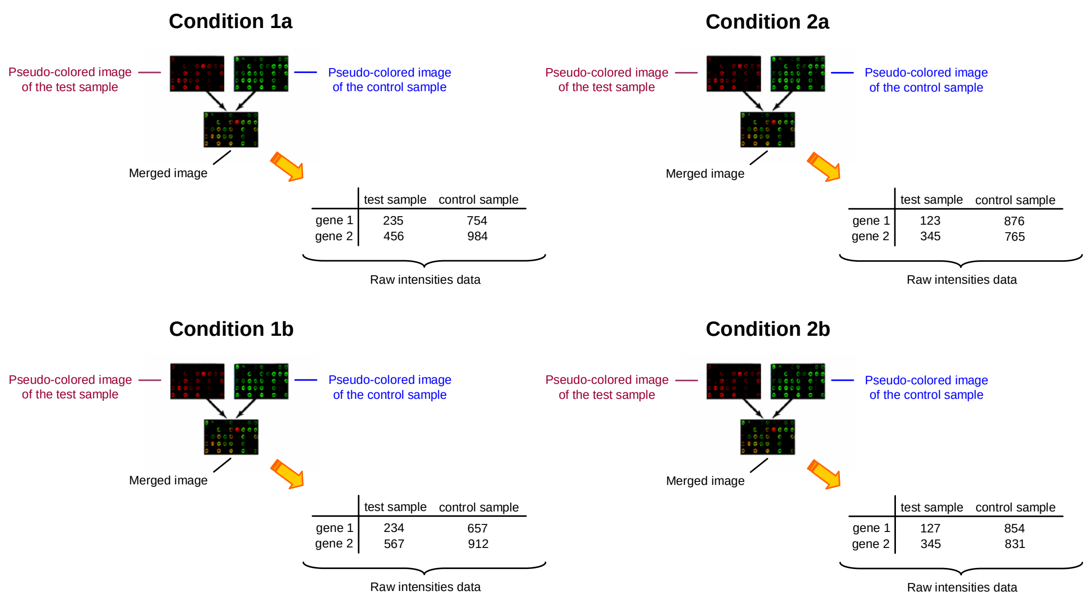

Condition 1a
Laser 2
Laser 1
Hybridized microarray slide
Monochrome image of the test sample
Monochrome image of the control sample
Condition 2a
Laser 2
Laser 1
Hybridized microarray slide
Monochrome image of the test sample
Monochrome image of the control sample
Condition 1b
Laser 2
Laser 1
Hybridized microarray slide
Monochrome image of the test sample
Monochrome image of the control sample
Condition 2b
Laser 2
Laser 1
Hybridized microarray slide
Monochrome image of the test sample
Monochrome image of the control sample

Figure 2.7.: Scanning the hybridized microarray slide


Condition 1a
Pseudo-colored image of the test sample
Pseudo-colored image of the control sample
Merged image
Raw intensities data
Condition 2a
Pseudo-colored image of the test sample
Pseudo-colored image of the control sample
Merged image
Raw intensities data
Condition 1b
Pseudo-colored image of the test sample
Pseudo-colored image of the control sample
Merged image
Raw intensities data
Condition 2b
Pseudo-colored image of the test sample
Pseudo-colored image of the control sample
Merged image
Raw intensities data

Condition 1a

	test sample	control sample
gene 1	235	754
gene 2	456	984

Condition 2a

	test sample	control sample
gene 1	123	876
gene 2	345	765

Condition 1b

	test sample	control sample
gene 1	234	657
gene 2	567	912

Condition 2b

	test sample	control sample
gene 1	127	854
gene 2	345	831

Figure 2.8.: Image processing after scanning the microarray slide

image for each replicated condition, and the raw intensities are extracted, as shown in Figure 2.8. The result of the image processing step is a simple spot matrix for each replicated condition listing the measured relative intensities for each arrayed spot and channel. The columns represents the two channels (Cye3 and Cye5), the rows the different arrayed spots (genes). All remaining steps consist in transformations of this data, which have been automated by the implemented process. Generally, the aim of all these transformations is to ensure the quality and standardization of the data set. For each replicated condition, the raw data needs to be normalized. The background has to be subtracted from each arrayed spot. In addition to its coordinates, each spot needs to be annotated with gene specific information. The replicated conditions are then combined together by taking the average intensities. The result is a simple merged spot matrix representing one experimental condition (Figure 2.9). Based on this matrix, the next step estimates an error model for each condition and uses it to identify differentially-expressed genes between the control and treated sample. Finally, the multiple conditions are merged into a global gene expression matrix in which the columns represent the different conditions, whereas the rows represent the different genes. The last step in the data analysis is the cluster analysis. Finally, the clustered data can be visualized with an appropriate software tool (Figure 2.10).

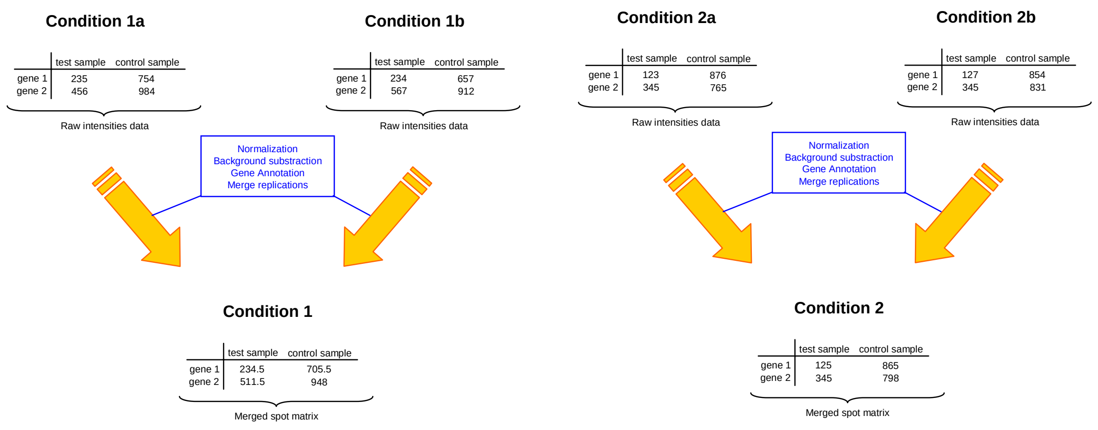


Figure 2.9.: Preprocessing the raw data

2.3. Experimental Data

The experimental dataset, we used for our tests, has been downloaded form the *Stanford Microarray Database (SMD)* [6]. *SMD* stores raw and normalized data from microarray experiments, and provides web interfaces to retrieve, analyze and visualize the data. The two immediate goals for *SMD* are to serve as a storage site for microarray data from ongoing research at *Stanford University*, and to facilitate the dissemination of that data once published, or released by the researcher [40].

The data originates from an experiment, where distinct types of diffuse large B-cell lymphoma (subtype of non-Hodgkin's lymphoma, cancer) have been identified by gene expression profiling [41]. The dataset comprises 133 microarrays whereof 30 conditions were performed in duplicate, one condition in triplicate and the remainder was performed once. For their data analysis, they have excluded three single conditions and two duplicated ones, due to quality reasons. Therefore their final dataset comprises 126 microarrays for the 96 conditions analyzed.

For our purpose, we used less than 126 microarrays, because our analysis tests for differentially-expressed genes. This test presumes that the experiments are replicated (at least 4 times). Our data does not fully comply to this requirement, therefore we only used 96 microarrays representing 66 different conditions in our analysis.

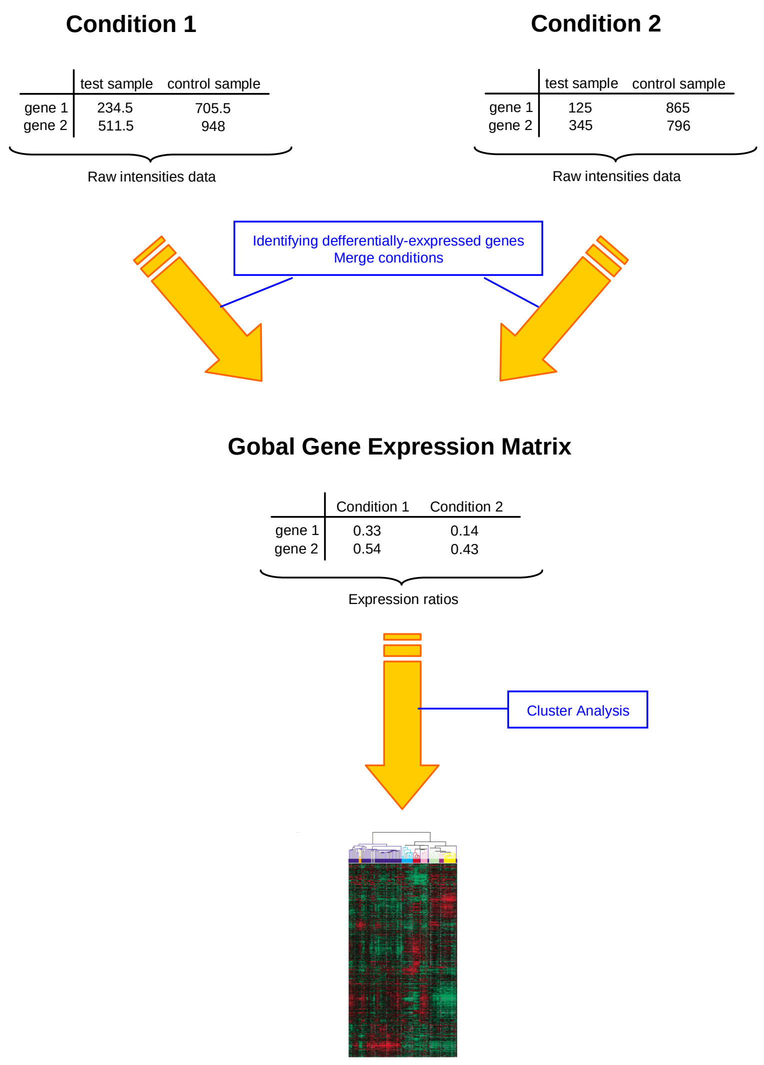


Figure 2.10.: Cluster analysis of the merged experimental conditions

3. BioOpera

3.1. The Concept of Process

BioOpera uses the concept of process to describe computations. A process is a set of computational tasks that are to be executed as well as a description of the dependencies between these tasks. Processes can be specified using both a graphical notation or a textual language. The programming language used by *BioOpera* is called Opera Canonical Representation (OCR) and it is a textual, rule-based language for high-level description of computations involving several interdependent steps. The result of the specification is a process template. Process templates can be executed by *BioOpera*. When *BioOpera* runs a specific process template, it generates a process instance from this template. It uses the information contained in the template to decide when to execute the tasks. This procedure is called navigation. This way, the instance always reflects the current state of the process: which tasks are running, which ones already terminated, which ones still need to be executed. Whenever a task is eligible to be executed, *BioOpera* determines the external software to call for this task and tries to find a computing resource to execute it on. To do this, it must be aware of the hardware and software at its disposal for executing a process. The user has to provide information about the available programs and computing resources by registering program descriptors and and resource descriptors with the system. For more details about *BioOpera*, refer to [42].

3.1.1. Resource Descriptors

We have to define the resource descriptors in order to specify the properties of the computing resources we want to use for the execution of the process. Resource descriptors consists of:

Host

- Registering a host simply requires the corresponding host name or IP address. *BioOpera* automatically identifies the hardware and software settings of the node (number of CPUs, available memory and swap space, operating system).

Group

- The system supports the definition of sub-clusters and arbitrary groups of nodes, including nested and overlapping ones. They are used for scheduling, access control, resource distribution (e.g. user A runs on one sub-cluster, user B on a different sub-cluster) and for describing the machines suitable for running a given activity.

3.1.2. Program Descriptors

Each activity in a process corresponds to an external application that needs to be invoked. Before an activity can be mapped to an application, the application must be registered with *BioOpera*:

Interface

- Specifies the interface (input and output parameters) of the program as well as how to run it.

Command

- Specifies the mechanism to be used by *BioOpera* to start the program on a host.

Resource

- Specifies the group (range of nodes) where the program can be invoked.

3.1.3. Process Components

A process is a set of computational tasks that are to be executed. A task can be a call to software residing outside of *BioOpera* (e.g. shell scripts, binary executables) or a call to another process. Tasks can be:

Activity

- An Activity involves the execution of an external program on one of the cluster nodes.

Subprocess

- A subprocess calls another *BioOpera* process. Subprocesses are used to support modularity and code encapsulation.

Tasks and processes have input and output parameters. Each one of these parameters can be assigned a default value that is used if the user or the tasks do not provide an alternative value.

3.1.4. Inter-Task Dependencies

A process also describes the dependencies between the above mentioned tasks. There are two types of dependencies between computational tasks:

Control Flow

- The partial order between the different steps of the computation is specified as a control flow. The current version of *BioOpera* supports only directed acyclic control flows. Each task has a boolean predicate (a condition) that can be arbitrarily defined over any data parameters within the scope of the process. This predicate is used to implement complex control flow dependencies such as conditional branches. A task will not start until all control flow dependencies are satisfied, and the starting condition evaluates to true.

Data Flow

- The mapping between the output and input parameters of different tasks is specified as a data flow. Data flow dependencies are created by connecting the output parameters of the task producing data to the input parameters of the task consuming data. The programmer has the possibility for specifying parallel tasks through specialized data flow connections. These connectors unfolds dynamically the tasks into multiple concurrent tasks as specified by the input parameters. Equivalent to saying: if an array parameter p is bound to a scalar input parameter q of the task t, at runtime *BioOpera* will start in parallel on instance t_i of the task t for each element i of the array p. Such tasks are called explosive tasks.

4. Process Design

To process our experimental data, we use two different processes:

1. Data Preparation Process (DPP)

2. Expression Profiling Process (EPP)

First, the original data was not in the file format required by the programs used by EPP. This is why DPP transforms the data into the required format. EPP performs the data analysis. The data preparation and the analysis have been implemented separately for the sake of modularity and flexibility. Before implementing the processes, the cluster nodes as well as the programs involved need to be set up. The following sections describe the hardware and software setup as well as the design of DPP and EPP.

4.1. Setting Up the Environment

As we have seen, the computing environment needs to be set up. The cluster needs to be defined and the programs have to be registered within *BioOpera*. The table 4.1 summarizes the main hardware and software characteristics of the used cluster of PCs.

Nodes		CPU(Mhz)	RAM(MB)	OS
L.	60	P-III (1000)	1024	LINUX v2.4.17

Table 4.1.: Cluster setup

Both the programs and the data set have been put into a common directory subtree, made available to each cluster node using NFS. This configuration fits our purpose as long as the appropriate data is moved to the cluster node's local disks. This needs to be done because NFS does not scale for a cluster of the above size.

The programs used are either (1) overtaken unmodified from the Institute for Systems Biology (Section 2.2) or (2) they have been wrapped. In both cases, minor adaptations of the scripts were necessary in order to use them through *BioOpera*.

4.2. Data Preparation Process (DPP)

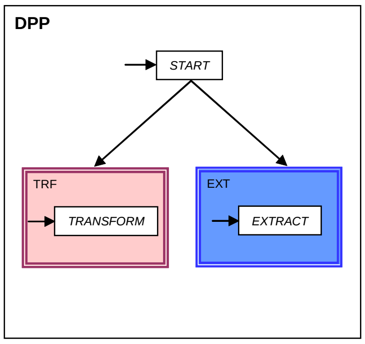


Figure 4.1.: Control flow of the data preparation process (DPP)

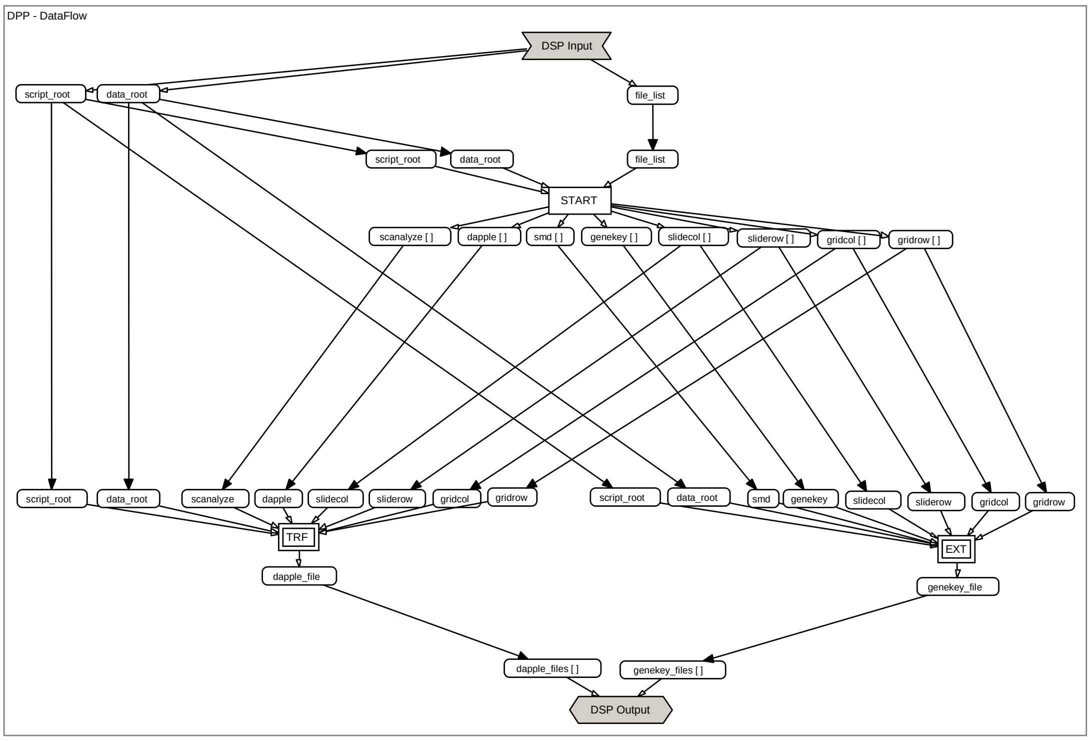


Figure 4.2.: Data flow of the data preparation process (DPP)

The *Institute for Systems Biology* uses *dapple* [43] for the image processing. The downloaded experimental data set however was evaluated with a different software tool, called *ScanAlyze* [44]. The only difference is the file format. The ScanAlyze data files need to be transformed into to correct format for *dapple* [43]. Furthermore, the *SMD* (see section 2.3) does not provide a genekey file for the microarray chips used in the experiment. Therefore, such a genekey[1] file needs to be generated by extracting the gene-specific information from the raw data, provided by the *SMD*. The data preparation process performs these tasks on the raw data. Figure 4.1 shows its control flow, Figure 4.2 the corresponding data flow. The process consists of a partitioning task (*START*) and two explosive tasks (*TRF*, *EXT*), the latter being started on termination of the former. The data flow consists of three classes of input/output parameters:

1. Deployment parameters
2. Input/output file locations
3. Slide geometry describing the spot arrangement in the underlying microarray slide.

The data processed by DPP and EPP is stored in flat files. The process parameters themselves do not hold the content of these files but merely their names. For the sake of readability, we use the term, "file" and "file name" interchangeably when talking about process parameter values.

The first task, *START*, partitions each row of the input file list and puts the content of each column into a separate array. The following list summarizes its input/output parameters as well as the involved programs:

INPUT

Deployment parameters:

data_root

Absolute path to the data.

script_root

Absolute path to the script invoked by *START*.

Input file locations:

file_list

List containing the files to be transformed, the files to be extracted as well as the corresponding spot arrangement (Table 4.2). A single row represents one microarray slide. The first two columns list the

[1]A genekey file contains the information about the genes (spots) arranged on a microarray slide

input and the output file for the transformational step, the next two columns list the input and output file for the generation of the gene information, and the remaining columns indicate the underlying microarray geometry described with four separate parameters. Figure 4.3 shows a 4x4x24x24 geometry.

OUTPUT

Output file locations:

scanalyze[]

Array holding a list of file names, each of them designating a ScanAlyze file to be transformed.

dapple[]

Array holding a list of file names, each of them designating a transformed dapple file

smd[]

Array holding a list of file names, each of them designating a SMD file from which the gene information will be extracted.

genekey[]

Array holding a list of file names, each of them designating the file containing gene information extracted from the corresponding SMD file.

Slide geometry:

slidecol[]

Array holding a list of numbers, each of them describing the maximum number of slide columns.

sliderow[]

Array holding a list of numbers, each of them describing the maximum number of slide rows.

gridcol[]

Array holding a list of numbers, each of them describing the maximum number of grid columns.

sliderow[]

Array holding a list of numbers, each of them describing the maximum number of grid rows.

INVOKED PROGRAMS

START.pl

see Appendix A.5.1

Access method

```
perl %script_root%START.pl %file_list% -dataroot %data_root%
```

START enables to invoke the two explosive tasks in the next step, which will be executed in parallel. At the end of the process, the produced output files are collected into a separate array for each subprocess (*dapple_files[]* and *genekey_files[]*).

ScanAlyze	*dapple*	*SMD*	*genekey*	*slidecol*	*sliderow*	*gridcol*	*gridrow*
lc4b039rex2.DAT	lc4b039.dat	5789.xls	genekey_lc4b039.dat	4	4	24	24
lc8n077rex2.DAT	lc8n077.dat	5821.xls	genekey_lc8n077.dat	4	8	24	24

Table 4.2.: Example Input File List for the data preparation process

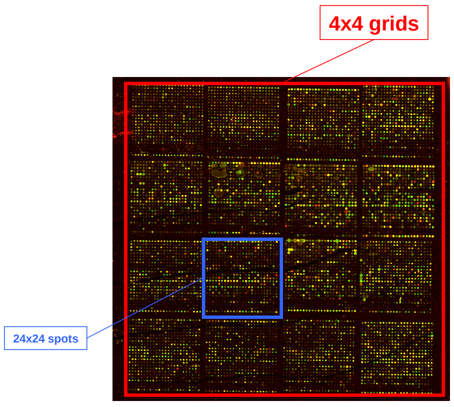


Figure 4.3.: Spot arrangement on a microarray slide. The outer square contains 4x4 grids, the inner one 24x24 spots.

4.2.1. Transforming the Raw Data File Format (TRANSFORM)

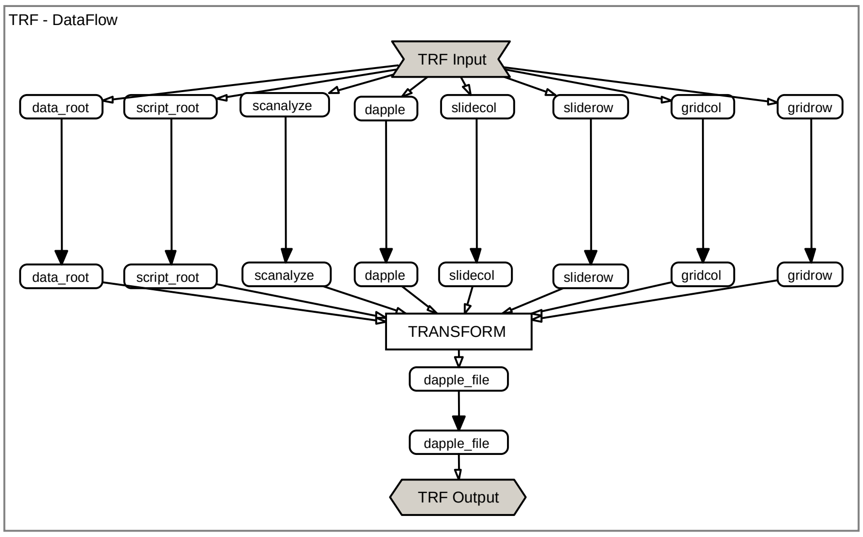


Figure 4.4.: Data flow of the TRF subprocess

The ScanAlyze file consists in a matrix describing the raw spot intensities of one microarray slide. *TRF* transforms this matrix into another by reordering the matrix elements. The output is written to a file, of which location and name is passed as input parameters. The output adheres to the dapple file format[2]. The explosive task involves the following parameters (Figure 4.4) as well as the following program:

INPUT

Deployment parameters:

data_root

Absolute path to the data.

script_root

Absolute path to the script invoked by *TRF*.

Input file locations:

scanalyze

ScanAlyze file containing the raw spot intensities for one microarray slide.

[2]Refer to Appendix A.1 and A.2 for a detailed description of the various file formats

dapple

Output file name of the transformed data.

Slide geometry:

slidecol

Number of slide columns of the underlying microarray slide.

sliderow

Number of slide rows of the underlying microarray slide.

gridcol

Number of grid columns of the underlying microarray slide.

gridrow

Number of grid rows of the underlying microarray slide.

OUTPUT

Output file locations:

dapple_file

Transformed output file.

INVOKED PROGRAMS

TRANSFORM.pl

see Appendix A.5.2

Access method

```
perl %script_root%TRANSFORM.pl %scanalyze% %dapple%
-arrayconfig %slidecol% %sliderow% %gridcol% %gridrow%
-dataroot %data_root%
```

4.2.2. Extracting Gene Information (EXTRACT)

Microarray image processing tools produce, for each spot, the position of the spot, the raw intensities for the two channels, and some statistical values concerning the latter. Biologically, this data is not extensively meaningful because there is no information about the name and the description of the gene represented by a particular spot. Therefore, each spot has to be annotated after the image processing. Usually, annotation information is provided by the chip manufacturer in a so-called genekey file. Our

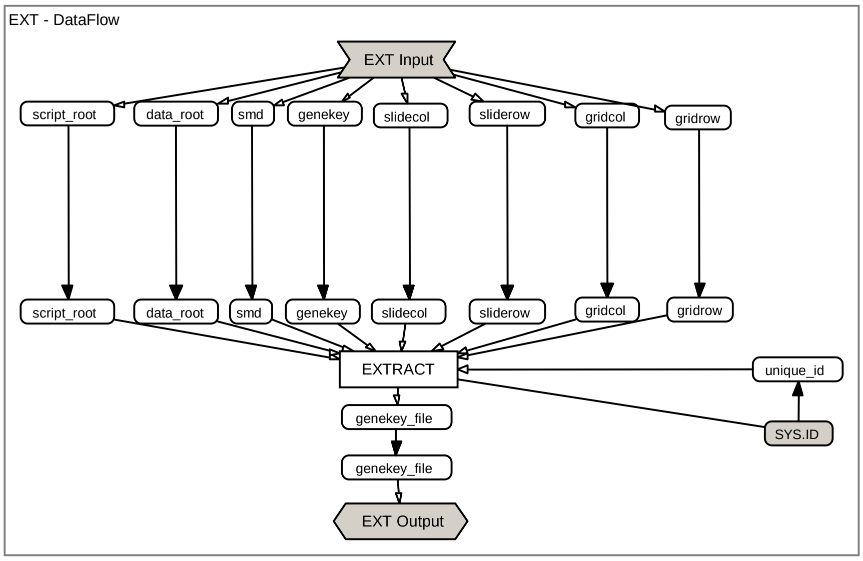


Figure 4.5.: Data flow of the EXT subprocess

experimental data, however, does not come with such a file. Therefore, the *EXT* subprocess generates all required genekey files[3] by extracting the appropriate information from the SMD file and coalescing it into a single file. The output is written to a file, of which location and name is passed as input parameters. The following list shows its involved parameters (Figure 4.5) and program:

INPUT

Deployment parameters:

data_root

Absolute path to the data.

script_root

Absolute path to the script invoked by *EXT*.

Input file locations:

smd

SMD file containing gene information.

[3]Refer to Appendix A.4 for a detailed description of the extracted data fields

genekey

Output file name of the extracted gene information.

Slide geometry:

slidecol

Number of slide columns of the underlying microarray slide.

sliderow

Number of slide rows of the underlying microarray slide.

gridcol

Number of grid columns of the underlying microarray slide.

gridrow

Number of grid rows of the underlying microarray slide.

Various input parameters:

unique_id

The involved scripts writes temporary information to a file. When several *EXTRACT* tasks are executed on the same machine in parallel, each of them needs to write to its own file. Therefore, the *EXTRACT* task needs a unique identification to name its temporary file.

OUTPUT

Output file locations:

genekey_file

Gene information file.

INVOKED PROGRAMS

EXTRACT.pl

see Appendix A.5.3

Access method

```
perl %script_root%EXTRACT.pl %smd% %genekey% -arrayconfig
%slidecol% %sliderow% %gridcol% %gridrow% -dataroot
%data_root% -uid %unique_id%
```

4.3. Expression Profiling Process (EPP)

In this section, the design of the expression profiling process (EPP), its control and data flow as well as each task's functional part within the analysis will be described.

4.3.1. EPP Design Issues

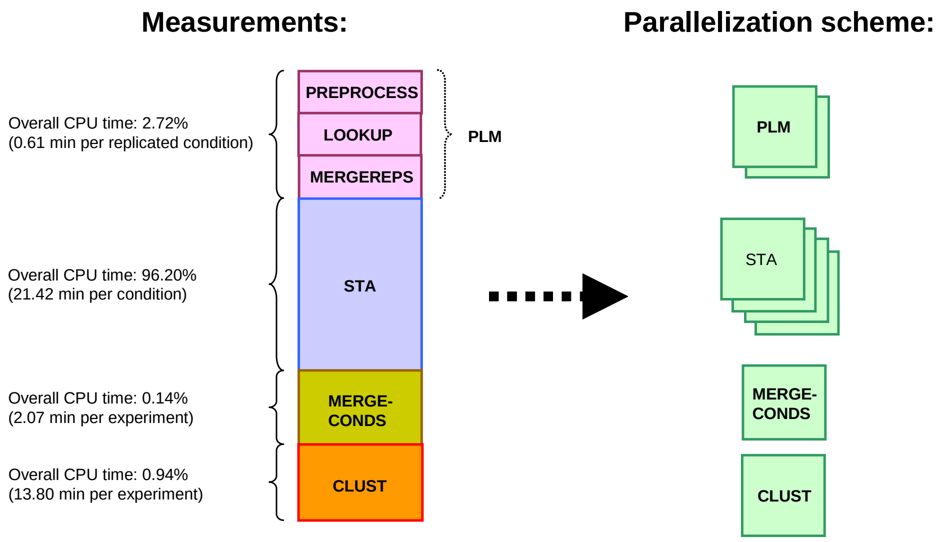


Figure 4.6.: Timing information for the microarray analysis pipeline

Before designing EPP, timing information for the microarray analysis pipeline has been collected by performing test runs for 96 different microarray slides. The averages shown above were derived from that timing information. As can be seen in Figure 4.6, *STA* consumes the major part of the overall CPU time (∼96%). *STA* consists of the programs *VERA* and *SAM* and performs some statistical analysis. *VERA* and *SAM* have to be executed sequentially. However, before *STA* can be invoked, the data has to be preprocessed. This is undertaken by the three programs *PREPROCESS*, *LOOKUP* as well as *MERGEREPS*. Together, these three scripts together consume ∼3% of the overall CPU time. *MERGECONDS* and *CLUST* can not be executed in parallel because they need to wait for the statistical analysis to be completed. This means that independently of the process design, there needs to be a synchronization point at the end of the statistical analysis (*STA*). As the first three steps, *PREPROCESS*, *LOOKUP* and *MERGEREPS*, are short-lived, they have been wrapped into one single program, *PLM*, to minimize the overhead incurred by *BioOpera*. Additionally, the degree of parallelism for *PLM* may be specified by the user. The main part of the overall CPU time is used by *STA*. Therefore, the process design should provide the highest possible degree of parallelism for this step. For each single experimental condition, a single *STA* is performed.

These considerations led to the process design described in the following section.

4.3.2. Components and Control Flow

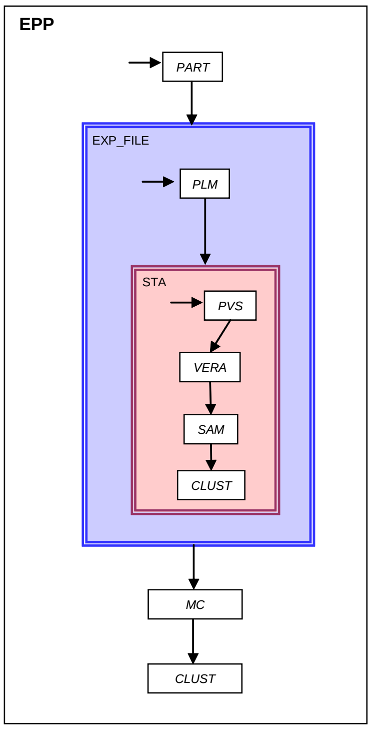


Figure 4.7.: Control flow of the expression profiling process (EPP)

As one can see in Figure 4.7, the expression profiling process (*EPP*) is based on the following tasks and subprocesses:

PART (Task)

Partitions the input data set into smaller subsets.

EXP_FILE (Subprocess)

Built from the following components: (1) *PLM*, (2) *STA* and (3) *CAST*.

PLM (Task)

Performs data normalization and background substraction for each single spot and computes average intensity data for replicated experiments. Also, it annotates each single spot with the corresponding gene information.

STA (Subprocess)

Built from the following components: (1) *PVS*, (2) *VERA* and (3) *SAM*.

PVS (Task)

Transforms the data in order for *VERA* and *SAM* to be invoked.

VERA (Task)

Computes a statistical error model based on the standardized data.

SAM (Task)

Determines, based on the previously computed error model, how likely each gene is differentially-expressed.

CAST (Task)

Casts its input parameter (array) to a string. This task is needed to overcome a limitation of the current *BioOpera* system.

MC (Task)

Merges different experimental conditions into one expression matrix.

CLUST (Task)

Clusters genes and experiments based on the different gene expression patterns represented in the expression matrix.

4.3.3. Data Partitioning (PART)

PART partitions the process' input data set into smaller subsets as defined by the user. The following enumeration lists its parameters (Figure 4.8) as well as the involved program:

INPUT

Deployment parameters:

data_root

Absolute path to the data.

script_root

Absolute path to the script invoked by *PART*.

Input file locations:

dapple_list

List containing all dapple files to be analyzed.

genekey_list

List containing the corresponding genekey files.

mergereps_list

List containing the output file names used to save the data of the merged replicated experiments.

Various input parameters:

partition_size

Parameter defining the size of a single partition.

OUTPUT

Output file locations:

part_dapple_list[]

Array containing all partitioned subsets of dapple files produced by *PART*.

part_genekey_list[]

Array containing all partitioned subsets of genekey files produced by *PART*.

part_mergereps_list[]

Array containing all partitioned subsets of mergereps output file names produced by *PART*.

INVOKED PROGRAM

PARTITION.pl

see Appendix A.6.1

Access method

```
perl %script_root%PARTITION.pl %dapple_list% %genekey_list%
%mergereps_list% -partitionsize %partition_size% -dataroot
%data_root%
```

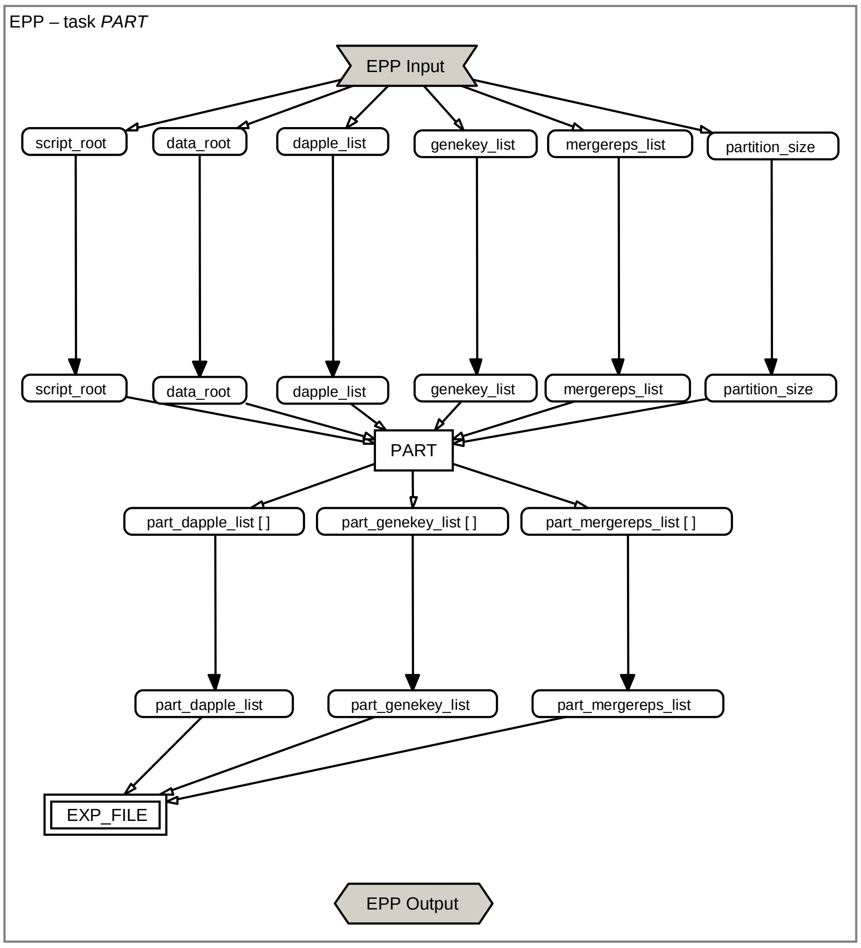


Figure 4.8.: Data flow of the expression profiling process (EPP) - *PART* Task

For a better understanding of the partitioning scheme, consider a biological experiment performed on four different conditions (C1 - C4). Conditions 1, 3 and 4 are replicated. Figure 4.9 shows the resulting partitions when setting the partition size to two. With this parameter, the user may specify the degree of parallelism of the next involved subprocess *EXP_FILE*. The mapped arrangement of the conditions is in accordance to the arrangement in the input files, taken by the task *PART*. The input passed as *dapple_list* and *genekey_list* is both a file containing a list of file names, one per condition and replica. The input passed by the third parameter (*mergereps_list*) is a file containing also a list of file names, but only one file name per condition.

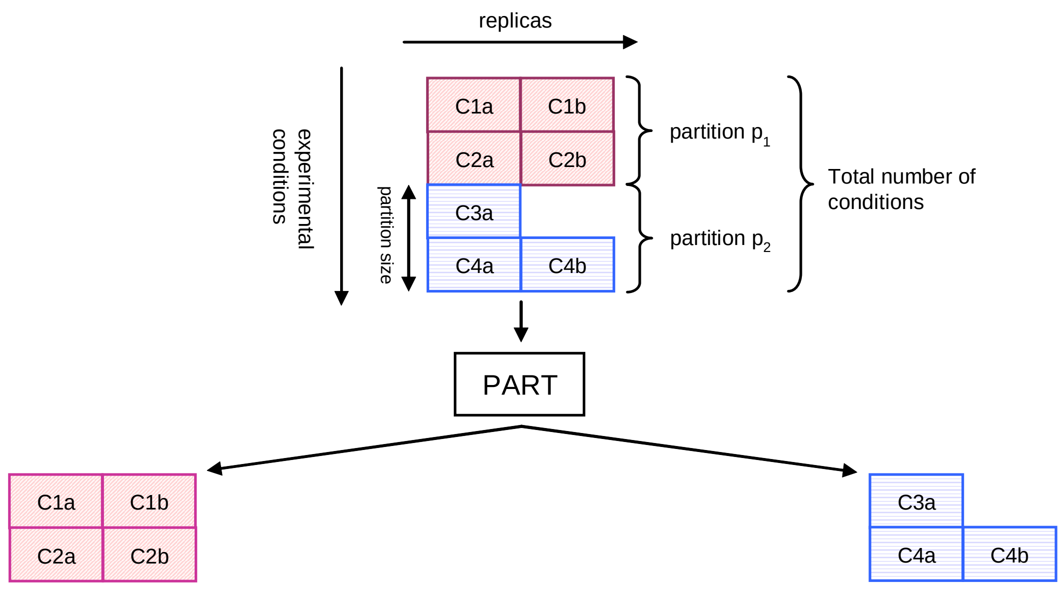


Figure 4.9.: Partitioning scheme of the *PART* task. The parameters hold only the respective file name. The files contains the data diagrammed as conditions.

4.3.4. Preprocessing (EXP_FILE)

Figure 4.11 as well as 4.12 shows the data flow of the *EXP_FILE* subprocess. The first task, *PLM*, converts the raw intensity for each single replicated experimental condition into a sorted list of background-substracted and normalized intensities for each spot on the cDNA microarray. After these mathematical transformations, each spot is being annotated with the corresponding gene information taken from the genekey file. At the end, replicated measurements are merged (Figure 4.10) by taking the average intensities as well as filtered to reject outliers [39]. The output is written to the file location indicated in *part_mergereps_list*. The following list summarizes the parameters as well as the program being involved in *PLM*:

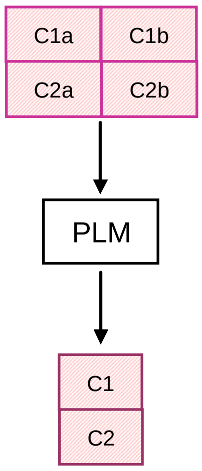


Figure 4.10.: Merging scheme of the *PLM* task. The replicas of Conditions 1 and 2 (C1a, C1b, C2a and C2b) are merged into a separate expression matrix (C1 and C2).

INPUT

Deployment parameters:

data_root

Absolute path to the data.

script_root

Absolute path to the script invoked by *PLM*.

Input file locations:

part_dapple_list

List containing all dapple files to be analyzed.

part_genekey_list

List containing the corresponding genekey files.

part_mergereps_list

List containing the output file names used to save the data of the merged replicated experiments.

Various input parameters:

scanner_saturation

Specify the saturating intensity for the microarray scanner. Spot intensities above this number are flagged, i.e. the specific spot is excluded from the analysis.

min_replications

Only those genes that are represented by at least the given number of replicate measurements in the merged data set are returned.

const_label_dir

Channel 1 usually represents the control condition and channel two the treated one. With this parameter one can define this order. In the former case, the parameter should hold the value 'r'. If channel 1 represents the treated condition and channel 2 the control one, then one should set the value 'f'. The correct channel assignment is very important, as the ratio of treated condition/control condition is taken.

unique_id

Some of the involved scripts exchange information through a file. When several *PLM* tasks are executed on the same machine in parallel, each of them needs to write to its own file. Therefore, the *PLM* task needs a unique identification to name its log file.

OUTPUT

Output file locations:

merged_files[]

Array containing the preprocessed and merged experimental conditions.

Various output parameters:

max_replications[]

This array holds the maximum numbers of replicated measurements for each condition.

INVOKED PROGRAM

Preproc_lookup_mergereps.pl

see Appendix A.6.2

Access method

```
perl %script_root%PREPROC_LOOKUP_MERGEREPS.pl
%part_dapple_list% %part_genekey_list% %part_mergereps_list%
-dataroot %data_root% -scriptroot %script_root% -minreps
%min_replications% -labeldir %const_label_dir% -sat
%scanner_saturation% -uid %unique_id%
```

As one can see from Figure 4.12, the subprocess *EXP_FILE* is built from a second task, *CAST*. However, before this task will be executed, the subprocess *STA* will be invoked first. Therefore the design description is continuing with this subprocess. The task *CAST* is described in subsection 4.3.6.

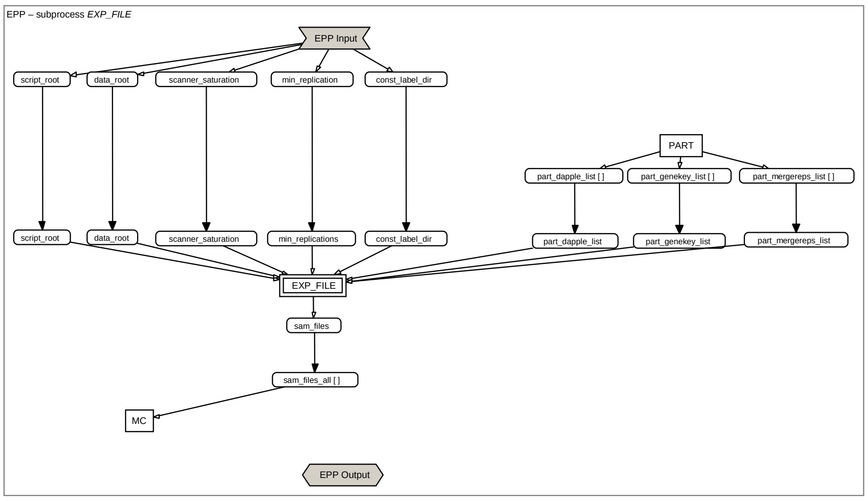


Figure 4.11.: Data flow of the expression profiling process (EPP) - *EXP_FILE* Subprocess

4.3.5. Statistical Analysis (STA)

Figure 4.13 shows the data flow of the subprocess *STA*. This subprocess determines, for each condition, wether any given gene is expressed at a different level in one cell population than in another, according to the microarray data (Subsection 2.1.2). It is built from three different tasks: (1) *PVS*, (2) *VERA* as well as (3) *SAM*.

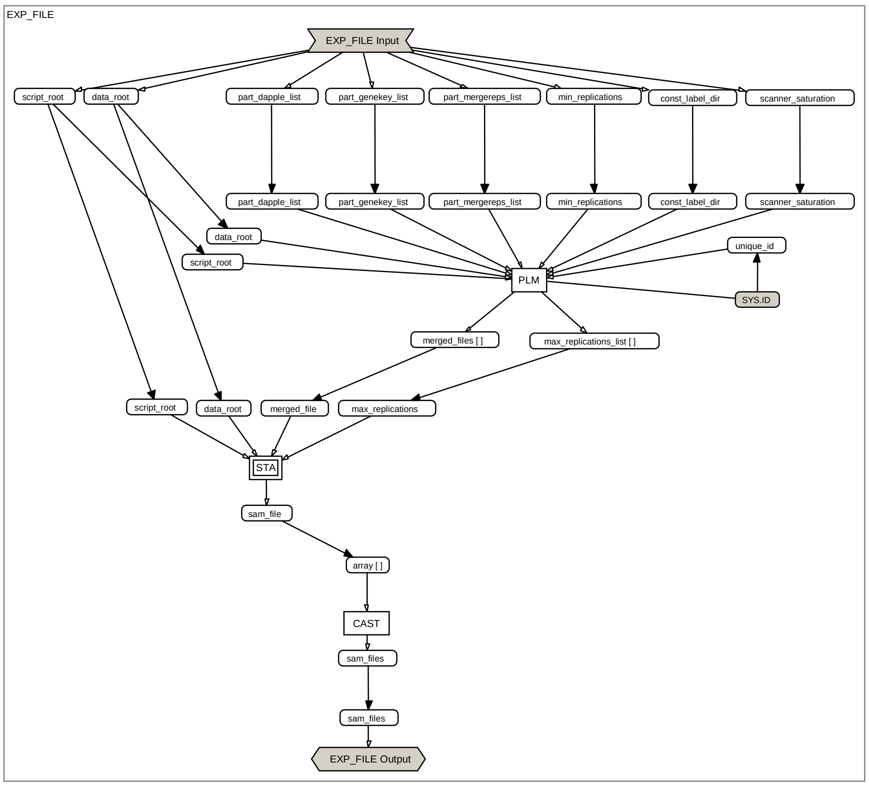


Figure 4.12.: Data flow of the *EXP_FILE* subprocess

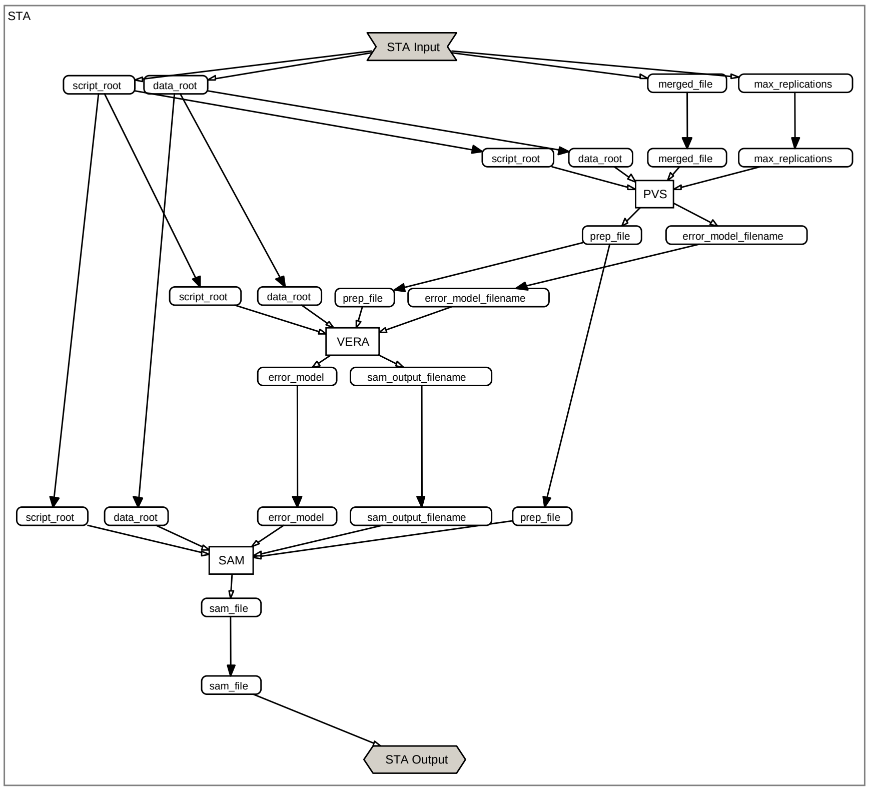


Figure 4.13.: Data flow of the *STA* subprocess.

Transforming File Format (PVS)

PVS transforms the output data produced by the preliminary steps into the right format. The following list shows the parameters as well as the program involved by *PVS*:

INPUT

Deployment parameters:

data_root

Absolute path to the data.

script_root

Absolute path to the script invoked by *PVS*.

Input file locations:

merged_file

Expression matrix of merged replicated experiments (i.e. one single condition) produced by *PLM*.

Various input parameters:

max_replications

Indicates the maximum number of replicated spots on the microarray. This value is used in the data standardization.

OUTPUT

Output file locations:

prep_file

Standardized expression matrix of a single condition.

error_model_filename

File name for the error model produced in the next subsequent step (*VERA*).

INVOKED PROGRAM

PREPARE_VERA_SAM.pl

see Appendix A.6.3

Access method

```
perl %script_root%PREPARE_VERA_SAM.pl %merged_file%
-dataroot %data_root% -replications %max_replications%
```

Variability and Error Assessment (VERA)

VERA takes the merged experimental data produced by *PLM*, and describes for each condition the overall variability of the data in terms of five parameters, called error model parameters. Error model parameters are fitted to the data by starting from an initial guess, optimizing them in iterated steps until they converge [32]. *VERA* involves the following parameters and program:

INPUT

Deployment parameters:

data_root

Absolute path to the data.

script_root

Absolute path to the script invoked by *VERA*.

Input file locations:

prep_file

Transformed expression matrix of a single condition.

error_model_filename

File name used to write out the computed error model.

OUTPUT

Output file locations:

error_model

Computed error model.

sam_output_filename

File name for the output produced in the subsequent step (*SAM*).

INVOKED PROGRAM

START_VERA.pl

see Appendix A.6.4

Access method

```
perl %script_root%START_VERA.pl %prep_file%
%error_model_filename% -dataroot %data_root% -scriptroot
%script_root%
```

Significance of Array Measurement (SAM)

SAM returns a value, lambda, for each gene present in a condition, which describes how likely it is that the gene is expressed-differentially in the two cell populations existing in a condition. A high lambda value indicates that the gene is differentially-expressed, while a low lambda value indicates that there is no evidence for differential expression [32]. The next step, *MERGECONDS*, allows the user to set a threshold for the lambda value. According to its lambda value, a particular gene will be considered for clustering. The appropriate threshold should be determined from control experiments. The following enumeration lists the parameters and the program involved by *SAM*:

INPUT

Deployment parameters:

data_root

Absolute path to the data.

script_root

Absolute path to the script invoked by *SAM*.

Input file locations:

error_model

Computed error model from the preliminary step (*VERA*). Used to compute a value, lambda, for each gene present in the expression matrix (*prep_file*).

prep_file

Transformed expression matrix of a single condition.

sam_output_filename

File name used to write out the modified expression matrix of one condition.

OUTPUT

Output file locations:

sam_file

Modified expression matrix of one condition complemented with a value, lambda, for each gene.

INVOKED PROGRAM

START_SAM.pl

see Appendix A.6.5

Access method

```
perl %script_root%START_SAM.pl %prep_file% %error_model%
%sam_output_filename% -dataroot %data_root% -scriptroot
%script_root%
```

4.3.6. Type Casting (CAST)

Figure 4.12 shows the data flow of *CAST*. The objective of this task is to overcome a limitation in the current system. It is not possible to copy an array into another. Therefore, *CAST* transforms its input array to a string containing the array elements separated by white space, thus allowing to copy the content of an array into another array. It involves the following parameters and the following program:

INPUT

Input file locations:

array[]

Array holding the output files produced by one explosive task *STA*.

OUTPUT

Output file locations:

sam_files

The same content as above, merged into a single string.

INVOKED PROGRAM

echo

Access method

```
echo "<sam_files>%array%</sam_files>"
```

4.3.7. Expression Matrix Generation (MC)

The task *MC* merges the separate experimental conditions into one 'global' expression matrix representing the whole biological experiment (Figure 4.14). Only those genes passing at least the defined lambda threshold, are listed in the expression matrix. Figure 4.15 describes the data flow of *MC*. It involves the following parameters and program:

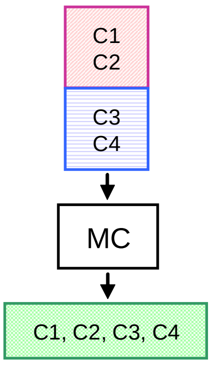


Figure 4.14.: Merging scheme of the *MC* task. The various conditions of a biological experiment (C1 - C4) are merged into a global expression matrix.

INPUT

Deployment parameters:

data_root

Absolute path to the data.

script_root

Absolute path to the script invoked by *MC*.

Input file locations:

sam_files_all

List containing the expression matrices for each single experimental condition of the overall experiment.

expression_matrix_name

File name used to write out the global expression matrix.

Various input parameters:

lambda_threshold

Threshold value, which have to be fulfill by each gene being listed in the mentioned above expression matrix.

OUTPUT

Output file locations:

merged_file

Global expression matrix representing the whole biological experiment.

INVOKED PROGRAM

MERGECONDS.pl

see Appendix A.6.6

Access method

```
perl %script_root%MERGECONDS.pl -out
%expression_matrix_name% -conds %sam_files_all% -dataroot
%data_root% -lam %lambda_threshold%
```

4.3.8. Clustering (CLUST)

CLUST represents the last step in the expression profiling process. Figure 4.16 shows its data flow. The task performs a hierarchical clustering of the various gene expression levels included in the expression matrix. *CLUST* involves the following input/output parameters and program:

INPUT

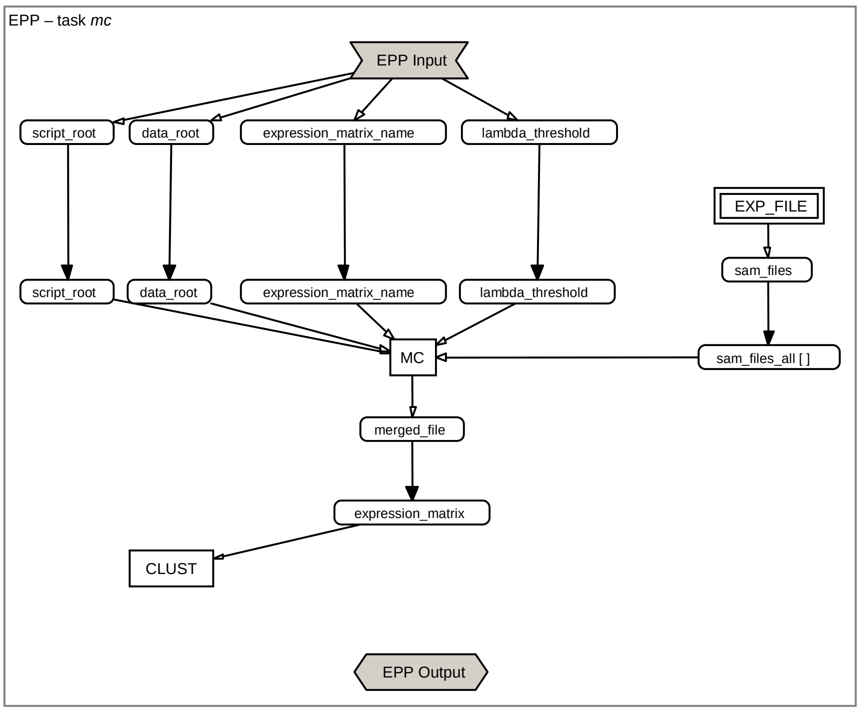


Figure 4.15.: Data flow of the expression profiling process (EPP) - *MC* Task

Deployment parameters:

data_root

Absolute path to the data.

script_root

Absolute path to the script invoked by *CLUST*.

Input file locations:

expression_matrix

Expression matrix representing the whole biological experiment.

Various input parameters:

gene_clustering

Indicates whether to cluster genes or not. 0 means no gene clustering, 1 means non-centered metric when clustering genes and 2 means centered metric when clustering genes.

experiment_clustering

Indicates whether to cluster experiments or not. See above for the possible values. One has to perform at least a kind of clustering, otherwise *CLUST* produces no output.

distance_metrics

Indicates whether to use pearson correlation (1) or Euclidean distance (0).

log_transform

Indicates whether to log transform (1) or not (0).

OUTPUT

Output file locations:

clustered_data_table_file

Output of *CLUST* containing the original data reordered based on the clustering result.

gene_tree_file

Output of *CLUST* that reports on the history of node joining during the gene clustering.

array_tree_file

Output of *CLUST* that reports on the history of node joining during the array clustering.

INVOKED PROGRAM

START_CLUSTER.pl

see Appendix A.6.7

Access method

```
perl %script_root%START_CLUSTER.pl %expression_matrix%
-dataroot %data_root% -scriptroot %script_root% -geneclust
%gene_clustering% -expclust %experiment_clustering%
-distmetrics %distance_metrics% -log %log_transform%
```

4.3.9. Process Output

The expression profiling process produces three different output files (Figure 4.16):

1. **clustered_data_table**

   Based on the clustering, the reordered expression matrix.

2. **gene_tree_file**

   Report on the history of node joining during the gene clustering. Gene clustering corresponds to cluster the rows (representing the genes).

3. **array_tree_file**

   Report on the history of node joining during the experiment clustering. Experiment clustering corresponds to cluster the columns (representing the different experimental conditions).

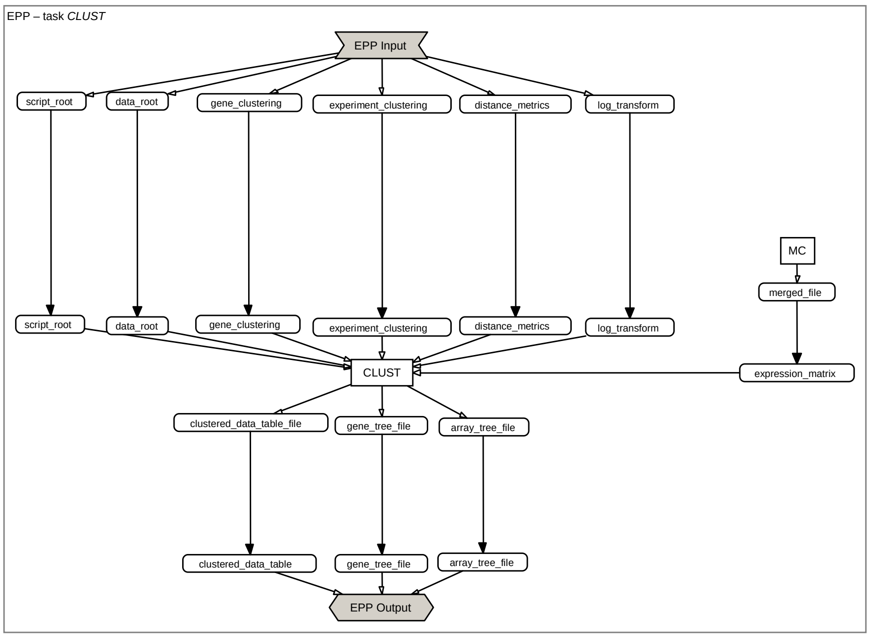


Figure 4.16.: Data flow of the expression profiling process (EPP) - *CLUST* Task

These three files describe a single hierarchical tree containing various clusters of similar expression profiles across experimental conditions. The hierarchical tree can be visualized by TreeView [45]. Figure 4.17 shows an example output of the expression profiling process visualized by TreeView. The dendrogram at the top lists the samples studied in the used experimental data set and provides a measure of the relatedness of gene expression in each sample. The dendrogram is color coded according the category of mRNA sample studied (see left key). The results presented represent the ratio of hybridization of fluorescent cDNA probes prepared from each experimental mRNA samples to a reference mRNA sample (control sample). These ratios are a measure of relative gene expression in each experimental sample and were depicted according to the color scale shown at the bottom.

4.3.10. Exploiting Parallelism

The main data flow of the expression profiling process (*EPP*) is best explained on the basis of an example microarray experiment.

Assume an experiment consists of four different experimental conditions (C1 - C4), of which conditions 1, 2 and 4 have been replicated. Figure 4.18 shows the data flow during the execution of *EPP* for this experiment. Through the input parameter *parti-*

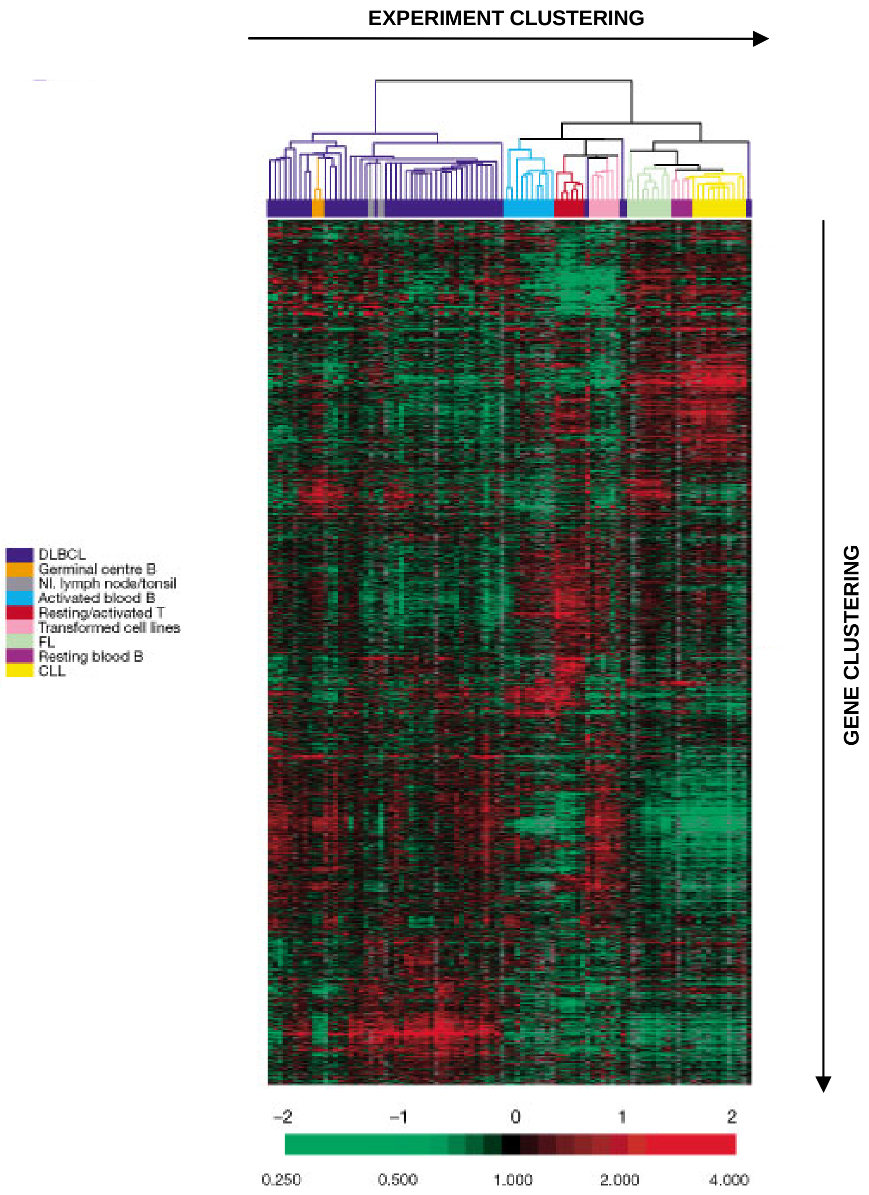


Figure 4.17.: Example Output of the expression profiling process visualized by Tree-View. The data has been cluster by experiment clustering. (Modified from Alizadeh, A.A. et al., Nature 403, 2000)

tion_size, the user may define the degree of parallelism of the subprocess *EXP_FILE*. With a partition size of two, the input is split into two separate partitions, p_1 and p_2, each of them containing two conditions. Within each instance of *EXP_FILE* (1 and 2), for each condition, generated by the task *PLM*, a separate copy of the subprocess *STA* is started. The task *CAST* joins the conditions, output by the two instance of *STA*. All experimental conditions are then merged into one expression matrix by the task *MC*. Starting the correct number of copies of *EXP_FILE* as well as *STA* at each case, is mastered by *BioOpera*'s explosive task module.

Figure 4.19 shows the main data flow for the general case, from which the correct data flow for a particular experiment can be deduced.

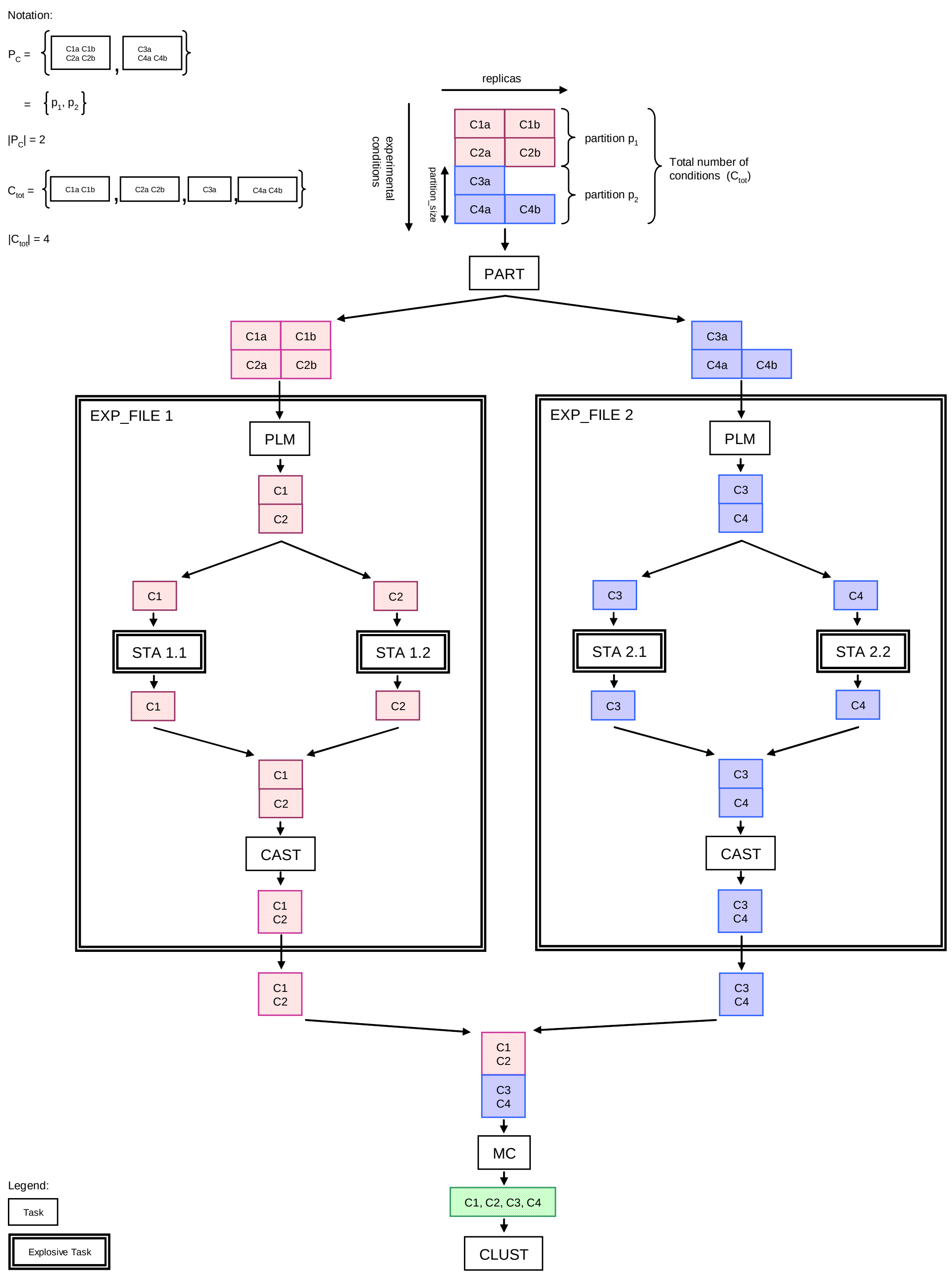


Figure 4.18.: Example data flow of the expression profiling process (EPP)

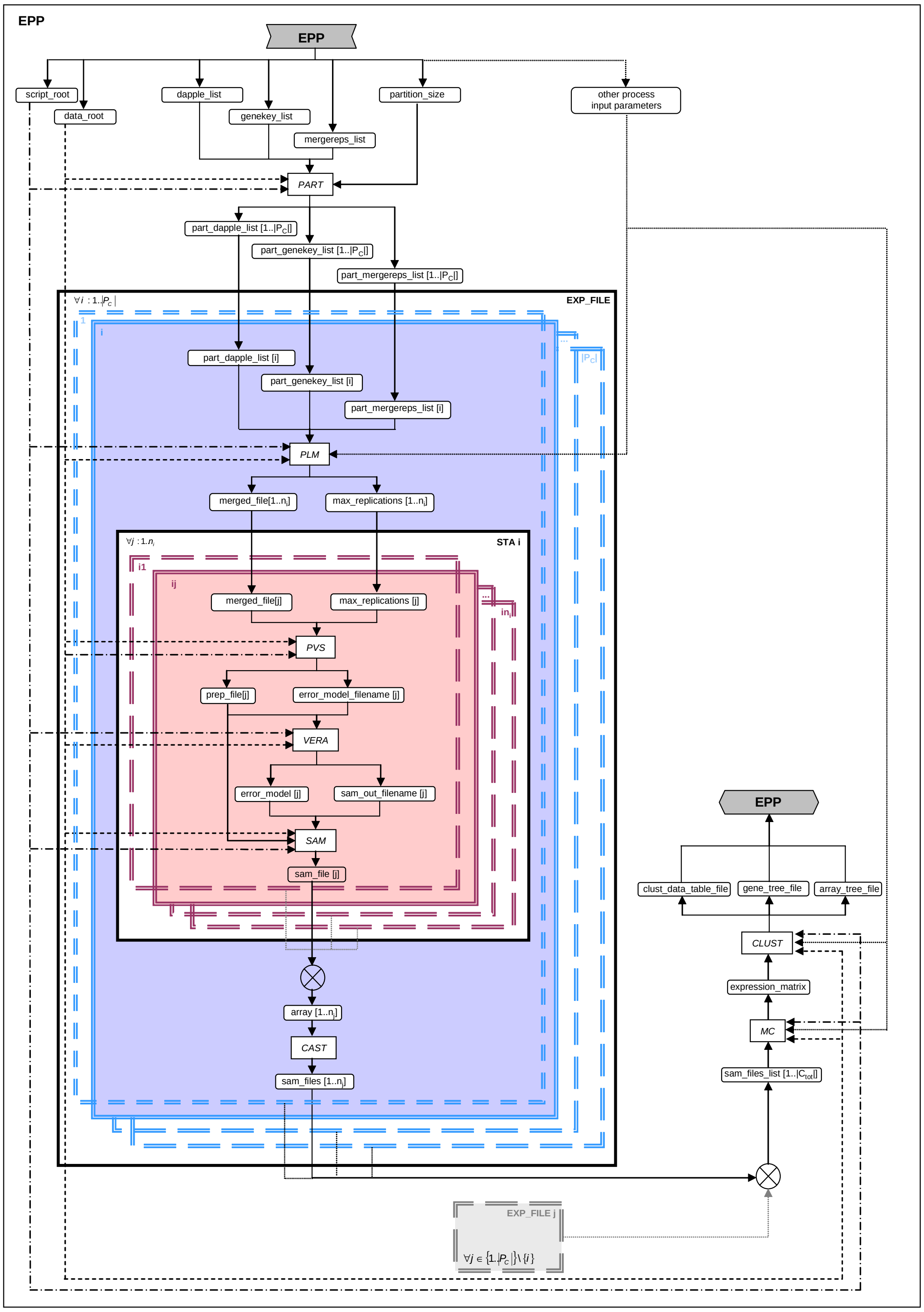


Figure 4.19.: Main data flow in the expression profiling process (EPP)

5. Measurements

5.1. Scalability

We have been testing the scalability of *BioOpera* using the EPP process. The most time consuming step in the analysis (∼96% of the overall CPU time) is the computation of the likelihood that a specific gene is differentially-expressed. This likelihood needs to be calculated for each experimental condition, each of which may be analyzed independently of the other. Our data set consisted of 66 different conditions (Section 2.3). The degree of parallelism between tasks is left to *BioOpera* to decide.

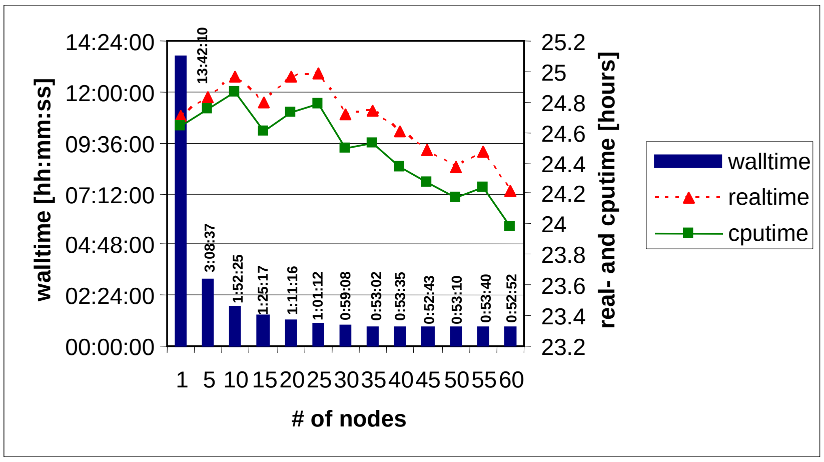


Figure 5.1.: Scalability measurements

Figure 5.1 shows the result of the test run. During the run, the cluster L (Section 4.1) was exclusively used by *BioOpera*. The left vertical axis shows the WALL time (time from beginning to end of the computation), the right vertical axis the CPU time (time spent computing) and REAL time (time spent in each node including computing and I/O waits). The horizontal axis indicates the number of nodes, each of them with 2 CPUs. The result prove that the process scales well up to 35 available nodes. Beyond 35 nodes, however, there is no improvement to be observed. This is due to the fact that *BioOpera* always schedules two concurrent tasks on a single cluster node so that, for EPP, beyond 35 nodes there is no more parallelism to be exploited. Overall, the experiment demonstrates that *BioOpera* can be used to parallelize computations and obtain performance gains without having to become familiar with sophisticated programming techniques.

5.2. Overhead Incurred by Short-Lived Tasks

Preprocessing the microarray data (normalization, background substraction, gene annotation and merge replicas) is a short-lived task (consuming altogether 0.61 minutes per replicated condition). On the one hand, we have therefore wrapped these single steps into one task, *PLM*. On the other hand, we provide the user the possibility to define the partition size in order to minimize the overhead incurred by *BioOpera*. To get a feeling for this overhead, we have performed test runs on a single host, varying the partition size. We used data originating from 96 different microarray slides representing 66 different conditions (after merging the replicated conditions).

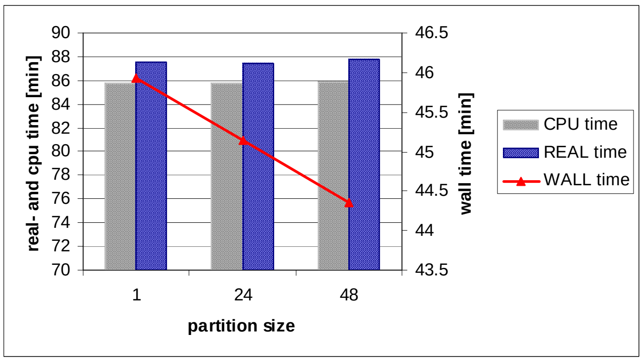


Figure 5.2.: Overhead measurements

Figure 5.2 shows the result of the test runs. Each test run was performed on a single dual processor machine, since we were interested in measuring the *BioOpera* overhead for starting an activity. The left vertical axis shows the CPU time (time spent computing) and REAL time (time spent in the node including computing in I/O waits), the right vertical axis the WALL time (time from beginning to the end of computation). The horizontal axis indicates the partition size. The times were taken for the two tasks *PART* and *PLM*. The result show that the WALL time decreases by increasing the partition size, as expected. This is due to the fact that a larger partition size results in a smaller number of parallel *PLM* tasks, that need to be scheduled and dispatched by *BioOpera*. For our data, a partition size of one results in 96 *PLM* tasks being started in parallel, a partition size of 24 results in 4 *PLM* tasks and a partition size of 48 results in only 2 tasks. One can say that in general, for explosive tasks consisting out of short-lived execution units, it makes sense to tune the partitioning scheme to generate as many jobs as there are available hosts, in order to minimize overhead. However, the WALL time gain obtained by this adjustment is relatively small.

6. Conclusions

Especially in the area of microarray technology, the focus has shifted from data generation to data analysis. Biologists are therefore more and more confronted with limitations in terms of computer skills and computing infrastructure. The concept of process as used within *BioOpera* is one possible starting point to bridge the gap between biology and computer science. Biologists and bioinformaticans implement programs, each of them solving a partial task in the overall analysis. *BioOpera* helps integrating these programs and takes care of efficiently parallelizing their execution. The implemented processes can be seen as a proof of concept.

Figure 6.1 compares the clustered experimental conditions produced by EPP (a) to the original clustered experimental conditions (b). A rudimentary similarity between the two hierarchical trees can be observed. However, our results are not directly comparable to the original ones. This is due to the fact that EPP uses a maximum-likelihood approach (MLA). MLA requires that a single experimental condition has been replicated at least four times. This constraint is not fulfilled by our data set. In addition, we used a hypothetical lambda value as threshold to determine the genes that are differentially-expressed, though this value should be determined by a control experiment. Such a threshold value was not provided because the original analysis was not performed using the MLA. That implies that we are "loosing" a high number of data points (expression ratios) during the analysis.

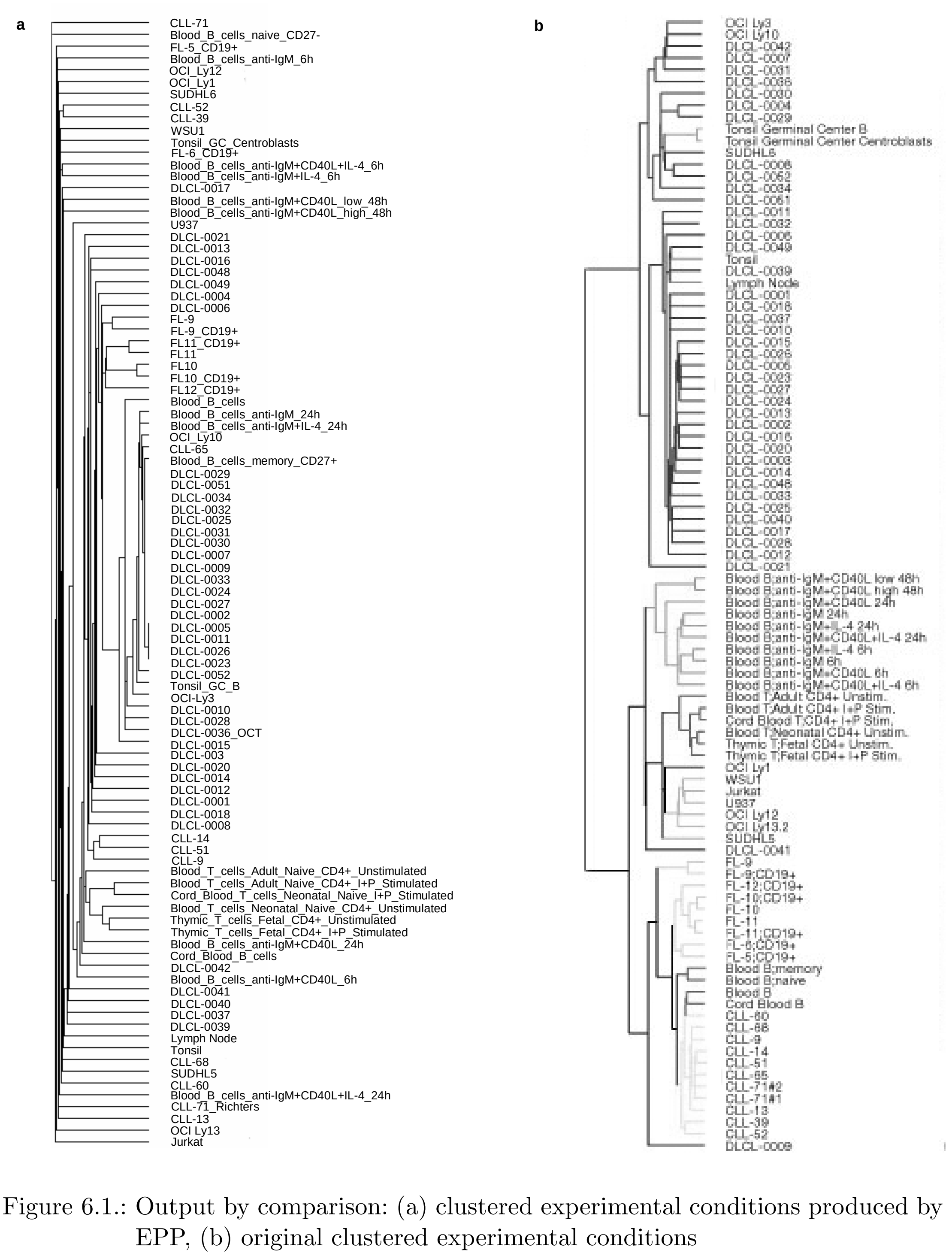


Figure 6.1.: Output by comparison: (a) clustered experimental conditions produced by EPP, (b) original clustered experimental conditions

A. Technicalities

A.1. Example Dapple File Format

3	2													
0	0	0	0	A	235	126	13	A	782	590	7.9	156	1152	0.5677
0	0	0	1	A	749	123	26.3	A	2041	546	28.9	156	973	0.4187
0	0	0	2	A	959	122	20.7	R	8091	558	36.2	156	976	0.1121

Table A.1.: Example of a *dapple* output file for a microarray chip which consists of 3 spots and which is based on the two channel technology

The first line of the example output file, as shown in table A.1, states the number of spots (3) and the number of channels quantified per spot (3). Each subsequent line represents a single spot. It has the following values, separated by a single space:

1. column

   - X grid coordinate of the spot.

2. column

   - Y grid coordinate of the spot.

3. column

   - X spot coordinate of the spot.

4. column

   - Y spot coordinate of the spot.

Figure A.1 explains the difference between the spot and grid coordinates.

5. column

   - Rated quality of the first channel[1]. The following values may be assigned:

[1]In practice the first channel usually represents the green channel

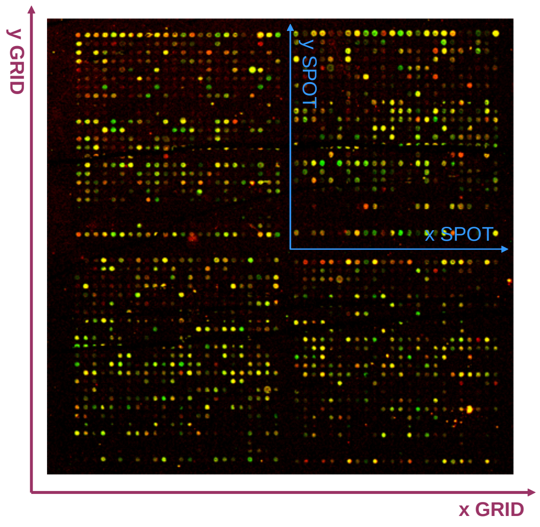


Figure A.1.: *Dapple*'s grid and spot coordinates - The example microarray image shown consists of 4 grids. Each grid contains 24x24 spots. The quantitation of the spots will be performed grid wise. *Dapple* starts with the grid which is placed in the upper-left-hand corner and follows first the y-axis towards the bottom of the image. This procedure will be repeated for each grid column. Inside a grid the image analysis starts in the upper-left-hand corner as well and follows the same pattern as described for the grids.

(1) "A" stands for "Accept"

(2) "S" stands for "Suspicious"

(3) "R" stands for "Reject".

6. column

   - Foreground intensity (mean or median)[2] of the first channel.

7. column

   - Local background intensity (median) of the first channel.

8. column

   - Standard deviation of the background pixel population of the first channel[3].

9. column

   - Rated quality of the second channel[4]. The following values may be assigned:

     (1) "A" stands for "Accept"

     (2) "S" stands for "Suspicious"

     (3) "R" stands for "Reject".

10. column

    - Foreground intensity (mean or median)[5] of the second channel.

11. column

    - Local background intensity (median) of the second channel.

12. column

    - Standard deviation of the background pixel population of the first channel.

13. column

    - The ratio of first to second channel intensity[6] according to the following equation:

$$\frac{foreground\ intensity\ CH1\ -\ background\ intensity\ CH1}{foreground\ intensity\ CH2\ -\ background\ intensity\ CH2} \tag{A.1}$$

[2]The foreground intensity may be set to the mean or median intensity inside the spot area

[3]The standard deviation of the background pixel population will not be used in further analysis

[4]In practice the second channel usually represents the red channel

[5]The foreground intensity may be set to the mean or median intensity inside the spot area

[6]The ratio will only be calculated if the underlying microarray chip is based on two channels

A.2. Example ScanAlyze File Format

HEADER	SPOT	GRID	TOP	LEFT	BOT	RIGHT	ROW	COL	CH1I
REMARK SOFTWARE ScanAlyze									
REMARK SOFTVERS 2.3									
REMARK CH1 IMAGE lc7b059_532 nm									
REMARK CH2 IMAGE lc7b059_635 nm									
REMARK GRID FILE /home/sreto/lc7b059_finalflagged.SAG									
REMARK DATE 8/27/99									
REMARK TIME 11:51:15 PM									
SPOT	1	1	83	58	97	72	1	1	235
SPOT	2	1	82	75	96	89	1	2	749
SPOT	3	1	82	92	96	106	1	3	959

Table A.2.: Example of a *ScanAlyze* output file - part 1. The underlying microarray chip consists of 3 spots and is based on the two channel technology.

CH1B	CH1AB	CH2I	CH2B	CH2AB	SPIX	BGPIX	EDGE	RAT2	MRAT
126	130	782	590	629	156	1152	0	1.761	2.301
123	126	2041	546	573	156	973	0	2.388	2.445
122	127	8019	558	583	156	976	0	8.914	8.829

Table A.3.: Example of a *ScanAlyze* output file - part 2

REGR	CORR	LFRAT	CH1GTB1	CH2GTB1	CH1GTB2	CH2GTB2
1.344	0.6182	3.04	0.7756	0.7051	0.4808	0.3269
2.102	0.94	2.331	0.7179	0.7949	0.5769	0.5897
0	0	0.1632	0.8269	0.8205	0.609	0.641

Table A.4.: Example of a *ScanAlyze* output file - part 3

The first line of the example output file, as shown in tables A.2 - A.5, annotates the columns. The next 7 lines contain remarks about the version of the program used, the wavelengths for gathering the raw intensities and others. Each subsequent line represents the data of a single spot. It has the following values, separated by a single tab:

CH1EDGEA	CH2EDGEA	FLAG	CH1KSD	CH1KSP	CH2KSD	CH2KSP
-0.0102	0.007381	0	0.4077	1.05E-20	0.223	1.71E-06
-0.008429	0.004306	0	0.5228	4.64E-33	0.492	2.49E-29
0.008815	0.01661	1	0.526	1.81E-33	0.5636	2.28E-38

Table A.5.: Example of a *ScanAlyze* output file - part 4

1. column - HEADER
   - Defines the type of the data in a row. Possible values are:
     (1) HEADER
     (2) REMARK
     (3) SPOT
2. column - SPOT
   - Unique index of the spot in the file. Counting starts with the gird which is placed in the upper-left-hand corner, moves along the first row from the first column until the last column, then advances to the second row; after all rows in the first grid are assigned an index, counting proceeds to the second grid, etc.
3. column - GRID
   - Represents the number of the grid in which the spot is contained. The grid in the upper-left-hand corner is assigned the value 1. Counting continues along the first row to the last column, then advances to the second row, etc.
4. column - TOP
   - Top coordinate of the box containing the spot ellipse, in image coordinates.
5. column - LEFT
   - Left coordinate of the box containing the spot ellipse, in image coordinates.
6. column - BOT
   - Bottom coordinate of the box containing the spot ellipse, in image coordinates.

7. column - RIGHT

   - Right coordinate of the box containing the spot ellipse, in image coordinates.

8. column - ROW

   - Tells you in which row the spot is located within the grid.

9. column - COL

   - Tells you in which column the spot is located within the grid.

Figure A.2 explains the sequence how *ScanAlyze* quantifies the spots.

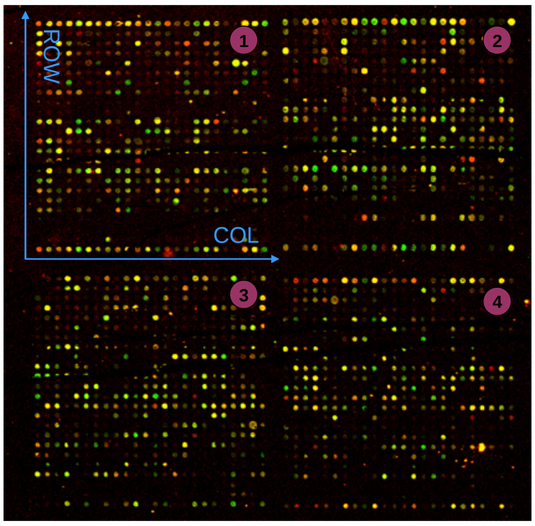


Figure A.2.: *ScanAlyze*'s row and column coordinates - The example microarray image shown consists of 4 grids. Each grid contains 24x24 spots. The quantitation of the spots will be performed grid-wise. *ScanAlyze* starts with the grid which is placed in the upper-left-hand corner (number 1) and advances then to the number 2, etc. Within each grid the sequence for the spot quantization is the following: the program starts with that spot which is located in the upper-left-hand corner, moves along the same row to the last column (column by column) and advances then to the second row, etc.

10. column - CH1I

    - The uncorrected mean pixel intensity for the first channel.

11. column - CH1B

    - The estimated median intensity of the background pixels for the first channel.

12. column - CH1AB

    - The estimated mean intensity of the background pixels for the first channel.

13. column - CH2I

    - The uncorrected mean pixel intensity for the second channel.

14. column - CH2B

    - The estimated median intensity of the background pixels for the second channel.

15. column - CH2AB

    - The estimated mean intensity of the background pixels for the second channel.

16. column - SPIX

    - Count of the number of pixels contained in the spot.

17. column - BGPIX

    - The number of background pixels used for estimating the local background.

18. column - EDGE

    - Not used.

19. column - RAT2

    - The ratio of second to first channel intensity according to the following equation:

$$\frac{foreground\ intensity\ CH1I\ -\ background\ intensity\ CH1B}{foreground\ intensity\ CH2I\ -\ background\ intensity\ CH2B} \tag{A.2}$$

While the amount of DNA often varies considerably across a spot, the background corrected ratio is fairly constant. *ScanAlyze* utilizes this property in alternate estimates of the ratio and spot quality. One alternate estimate of the ratio for a spot is the median of the set of background corrected single pixel ratio for all pixels within the spot. Unlike pixel intensities, the background corrected pixel ratios are expected to be uniform, and thus the median is a useful value.

20. column - MRAT

    - The exported column MRAT contains the median of

$$\frac{Ch2PI\ -\ CH2B}{Ch1PI\ -\ CH1B} \tag{A.3}$$

where Ch1PI and Ch2PI represent single pixel intensities. The primary utilization of this value is that it is less susceptible to artifacts which can corrupt mean intensity based ratios, such as bright fluorescent specks.

Two additional estimates of the ratio are based upon the assumption that, when single pixel intensities in channel 1 are plotted against intensities in channel 2, they will fall on a straight line with slope equal to the ratio. This assumption is based on a model for single pixel intensities that assumes a constant background in each channel and varying DNA content in each pixel. Thus, for a spot representing a gene with cognate probe in the hybridization solution having a red/green ratio of R, the channel 1 (green) and channel 2 (red) intensities at each pixel will be

$$Ch1BG + k * D \tag{A.4}$$

and

$$Ch2BG + R * k * D \tag{A.5}$$

respectively, where Ch1BG and Ch2BG are the uniform backgrounds in the two channels, k is a constant and D is the amount of DNA in the region covered by the pixel. Thus, when channel 1 pixel intensities are plotted against channel 2 pixel intensities, they will fall along a line with slope k passing through the point (Ch1BG, Ch2BG).

21. column - REGR
    - Estimates the slope of the above mentioned line by simple linear regression of channel 2 on channel 1.
22. column - CORR
    - Contains the correlation between channel 1 and channel 2 pixels within the spot, and is a useful quality control parameter (in general, high values imply better fit and good spot quality).
23. column - LFRAT
    - Estimates the slope of the above mentioned line by a least-squares fit of a line to the points, minimizing the sum of the squares shortest distance from each point to the line.

24. column - CH1GTB1
    - Additional quality parameter. Fraction of pixels in the spot greater than the background of channel 1 (CH1B).
25. column - CH2GTB1
    - Additional quality parameter. Fraction of pixels in the spot greater than the background of channel 2 (CH2B).
26. column - CH1GTB2
    - Additional quality parameter. Fraction of pixels in the spot greater than 1.5 times than the background of channel 1 (CH1B).
27. column - CH2GTB2
    - Additional quality parameter. Fraction of pixels in the spot greater than 1.5 times than the background of channel 2 (CH2B).

On a perfectly clean image with no spotted DNA, roughly half of the pixels in a randomly placed spot should have intensities greater than the local background and few pixels much greater than this background. Thus, an empty spot will have CH1GTB1 and CH2GTB1 values of close to 0.50 and CH1GTB2 and CH2GTB2 values close to 0, while for uniformly bright spots these values should all be close to 1. This can be used to filter out weak spots and spots where the intensity comes from a few bright pixels (these will have high mean intensities but low values of these parameters). Cutoff values of between 0.55 and 0.65 are recommended.

28. column - CH1EDGEA
    - Mean magnitude of the horizontal and vertical Sobel edge vectors contained within each spot for channel 1.
29. column - CH2EDGEA
    - Mean magnitude of the horizontal and vertical Sobel edge vectors contained within each spot for channel 2.

These two values are used during refinement. It is expected that a good spot should have relatively high mean edge scores.

30. column - FLAG
    - User defined spot flag. The default value is 0.

An alternative method for capturing the same information like CH1GTB1, CH2GTB1, CH1GTB2 and CH2GTB2 is output in the following four columns. These values compare the distribution of pixel intensities within the spot circle and in the background.

31. column - CH1KSD

    - This is the value of the Komogorov-Smirnov statistic for the channel 1 that assesses the likelihood that the spot pixel intensity distribution is drawn from the background distribution.

32. column - CH1KSP

    - The actual probability for CH1KSD.

33. column - CH2KSD

    - This is the value of the Komogorov-Smirnov statistic for the channel 2 that assesses the likelihood that the spot pixel intensity distribution is drawn from the background distribution.

34. column - CH2KSP

    - The actual probability for CH2KSD.

A.3. Transforming ScanAlyze File Format Into Dapple File Format

This section describes the transformation of the two file formats in more detail. The perl script *TRANSFORM.pl* takes as input a *ScanAlyze* file and produces as output a *dapple* file. The following columns from the input file are considered:

- CH1I
- CH1B
- CH2I
- CH2B
- SPIX
- BGPIX
- FLAG

As *ScanAlyze* does not export the standard deviation of the background pixel population of the first and second channel, the script populates the respective values with a zero for each spot and channel (8. and 12. column). This procedure is for the further analysis irrelevant, because these values are not used. The last column in the *dapple* output files contains the background subtracted ratio of first to second channel. *ScanAlyze* does not provide this ratio. Therefore, the ratio is calculated according equation A.1. If the numerator is equal to zero the ratio gets the string "Undefined". In the software tool *ScanAlyze*, the user does have the possibility to flag manually a single spot[7], whereas the user defines the flag value[8]. In *dapple*, a spot can be flagged in channel 1 or channel 2, *ScanAlyze* does not support this functionality, a spot can only be flagged as a whole. If the transformation script encounters a flagged spot in the input file then each of the two channels will be flagged in the output file. In the experimental dataset used we are interested in the ratio of the second to the first channel (in order to state if a particular gene has been up- or downregulated or if the expression level has not been changed), therefore it is legitimate to flag both of the two channels in the output file. With only one channel flagged, one is not able to build the above mentioned ratio. The flag value differs from the original file format. An accepted spot gets the value zero, a rejected spot in contrast gets one. As we have learned from the detailed description about the two different file formats, the quantitation sequence in *ScanAlyze* and *dapple* is different. In order to obtain the right *dapple* sequence based on the *ScanAlyze* sequence, we must pass the array geometry to the script as a command line argument:

- arrayconfig

  $< number\ of\ grid\ columns >$

  $< number\ of\ grid\ rows >$

  $< number\ of\ spot\ columns\ within\ a\ single\ grid >$

  $< number\ of\ spot\ rows\ within\ a\ single\ grid >$

Table A.6 shows you an example *dapple* output file produced by the transformation script. As input, the same *ScanAlyze* file as shown in tables A.2 - A.5.

3	2													
0	0	0	0	0	235	126	0	0	782	590	0	156	1152	0.5677
0	0	1	0	0	749	123	0	0	2041	546	0	156	973	0.4187
0	0	2	0	1	959	122	0	1	8091	558	0	156	976	0.1121

Table A.6.: Example *dapple* output file produced by the script *TRANSFORM.pl*. The highlighted values are different to the original file format.

[7]Usually, a flagged spot will be unaccounted for the further analysis

[8]For the experiment dataset used in this diploma thesis, the user defined flag value is equal to 1

A.4. Generating Genekey File

The perl script *EXTRACT.pl* generates a genekey file, as shown in table A.7, as the experiment dataset does not provide such files, by extracting the gene information for each microarray from the respective *SMD* raw data file.

num_rows_per_slide 1				
0	0	160882	AA768786	KIAA0130
0	1	160886	AA805528	PPP3CC
0	2	160890	AA806043	IGHM

Table A.7.: Example genekey file generated by the script *EXTRACT.pl*. This file provides the required gene information to annotate the raw data shown in table A.6.

The first line specifies the number of rows of spots on the microarray slide with the keyword "num_rows_per_slide". Each of the following rows lists the mapping between a particular spot position and a the corresponding gene information using four columns:

1. column
   - Microarray row.
2. column
   - Microarray column.

Row and column numbering starts with the coordinates (0,0) in the upper-left-hand corner of the microarray slide and moves towards the lower right.

3. column
   - SUID - *Stanford University* identification number which is a unique number.
4. column
   - GeneBank[9] accession number. With this accession number you will get immediately additional information about the underlying gene inclusive the whole sequence string.
5. column
   - Gene symbol.

[9]The *Entrez Nucleotides* database is a collection of sequences from several sources, including GeneBank, RefSeq, and PDB

The genekey file thus generated will be merged with the raw data in the proper expression profiling process during the lookup gene information step.

A.5. Scripts for the Data Preparation Process

A.5.1. Start.pl

The script *START.pl* takes the following arguments:

`<FileList>`

> List holding the filenames of the ScanAlyze files, the SMD files and the names of the corresponding output files. Additionally, the list contains for each microarray slide its array geometry.

`-dataroot`

> Absolute path to the data.

The script splits each line of the input list and puts the content into separate arrays. Each element of an array is separated by a single space. These tagged arrays are written to standard out:

1. `<scanalyze> ... </scanalyze>`
2. `<dapple> ... </dapple>`
3. `<smd> ... </smd>`
4. `<genekey> ... </genekey>`
5. `<slidecol> ... </slidecol>`
6. `<sliderow> ... </sliderow>`
7. `<gridcol> ... </gridcol>`
8. `<gridrow> ... </gridrow>`

A.5.2. Transform.pl

The script *TRANSFORM.pl* transforms the ScanAlyze file format as above described. It takes the following arguments:

`<ScanAlyzeFile>`

> ScanAlyze file to be transformed.

`<TransformedOutput>`

Transformed file in dapple format.

`-arrayconfig <numslidecols> <numsliderows> <numgridcols> <numgridrows>`

Describes the array geometry.

`-dataroot`

Absolute path to the data.

The name of the transformed file is written as a tagged string to standard out:

1. `<dapple_file> ... </dapple_file>`

A.5.3. Extract.pl

The script *TRANSFORM.pl* extracts the gene information of a SMD file. It takes the following arguments:

`<SMDFile>`

SMD file from which the gene information being extracted.

`<TransformedOutput>`

Gene information file.

`-arrayconfig <numslidecols> <numsliderows> <numgridcols> <numgridrows>`

Describes the array geometry.

`-uid <unique id from bioopera>`

The involved scripts writes temporary information to a file. When several *EXTRACT.pl* scripts are executed on the same machine in parallel, each of them needs to write to its own file. Therefore, the script needs a unique identification to name its temporary file.

`-dataroot`

Absolute path to the data.

The name of the gene information file is written as a tagged string to standard out:

1. `<genekey_file> ... </genekey_file>`

A.6. Scripts for the Expression Profiling Process

A.6.1. Partition.pl

The *PARTITION.pl* script partitions the process' input data set into smaller subsets as defined by the user. The details about the partition scheme are shown in Figure 4.9. The script takes the following arguments:

`<DappleFileList>`

List holding the filenames of the dapple files.

`<GeneKeyFileList>`

List holding the file names of the genekey files.

`<MergeRepsOutputNames>`

List holding the output names of the merged replicated conditions.

`-partitionsize`

Size of a single partition.

`-dataroot`

Absolute path to the data.

As output, the script produces for each input category the partitioned lists. The total number of the output files depends on the used partition size and the number of microarray slides. For each category, the names of the produced partitions are collected into a tagged array, separated by a single space. These arrays are written to standard out:

1. `<part_dapple_list> ... </part_dapple_list>`
2. `<part_genekey_list> ... </part_genekey_list>`
3. `<part_mergereps_list> ... </part_mergereps_list>`

A.6.2. Preproc_lookup_mergereps.pl

The *PREPROC_LOOKUP_MERGEREPS.pl* script is a wrapper for *PREPROCESS.PL*, *LOOKUP.pl* and *MERGEREPS.pl*. The script takes the following arguments:

`<DappleFileList>`

List holding the filenames of the dapple files.

`<GeneKeyFileList>`

List holding the file names of the genekey files.

`<MergeRepsOutputNamesList>`

List holding the output names of the merged replications.

`-dataroot`

Absolute path to the data.

`-scriptroot`

Absolute path to the three invoked scripts.

`-uid`

The last script invoked by the wrapper, *MERGEREPS.pl* writes the maximum number of replicated spots to standard out. This number is required as input argument in the next step. Therefore, the wrapper redirects the output of *MERGEREPS.pl* to a log file. When several *PREPROC_LOOKUP_MERGEREPS.pl* scripts are executed on the same machine in parallel, each of them needs to write to its own file. Therefore, each script needs a unique identification to name its log file.

`-sat`

Specify the saturating intensity for the microarray scanner. Spot intensities above this number are flagged, i.e. the specific spot is excluded from the analysis.

`-minreps`

Only those genes that are represented by at least the given number of replicate measurements in the merged data set are returned.

`-labeldir`

Channel 1 usually represents the control condition and channel two the treated one. With this parameter one can define this order. In the former case, the parameter should hold the value 'r'. If channel 1 represents the treated condition and channel 2 the control one, then one should set the value 'f'. The correct channel assignment is very important, as the ratio of treated condition/control condition is taken.

The names of the merged replicated conditions and their corresponding maximum number of replicated spots are collected into separate arrays. These tagged arrays are written to standard out:

1. `<merged_files> ... </merged_files>`
2. `<max_replications_list> ... </max_replications_list>`

A.6.3. Prepare_vera_sam.pl

The *PREPARE_VERA_SAM.pl* scripts transforms the output file format of *MERGEREPS.pl* in order to invoke *VERA* and *SAM*. An example output file produced by *MERGEREPS.pl* is shown in table A.8.

#suid	accession number	preferred name	N	RATIO	STD	X0	Y0	F0	X1	Y1	F1
105083	AA406115	SCYB13	2	-1.1539	0.0000	172	2454	-	845	4534	-
179038	AA836440	Unknown	2	-1.0276	0.0000	55	588	-	132	908	-

Table A.8.: Example of a output file produced by *MERGEREPS.pl*. Both spots (105083 and 179038) are replicated.

1. **suid**

   - Unique identification number of a single spot.

2. **accession number**

   - Unique identification number of the corresponding gene.

3. **preferred name**

   - Name of the corresponding gene.

4. **N**

   - Number of spot replicas.

5. **RATIO**

   - Gene expression ratio.

6. **STD**

   - Standard deviation (not used).

7. **X0**

   - Intensity of the first spot of channel 1.

8. **Y0**

   - Intensity of the first spot of channel 2.

9. **F0**

   - Flag value of the first spot.

10. **X1**

    - Intensity of the second (replicated) spot of channel 1.

11. **Y1**

    - Intensity of the second spot of channel 2.

12. **F1**

    - Flag value of the second spot.

Table A.9 shows the transformed file produced by *PREPARE_VERA_SAM.pl*. The first and the third column are renamed to *unique* and *other*. *Accession number*, *RATIO* and *STD* are omitted. The remaining columns are only reordered.

unique	other	X0	Y0	X1	Y1	N	F0	F1
105083	SCYB13	172	2454	845	4534	2	-	-
179038	Unknown	55	588	132	908	2	-	-

Table A.9.: Example of a output file produced by *PREPARE_VERA_SAM.pl*

The *PREPARE_VERA_SAM.pl* script takes the following arguments:

`<MergeRepFile>`

File produced by *MERGEREPS.pl*.

`-dataroot`

Absolute path to the data.

`-replications`

Maximum number of replicated spots.

The name of the transformed file is written as a tagged string to standard out. Additionally, the script produces the output name for the output file in the next step based on the script's input file name:

1. `<prep_file> ... </prep_file>`
2. `<error_model_filename> ... </error_model_filename>`

A.6.4. Start_vera.pl

The *START_VERA.pl* script is a wrapper in order to start *VERA*. It takes the following arguments:

`<prepFile>`

Transformed file produced by *PREPARE_VERA_SAM.pl*.

`<errorModelFileName>`

Output file name.

`-dataroot`

Absolute path to the data.

`-scriptroot`

Absolute path to the *VERA* script.

The name of the computed error model is written as a tagged string to the standard out. Additionally, the script produces the output file name for the next step based on the script's input file name:

1. `<error_model> ... </error_model>`
2. `<sam_output_filename> ... </sam_output_filename>`

A.6.5. Start_sam.pl

The *START_SAM.pl* script is a wrapper in order to start *SAM*. It takes the following arguments:

`<prepFile>`

Transformed file produced by *PREPARE_VERA_SAM.pl*.

`<errorModelFile>`

Error model for the corresponding `prepFile`.

`<samOutputFileName>`

Name of the output file.

`-dataroot`

Absolute path to the data.

`-scriptroot`

Absolute path to the *SAM* script.

The name of the output file is written as a tagged string to the standard out:

1. `<sam_file> ... </sam_file>`

A.6.6. Mergeconds.pl

The *MERGECONDS.pl* merges the experimental conditions and produces a global expression matrix. Table A.10 shows the file format of the produced matrix. The script takes the following arguments:

UID	Name	GWEIGHT	Condition A	Condition B
EWEIGHT			1	1
105083	SCYB13	1	0.103	0.462
179038	Unknown	1	-0.683	-0.527

Table A.10.: Example of a global expression matrix

The first line contains the column headers. *GWEIGHT* indicates the weight of the corresponding gene during the cluster analysis. The second line (*EWEIGHT*) indicates the weight of each particular condition during the cluster analysis. The subsequent lines list the unique identification number and the name of a particular gene, the value of the gene weight and the expression ratios of each condition.
The script takes the following arguments:

`<GeneExpressionMatrix>`

Output file name.

`-conds <MergeFile 1> [<MergeFile 2> ...]`

The names of the files to be merged into the global expression matrix.

`-dataroot`

Absolute path to the data.

`-lam`

Threshold lambda value, which have to be fulfill by each gene being listed in the expression matrix.

`-rat`

Threshold ratio value, which have to be fulfill by each gene being listed in the expression matrix.

`-std`

Threshold standard deviation value, which have to be fulfill by each gene being listed in the expression matrix.

`-n`

Include only those genes, which are represented by at least n samples per condition.

The name of the output file is written as a tagged string to the standard out:

1. `<merged_file> ... </merged_file>`

A.6.7. Start_cluster.pl

The script *START_CLUSTER.pl* performs the cluster analysis on the global expression matrix. It takes the following arguments:

`<GeneMatrixFile>`

Global expression matrix.

`-dataroot`

Absolute path to the data.

`-scriptroot`

Absolute path to *XCluster*.

`-geneclust <0|1|2>`

Indicates whether to cluster genes or not. 0 means no gene clustering, 1 means non-centered metric when clustering genes and 2 means centered metric when clustering genes.

`-expclust <0|1|2>`

Indicates whether to cluster experiments or not. See above for the possible values. One has to perform at least a kind of clustering, otherwise *START_CLUSTER.pl* produces no output.

`-distmetrics <0|1>`

Indicates whether to use pearson correlation (1) or Euclidean distance (0).

`-som <0|1>`

Whether to make a self-organizing map (SOM) (1) or not (0).

`-xdim <x dimension of som>`

Specify x dimension of SOM.

`-ydim <y dimension of som>`

Specify y dimension of SOM

`-random <0|1>`

Whether to seed the random number generator with the time when making a SOM (1) or not (0).

`-knum <number of k-means clusters>`

How many k-means clusters to make.

`-log <0|1>`

Indicates whether to log transform (1) or not (0).

`-out <name of outfile>`

Unique identifier by which to name the output files.

The script produces three output files. Each name of them is written as a tagged string to standard out:

1. `<clustered_data_table_file> ... </clustered_data_table_file>`
2. `<gene_tree_file> ... </gene_tree_file>`
3. `<array_tree_file> ... </array_tree_file>`

Bibliography

[1] National Center for Biotechnology Information (NCBI). *dbEST: summary by organism.* http://www.ncbi.nlm.nih.gov/dbEST/dbEST_summary.html.

[2] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, and et al. The sequence of the human genome. *Science*, 291(5507):1304–51., 2001.

[3] E. S. Lander, L. M. Linton, B. Birren, C. Nusbaum, and et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921., 2001.

[4] Affymetrix, Inc. *Microarray Suite Software.* http://www.affymetrix.com/products/software /specific/mas.affx.

[5] GeneData AG. *GeneData Expressionist.* http://www.genedata.com/products/expressionist/.

[6] Stanford University. *Stanford Microarray Database.* http://genome-www5.stanford.edu/MicroArray/SMD/.

[7] EMBL-EBI European Bioinformatics Institute. *ArrayExpress at the EBI.* http://www.ebi.ac.uk/microarray/ArrayExpress/arrayexpress.html.

[8] V.M. Markowitz, I.A. Chen, and A. Kosky. Gene expression data management: A case study. In *EDBT 2002*, pages 722–731, 2002.

[9] Microarray Gene Expression Data Group - MGED Group. *MGED Home.* http://www.mged.org/.

[10] G. Sherlock. Analysis of large-scale gene expression data. *Briefings in Bioinformatics*, 2(4):350–62., 2001.

[11] J. C. Alwine, D. J. Kemp, and G. R. Stark. Method for detection of specific rnas in agarose gels by transfer to diazobenzyloxymethyl-paper and hybridization with dna probes. *Proceedings of the National Academy of Sciences of the United States of America*, 74(12):5350–4., 1977.

[12] A. J. Berk and P. A. Sharp. Sizing and mapping of early adenovirus mrnas by gel electrophoresis of s1 endonuclease-digested hybrids. *Cell*, 12(3):721–32., 1977.

[13] P. Liang and A. B. Pardee. Differential display of eukaryotic messenger rna by means of the polymerase chain reaction. *Science*, 257(5072):967–71., 1992.

[14] M. D. Adams, J. M. Kelley, J. D. Gocayne, M. Dubnick, M. H. Polymeropoulos, H. Xiao, C. R. Merril, A. Wu, B. Olde, R. F. Moreno, and et al. Complementary dna sequencing: expressed sequence tags and human genome project. *Science*, 252(5013):1651–6., 1991.

[15] K. Okubo, N. Hori, R. Matoba, T. Niiyama, A. Fukushima, Y. Kojima, and K. Matsubara. Large scale cdna sequencing for analysis of quantitative and qualitative aspects of gene expression. *Nature Genetics*, 2(3):173–9., 1992.

[16] V. E. Velculescu, L. Zhang, B. Vogelstein, and K. W. Kinzler. Serial analysis of gene expression. *Science*, 270(5235):484–7, 1995.

[17] M. Schena, D. Shalon, R.W. Davis, and P.O. Brown. Quantitative monitoring of gene expression patterns with a complementary dna microarray. *Science*, 270(5235):467–70, 1995.

[18] M. Schena, D. Shalon, R. Heller, A. Chai, P.O. Brown, and R.W. Davis. Parallel human genome analysis: microarray-based expression monitoring of 1000 genes. *Proceedings of the National Academy of Sciences of the United States of America*, 93(20):10614–9, 1996.

[19] D.J. Lockhart, H. Dong, M.C. Byrne, M.T. Follettie, M.V. Gallo, M.S. Chee, M. Mittmann, C. Wang, M. Kobayashi, H. Horton, and E.L. Brown. Expression monitoring by hybridization to high-density oligonucleotide arrays. *Nature Biotechnology*, 14(13):1675–80, 1996.

[20] Affymetrix, Inc. http://www.affymetrix.com.

[21] The Brown Lab. *The Brown Lab's complete guide to microarraying for the molecular biologist.* http://cmgm.stanford.edu/pbrown/mguide/index.html.

[22] Affymetrix, Inc. *Technology.* http://www.affymetrix.com/technology/index.affx.

[23] National Center for Biotechnology Information (NCBI). *UniGene database.* http://www.ncbi.nlm.nih.gov/UniGene/.

[24] National Center for Biotechnology Information (NCBI). *GeneBank database.* http://www.ncbi.nlm.nih.gov/Genbank/index.html.

[25] Affymetrix, Inc. *Human Genom U133 Set product specification.* http://www.affymetrix.com/products/arrays/specific/hgu133.affx.

[26] C. Kooperberg, T. G. Fazzio, J. J. Delrow, and T. Tsukiyama. Improved background correction for spotted dna microarrays. *Journal of Computational Biology*, 9(1):55–66., 2002.

[27] J. Quackenbush. Computational analysis of microarray data. *Nature Reviews Genetics*, 2(6):418–27., 2001.

[28] A. Zien, T. Aigner, R. Zimmer, and T. Lengauer. Centralization: a new method for the normalization of gene expression data. *Bioinformatics*, 17(Suppl 1):S323–31., 2001.

[29] Y. Wang, J. Lu, R. Lee, Z. Gu, and R. Clarke. Iterative normalization of cdna microarray data. *IEEE Trans Inf Technol Biomed*, 6(1), 2002.

[30] J. Schuchhardt, D. Beule, A. Malik, E. Wolski, H. Eickhoff, H. Lehrach, and H. Herzel. Normalization strategies for cdna microarrays. *Nucleic Acids Research*, 28(10):E47., 2000.

[31] Y. H. Yang, S. Dudoit, P. Luu, D. M. Lin, V. Peng, J. Ngai, and T. P. Speed. Normalization for cdna microarray data: a robust composite method addressing single and multiple slide systematic variation. *Nucleic Acids Research*, 30(4):e15., 2002.

[32] T. Ideker, V. Thorsson, A. F. Siegel, and L. E. Hood. Testing for differentially-expressed genes by maximum-likelihood analysis of microarray data. *Journal of Computational Biology*, 7(6):805–17., 2000.

[33] M.B. Eisen, P.T. Spellman, P.O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences of the United States of America*, 95(25):14863–8, 1998.

[34] S. Tavazoie, J. D. Hughes, M. J. Campbell, R. J. Cho, and G. M. Church. Systematic determination of genetic network architecture. *Nature Genetics*, 22(3):281–5., 1999.

[35] T. Kohonen. Self organizing maps. *Springer, Berlin*, 1995.

[36] M. P. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. W. Sugnet, T. S. Furey, Jr. Ares, M., and D. Haussler. Knowledge-based analysis of microarray gene expression data by using support vector machines. *Proceedings of the National Academy of Sciences of the United States of America*, 97(1):262–7., 2000.

[37] J. S. Almeida. Predictive non-linear modeling of complex data by artificial neural networks. *Current Opinion in Biotechnology*, 13(1):72–6., 2002.

[38] J. Khan, J. S. Wei, M. Ringner, L. H. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C. R. Antonescu, C. Peterson, and P. S. Meltzer. Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nature Medicine*, 7(6):673–9., 2001.

[39] Institute for Systems Biology. *DNA Microarray Data Processing.* http://www.systemsbiology.org/ArrayProcess/.

[40] G. Sherlock, T. Hernandez_Boussard, A. Kasarskis, G. Binkley, J.C. Matese, S.S. Dwight, M. Kaloper, S. Weng, H. Jin, C.A. Ball, M.B. Eisen, P.T. Spellman, P.O. Brown, D. Botstein, and J.M. Cherry. The stanford microarray database. *Nucleic Acids Research*, 29(1):152–5, 2001.

[41] A.A. Alizadeh, M.B. Eisen, R.E. Davis, C. Ma, I.S. Lossos, A. Rosenwald, J.C. Boldrick, H. Sabet, T. Tran, X. Yu, J.I. Powell, L. Yang, G.E. Marti, T. Moore, J. Hudson, L. Lu, D.B. Lewis, R. Tibshirani, G. Sherlock, W.C. Chan, T.C. Greiner, D.D. Weisenburger, J.O. Armitage, R. Warnke, and L.M. Staudt. Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. *Nature*, 403(6769):503–11, 2000.

[42] G. Alonso, W. Bausch, C. Pautasso, M. Hallett, and A. Kahn. Dependable Computing in Virtual Laboratories. In *Proc. of the 17th International Conference on Data Engineering (ICDE2001)*, Heidelberg, Germany, 2001.

[43] Washington University in St. Louis. *Dapple: Image Analysis Software for DNA Microarrays.* http://www.cs.wustl.edu/ jbuhler/research/dapple/.

[44] Eisen Lab. *ScanAlyze.* http://rana.lbl.gov/EisenSoftware.htm.

[45] Eisen Lab. *TreeView.* http://rana.lbl.gov/EisenSoftware.htm.

Glossary

Adenine short A; a nitrogenous base, one member of the base pair A-T (adenine-thymine).

Base pairing rules for DNA: A-T, G-C; for RNA: A-U, G-C

Base Any basic (alkaline) compound containing nitrogen, but generally referring to one of four complex molecules (nucleotides) that form the building blocks of the nucleic acids, DNA and RNA.

cDNA Library A collection of all of the mRNA molecules present in a cell or organism, all turned into cDNA molecules with the enzyme reverse transcriptase, then inserted into vectors (other DNA molecules which can continue to replicate after addition of foreign DNA). The library can then be probed for the specific cDNA (and thus mRNA) of interest.

cDNA complementary DNA. DNA synthesized from an RNA template using reverse transcriptase. Because cDNA is synthesized of mRNA, it only contains the information needed for protein production. Compare exon.

Chromosome The self-replicating genetic structures of cells containing the cellular DNA that bears in its nucleotide sequence the linear array of genes. In prokaryotes, chromosomal DNA is circular, and the entire genome is carried on one chromosome. Eukaryotic genomes consist of a number of chromosomes whose DNA is associated with different kinds of proteins.

Cloning The process of asexually producing a group of cells (clones), all genetically identical, from a single ancestor. In recombinant DNA technology, the use of DNA manipulation procedures to produce multiple copies of a single gene or segment of DNA is referred to as cloning DNA.

Complementary Base Pairing The pairing of complementary nucleotide bases (adenine and thymine, guanine and cytosine) to each other via hydrogen bonds from opposite strands of a double stranded nucleic acid (such as DNA or RNA), thereby holding the double-stranded nucleic acid together.

Cye3-dUTP Fluorescently labeled desoxy uridine triphosphate, which the mRNA representing the experimental samples is prepared for hybridization with DNA microarrays by reverse transcription. Cye3 is the fluorescent dye, which emits green light by excitation with light of wavelength 550nm. Usually used for the control samples.

Cye5-dUTP Fluorescently labeled desoxy uridine triphosphate, which the mRNA representing the experimental samples is prepared for hybridization with DNA microarrays by reverse transcription. Cye5 is the fluorescent dye, which emits red light by excitation with light of wavelength 650nm. Usually used for the control samples.

Cytosine short C; a nitrogenous base, one member of the base pair G-C (guanine and cytosine).

Domain A discrete portion of a protein with its own function. The combination of domains in a single protein determines its overall function.

EST A short (200 to 500 base pairs) DNA sequence derived from cDNA that has a single occurrence in the human genome and whose location and base sequence are known. EST are putative genes which still have to be described.

Exon The protein-coding DNA sequences of a gene. Compare intron.

Gene Expression The process by which a gene's coded information is converted into the structures present and operating in the cell. Expressed genes include those that are transcribed into mRNA and then translated into protein and those that are transcribed into RNA but not translated into protein (e.g., transfer and ribosomal RNAs).

Gene Mapping Determination of the relative positions of genes on a DNA molecule (chromosome or plasmid) and of the distance, in linkage units or physical units, between them.

Gene The fundamental physical and functional unit of heredity. A gene is an ordered sequence of nucleotides located in a particular position on a particular chromosome that encodes a specific functional product (i.e., a protein or RNA molecule).

Genome All the genetic material in the chromosomes of a particular organism; its size is generally given as its total number of base pairs.

Genomics The study of genomes, which includes genome mapping, gene sequencing and gene function.

Genotype The genetic constitution of an organism. Compare phenotype.

Guanine short G; a nitrogenous base, one member of the base pair G-C (guanine and cytosine).

Hybridization The process of joining two complementary strands of DNA or one each of DNA and RNA to form a double- stranded molecule.

Intron The DNA base sequences interrupting the protein- coding sequences of a gene; these sequences are transcribed into RNA but are cut out of the message before it is translated into protein. Compare exons.

mRNA RNA that serves as a template for protein synthesis.

Nucleotide A subunit of DNA or RNA consisting of a nitrogenous base (adenine, guanine, thymine, or cytosine in DNA; adenine, guanine, uracil, or cytosine in RNA), a phosphate molecule, and a sugar molecule (deoxyribose in DNA and ribose in RNA). Thousands of nucleotides are linked to form a DNA or RNA molecule.

Oligonucleotide A compound comprising a nucleotide linked to phosphoric acid. When polymerized, it gives rise to a nucleic acid.

Phenotype The physical appearance/observable characteristics of an organism. Compare genotype.

Plasmid Autonomously replicating, extrachromosomal circular DNA molecules, distinct from the normal bacterial genome and nonessential for cell survival under nonselective conditions. Some plasmids are capable of integrating into the host genome. A number of artificially constructed plasmids are used as cloning vectors.

Reverse Transcriptase An enzyme found in retroviruses that enable the virus to make DNA from viral RNA.

Ribosomes Small cellular components composed of specialized ribosomal RNA and protein; site of protein synthesis.

RNA A chemical found in the nucleus and cytoplasm of cells; it plays an important role in protein synthesis and other chemical activities of the cell. The structure of RNA is similar to that of DNA. There are several classes of RNA molecules, including messenger RNA, transfer RNA, ribosomal RNA, and other small RNAs, each serving a different purpose.

rRNA A class of RNA found in the ribosomes of cells providing them with structural and functional properties.

Sequencing Any lab technique used to find out the sequence of nucleotide bases in a DNA molecule or fragment.

Thymine short T; a nitrogenous base, one member of the base pair A-T (adenine-thymine).

Transcription The synthesis of an RNA copy from a sequence of DNA (a gene); the first step in gene expression. Compare translation.

Translation The process in which the genetic code carried by messenger RNA directs the synthesis of proteins from amino acids. Compare transcription.

tRNA transfer RNA. A class of RNA having structures with triplet nucleotide sequences that are complementary to the triplet nucleotide coding sequences of mRNA. The role of tRNAs in protein synthesis is to bond with amino acids and transfer them to the ribosomes, where proteins are assembled according to the genetic code carried by mRNA.

Uracil short U; a nitrogenous base normally found in RNA but not DNA; uracil is capable of forming a base pair with adenine.