# Parallel Computing for Biological Data

Markus Schmidberger

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

München, den 11. August 2009

# Parallel Computing for Biological Data

**Markus Schmidberger**

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Markus Schmidberger
aus Weilheim

München, den 11. August 2009

Erstgutachter: Prof. Dr. Ulrich Mansmann
Zweitgutachter: Prof. Dr. Friedrich Leisch
Tag der mündlichen Prüfung: 18. November 2009

# Contents

# List of Figures

# List of Tables

# Abstract - English

In the 1990s a number of technological innovations appeared that revolutionized biology, and 'Bioinformatics' became a new scientific discipline. Microarrays can measure the abundance of tens of thousands of mRNA species, data on the complete genomic sequences of many different organisms are available, and other technologies make it possible to study various processes at the molecular level. In Bioinformatics and Biostatistics, current research and computations are limited by the available computer hardware. However, this problem can be solved using high-performance computing resources. There are several reasons for the increased focus on high-performance computing: larger data sets, increased computational requirements stemming from more sophisticated methodologies, and latest developments in computer chip production.

The open-source programming language 'R' was developed to provide a powerful and extensible environment for statistical and graphical techniques. There are many good reasons for preferring R to other software or programming languages for scientific computations (in statistics and biology). However, the development of the R language was not aimed at providing a software for parallel or high-performance computing. Nonetheless, during the last decade, a great deal of research has been conducted on using parallel computing techniques with R.

This PhD thesis demonstrates the usefulness of the R language and parallel computing for biological research. It introduces parallel computing with R, and reviews and evaluates existing techniques and R packages for parallel computing on Computer Clusters, on Multi-Core Systems, and in Grid Computing. From a computer-scientific point of view the packages were examined as to their reusability in biological applications, and some upgrades were proposed.

Furthermore, parallel applications for next-generation sequence data and preprocessing of microarray data were developed. Microarray data are characterized by high levels of noise and bias. As these perturbations have to be removed, preprocessing of raw data has been a research topic of high priority over the past few years. A new Bioconductor package called **affyPara** for parallelized preprocessing of high-density oligonucleotide microarray data was developed and published. The partition of data can be performed on arrays using a block cyclic partition, and, as a result, parallelization of algorithms becomes directly possible. Existing statistical algorithms and data structures had to be adjusted and reformulated for the use in parallel computing. Using the new parallel infrastructure, normalization methods can be enhanced and new methods became available. The partition of data and distribution to several nodes or processors solves the main memory problem and accelerates the methods by up to the factor fifteen for 300 arrays or more.

The final part of the thesis contains a huge cancer study analysing more than 7000 microarrays from a publicly available database, and estimating gene interaction networks. For this purpose, a new R package for microarray data management was developed, and various challenges regarding the analysis of this amount of data are discussed. The comparison of gene networks for different pathways and different cancer entities in the new amount of data partly confirms already established forms of gene interaction.

# Abstract - Deutsch

In den 1990er Jahren sind einige technische Neuerungen erschienen, die die biologische Forschung revolutioniert haben, und das neue Forschungsgebiet 'Bioinformatik' wurde geboren. Microarrays messen mehrere 10.000 mRNA Arten, die vollständigen genomischen Sequenzen vieler verschiedener Lebewesen sind verfügbar, und andere Technologien ermöglichen die Erforschung von molekular-biologischen Prozessen. Die heute verfügbare Computerhardware limitiert die aktuelle Forschung und Simulationen in der Bioinformatik und Biostatistik. Dieses Problem kann aber durch den Einsatz von Hochleistungs-Rechenarchitekturen gelöst werden. Als Gründe für das gewachsene Interesse an neuen Rechenarchitekturen können folgende Punkte genannt werden: große Datensätze, anspruchsvollere und rechenintensivere Methoden und neue Entwicklungen in der Computerchip-Herstellung.

Die freie Programmiersprache und Statistiksoftware 'R' wurde als umfangreiche und erweiterbare Programmierumgebung für statistische Methoden und grafische Auswertungen entwickelt. Es gibt viele gute Gründe dafür, R gegenüber anderer Software oder anderen Programmiersprachen für wissenschaftliches Rechnen (in der Statistik und Biologie) zu bevorzugen. R wurde nicht für den Einsatz auf Hochleistungs-Rechenarchitekturen entwickelt, jedoch wurde der Einsatz von Hochleistungrechnern mit R in den letzten Jahren verstärkt vorangetrieben.

Diese Doktorarbeit demonstriert die Brauchbarkeit der R Software sowie des parallelen Rechnens für biologische Forschungszwecke. Die Arbeit gibt eine Einführung in paralleles Rechnen mit R und vergleicht und bewertet existierende Technologien und R Pakete für paralleles Rechnen auf Computer Clustern, auf Multi-Core Rechnern und in Grid Netzwerken. Die Pakete wurden aus der Sicht des Informatikers für den Einsatz in biologischen Anwendungen untersucht und einige Verbesserungen wurden vorgeschlagen.

Des weiteren wurden parallele Algorithmen für die neuen Sequenzierungstechniken – oft 'Next-Generation Sequencing' genannt – und für die Vorverarbeitung von Microarrays – oft 'Preprocessing' genannt – entwickelt. Von Microarray-Chips gemessene Daten sind für ihr starkes Rauschen und ihre Verzerrung bekannt. Da diese Fehler beseitigt werden müssen, wurden in den letzten Jahren viel auf diesem Gebiet geforscht. In dieser Arbeit wurde das Bioconductor Paket **affyPara** mit parallelisierten Preprocessingverfahren für high-density Oligonucleotide Microarrays entwickelt und veröffentlicht. Mit einer block-cyclic Aufteilung der Arrays können die Daten verteilt werden und die Parallelisierung der meisten existierenden Algorithmen ist unmittelbar möglich. Existierende statistische Verfahren und Datenstrukturen wurden angepasst und für paralleles Rechnen umformuliert. Durch die neue parallele Infrastruktur werden Normalisierungsverfahren verbessert und Ideen für neue Methoden entstanden. Die Aufteilung der Daten und Verteilung auf mehrere Rechner oder Prozessoren löst das Arbeitsspeicherproblem und beschleunigt die Verfahren bis zum 15-fachen für 300 oder mehr Arrays.

Die Arbeit endet mit einer großen Studie zu Krebsdaten, bestehend aus mehr als 7000 Microarrays von einer öffentlich zugänglichen Datenbank. Die Daten wurden alle zusammen analysiert und Gen-Interaktions-Netzwerke geschätzt. Dafür wurde ein neues R Paket für das Datenmanagement von Micorarray Experimenten entwickelt. Herausforderungen, die bei der Analyse dieser Datenmenge entstanden sind, wurden diskutiert. Durch den Vergleich der Gennetzwerke für verschiedene Pathways und Krebsentitäten ist eine teilweise Bestätigung der bekannten Genezusammenhänge möglich.

# Chapter 1

# Introduction

**Bioinformatics and Computational Biology:** *Bioinformatics* is an interdisciplinary research area at the interface between computer science and biological science. In literature a number of different detailed definitions of bioinformatics exist. Following [LGG01] 'bioinformatics is conceptualising biology in terms of molecules and applying informatics techniques (derived from disciplines such as applied maths, computer science and statistics) to understand and organise the information associated with these molecules, on a large scale. [. . . ]'. Therefore, it deals with techniques involving computers for storage, retrieval, manipulation, and distribution of information related to biological macromolecules such as DNA, RNA, and proteins. The emphasis here is on the use of computers, because most of the tasks in genomic data analysis are highly repetitive or mathematically complex. Bioinformatics deals with macromolecules whereas *computational biology* is an interdisciplinary field that applies the techniques of computer science, applied mathematics and statistics to address all biological problems. It encompasses the fields of bioinformatics, computational biomodeling, systems biology, protein structure prediction and structural genomics, . . . . Further, the field of bioinformatics is already looking forward to what is currently termed a 'systems biology' approach and to simulations of whole cells with incorporation of more levels of complexity.

The primary target of bioinformatics is to increase our understanding of biological processes. Two major research areas are the analysis of gene expression using microarrays and sequence analysis.

**New Challenges in Bioinformatics:** There are three primary reasons for the increased focus on high-performance computing in bioinformatics: larger data sets, increased computational requirements stemming from more sophisticated methodologies, and latest developments in computer chip production.

Bioinformatics clearly illustrates the 'growing data' problem, considering that just a few years ago, experimentally-generated data sets often fit on a single CD-ROM. Today, data sets from high-throughput data sources as for examples next-generation DNA sequencing no longer fit on a single DVD-ROM. In genome research, data sets appear to be growing at a rate that is faster than the corresponding increase in hardware performance. At the same time, methodological advances have led to computationally more demanding solutions. A common approach among these recent methods is the use of simulation and resampling techniques. Markov Chain Monte Carlo and Gibbs Sampling, Bootstrapping, and Monte Carlo Simulation are examples of increasingly popular methods that are important in various areas of statistical analysis investigating geographic, econometric, and biological data. For example in microarray data, building classification or prognostic rules from large microarray sets will be very time consuming. Here, preprocessing must become part of the cross-validation and resampling strategy, which is necessary to estimate the rule's prediction quality correctly. Both increased data size and increased simulation demands have been approached by researchers by means of parallel computing [GH04].

Data sets can often be subdivided into 'chunks' that can undergo analysis in parallel. This is particularly the case when the analysis of a given data 'chunk' does not depend on other chunks. Similarly, simulation runs that are not dependent on other simulations can be conducted in parallel. Hence, both reasons – data size and simulation demands – for an increased interest in high-performance and parallel computing can be seen as so-called *embarrassingly parallel* problems that are suitable for execution in parallel.

**High-Performance Computing:** *High-Performance Computing* (HPC) is one aspect of computer based computing and includes tasks which require a huge amount of computing power and memory. In most cases it involves the use of super computers, computer clusters, grid environments, but even graphical processing units to solve advanced computing problems. *Parallel computing* is a form of computation in which many calculations are carried out simultaneously and which can run on different hardware environments. Parallel computing is used to save computation time and money, to solve large problems, to use non-local resources, etc.. In the last years more and more research has focused on using HPC and parallel computing techniques for bioinformatic applications to solve these new challenges.

**R and Bioconductor:** *R* is an open-source programming language and software environment for statistical computing and graphics which provides many statistical and modelling functions and is highly extensible due to the use of packages. *Bioconductor* is an open-source project and R package repository for the analysis and comprehension of genomic data. The development of the R language was not aimed at providing a software for parallel or high-performance computing. Nonetheless, several packages for the use of parallel computing techniques exist. Especially the **snow** package is useful for the use of computer clusters and on multi-processor machines.

**Outline:** This work reviews parallel computing techniques in the R language for the application in biological - especially genomic - data and demonstrates the usefulness of the R language and parallel computing for biological research. Existing statistical algorithms and data structures had to be adjusted to and reformulated for parallel computing. Using the parallel infrastructure, it is possible to improve existing methods and to develop new more efficient methods. New R packages for parallel preprocessing of microarray data and data management of a huge amount of microarray data are presented. The use of parallel computing in next-generation sequence data is discussed.

Chapters 1 to 3 introduce the combined research areas and existing methodology. Chapter 2 describes basic biological concepts and the standard analysis processes for various biological data. Chapter 3 introduces the R language and the Bioconductor repository, describes available serial analysis tools, problems, challenges, and solutions occurring as a result of new challenges.

The main part (Chapters 4 to 7) develops and applies new concepts (parallel computing) regarding genomic data. At the end of each chapter, the results are summarized. Chapter 4 in detail deals with parallel computing and packages for the use of computer clusters and multi-processor environments. The idea, development, implementation, and efficiency of the **affyPara** package is discussed in Chapter 5. This is a package with a parallel code for the preprocessing of huge amounts of microarray data. Chapter 6 reviews first ideas and test implementations for next-generation sequence data. Especially problems arising from the large amount of network traffic have to be solved. In Chapter 7 the newly developed **affyPara** package is applied to a cancer data set with more than 7000 microarrays. For the purpose of solving data management problems, a new package called **ArrayExpressDataManage** was developed and applied. Several problems arising from the huge amount of data and the use of data from different experiments are discussed. Using network analysis tools, first biological interpretations and validations are presented.

The thesis ends with a summary and critical discussion of using R in parallel computing for biological data, and new trends in HPC are discussed for the use in bioinformatics.

# Chapter 2

# Biological Data

Based on the definition in the online encyclopedia Wikipedia (`http://en.wikipedia.org/wiki/Biological_measurement`, 20. March 2009) *biological data* are 'data or measurements collected from biological sources, which are stored or exchanged in a digital form. Biological data is commonly stored in files or databases [...].' This is a very general definition and covers a wide field of biological research. Some examples for biological data are DNA base-pair sequences, population data used in ecology, biosonar data, cell-based assays, microarray data, and many more. Often biological data have very similar and characteristic features: They have more variables than observations, the raw data is noisy, and they are high-dimensional.

This work focuses on biological data arising from the study of genomes, especially microarray data and data from next-generation sequence data.

The study of genomes is called *genomics*. The field of genomics encompasses two main areas, *structural genomics* and *functional genomics*. The former mainly deals with genome structures with a focus on the study of genome mapping and assembly as well as genome annotation and comparison. The latter is largely experiment based with a focus on gene function at the whole genome level using high-throughput approaches. *High-throughput* is the simultaneous analysis of a lot of genes in a genome. The high-throughput analysis of all expressed genes is also termed *transcriptome analysis*.

This chapter discusses aspects of the transcriptome analysis that can be conducted using either microarray- or sequence-based approaches. First, it gives some biological background information about the field of genomics. Section 2.2 introduces the microarray technology, especially DNA microarrays, the DNA microarray analyses process and an error model for microarray data. Section 2.3 describes aspects and analyses of next-generation sequence data. The chapter ends with a listing of further biological data and their analyses.

## 2.1 Biological Background

*Cells* contain, within each nucleus, the entire genome for their organism. This *genome* contains the organism's complete hereditary information in the form of *deoxyribonucleic* *acid* (DNA), which contains the genetic instructions used in the development and functioning of all known living organisms and some viruses. The main role of DNA molecules is the long-term storage of information. DNA is often compared to a set of blueprints or a recipe, or a code, since it contains the instructions needed to construct other components of cells, such as proteins and RNA molecules. The DNA segments that carry this genetic information are called *genes*.

In the human body, the genome consists of 23 pairs of *chromosomes*. One of the two copies is inherited from the mother and the other one from the father. Each chromosome is made of chains of DNA. DNA consists of two polymers made up of units called *nucleotides*. Each nucleotide consists of a deoxyribose sugar, a phosphate group and one of the four nitrogen bases, *guanine*, *adenine*, *thymine* and *cytosine*. These bases, which are usually represented by their first letters, G, A, T and C, actually encode the hereditary genetic information. One of the two strands of the DNA double helix will suffice to describe this information; this is due to complementary base pairing, where an A on one strand always binds to a T on the other and a C always binds to a G (see Figure 2.1).

*Genes* are essentially segments of the DNA structure described above. Loosely speaking, a gene is a section of DNA that defines a single trait by encoding a particular pattern, about 27,000 of which exist in humans. Often though we are faced with the problem that protein-coding sequences have no clear beginning or end; more technically, a gene is a locatable region of genomic sequence, corresponding to a unit of inheritance, which is associated with regulatory regions, also known as *exons*, transcribed regions, also known as *introns* and/or other functional sequence regions [Pea06].

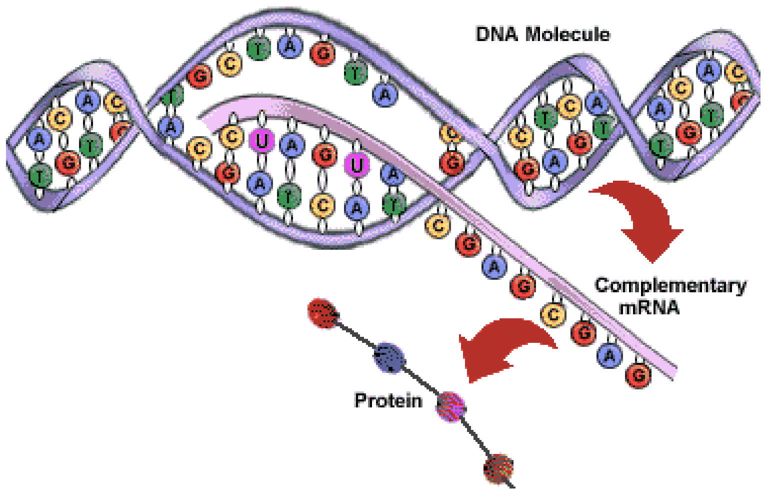


Figure 2.1: The transcription process (`http://www.scientificpsychic.com/fitness/aminoacids1.html`).

The main purpose of genes is to act as a blueprint in the creation of proteins. Proteins are made of amino acids and are responsible for the structure and activity of an organism at a cellular level. They are created as follows and as visualized in Figure 2.1: Starting at the 5' end (the leading end) of a gene and proceeding to the 3' end (the tail end), the information contained in the gene is transcribed into a *messenger ribonucleic acid* (mRNA) strand. This process is performed by an enzyme called RNA polymerase. After transcription this mRNA molecule leaves the nucleus of the cell, where it is transcribed into a protein in a process called *translation*. This is performed by ribosomes, which read the code carried by mRNA molecules from the cell nucleus and create proteins combining any of the 20 amino acids in the body into complex polypeptide chains. These proteins are the building blocks of the organism. This process of translating a gene into a functional product is known as *gene expression*.

## 2.2 Microarray Data

*Microarray* is a collective name for a modern molecular biology analysis tool. It consists of an arrayed series of thousands of microscopic spots - called *features* - and allows the parallel analysis of several thousands of genes. There are different kinds of microarrays - sometimes called *Genchips* or *Biochips*. Most prominent are the *DNA microarrays*, which contain picomoles of a specific DNA sequence at each spot. Other types of microarrays are Protein-Microarrays or Transfection-Microarrays.

### 2.2.1 DNA Microarrays

*DNA microarrays* are a high throughput technology used to measure the expression levels of thousands of genes simultaneously. The fundamental idea behind most microarrays is to exploit complementary base pairing to measure the amount of the different types of mRNA molecules in a cell, thus indirectly measuring the expression levels of the genes that are responsible for the synthesis of those particular mRNA molecules.

The spots on a microarray contain single stranded DNA oligonucleotides called *probes*. Each of these spots will contain DNA, which is of a complementary sequence to the specific mRNA molecule, that corresponds to the gene, that it is targeting. A mRNA molecule, which is complementary to the probe in question, should hybridise to that probe, forming a strong mRNA-DNA bond. These mRNA molecules have been labeled with fluorescent dye, which means that the amount of hybridisation, that has taken place, can be measured by the level of fluorescence of the dye, which is examined with a scanner. This scanner then outputs a text file for each array, which contains the relevant data pertaining to that array, such as the level of fluorescence of each spot and the level of background noise. In theory, a spot with brighter fluorescence means, that more mRNA has hybridised, which in turn infers, that more mRNA was present in the sample extracted from the original cell and that the gene represented by this spot is experiencing a higher level of expression.

DNA microarrays can be manufactured in different ways, depending on the number of probes under examination, costs, customization requirements, manufacturers, etc.. Arrays may have as few as 10 to up to 2 million probes. There are several microarray manufacturers, the most prominent ones are Affymetrix, Inc. (`http://www.affymetrix.com/`) and

Illumina, Inc. (`http://www.illumina.com/`).

### DNA Microarray Platforms

The types of DNA microarrays most widely used today can be broadly devided into two categories that are differentiated by the type of data they produce. The *high-density oligonucleotide array* (oligo) platforms produce **one** set of probe-level data per microarray with some probes designed to measure specific binding and other to measure non-specific binding. The *two-color spotted* (cDNA) platforms produce **two** sets of probe level data per microarray (red and green channel).

In the following description of microarray production only the manufacturing process for Oligonucleotide Affymetrix Microarray Chips will be described.

**High-density Oligonucleotide Array - Affymetrix GeneChip:** The Affymetrix GeneChip® array is an oligonucleotide array and is the most commonly used type of DNA microarray. Figure 2.2 shows an image of two Affymetrix GeneChips.

Figure 2.2: Two Affymetrix Microarray GeneChips® for Human (HG-U133 Plus 2.0) and Mouse Genome (Mouse-430 2.0).

GeneChips use short oligonucleotides to probe for genes in a RNA sample. Each array will contain hundreds of thousands of probe spots and each of these spots will in turn contain millions of copies of an individual 25 base long DNA oligonucleotide. Genes are represented by a set of oligo probes each with a length of 25 bases. Because of their short length, multiple probes are used to improve specificity. Affymetrix arrays typically use between 11 and 20 probe pairs, referred to as a *probe set*, for each gene. One component of these pairs is referred to as a *perfect match probe* (PM) and is designed to hybridize only with transcripts from the intended gene. However, hybridization to the PM probes by other mRNA species is unavoidable. Therefore, the observed intensities need to be adjusted to be accurately quantified. The other component of a probe pair, the *mismatch probe* (MM), is constructed with the intention of measuring only the non-specific component of the corresponding PM probe. Affymetrix's strategy is to make MM probes identical to their PM counterpart expect that the 13-th base is exchanged with its complement. The

Affymetrix chip design is illustrated in Figure 2.3, visualizing the pseudo-image and the probe set structure. For example the Affymetrix human 'HG-U133A' chip contains more

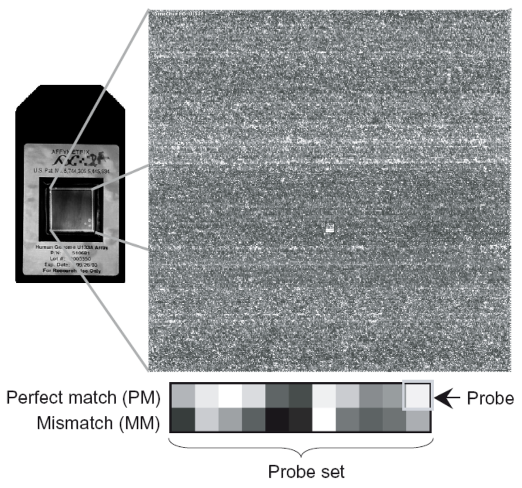


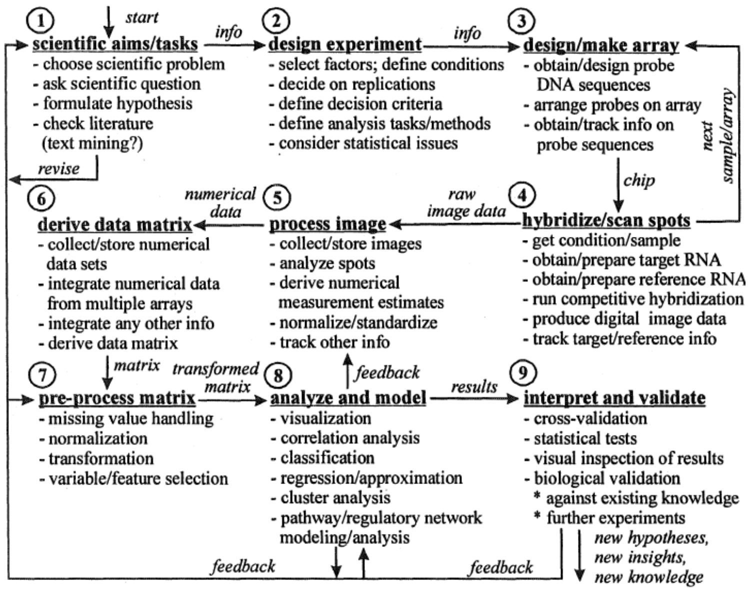
Figure 2.3: Illustration of the Affymetrix Microarray GeneChip® design with 11 probe pairs.

than 14.500 genes, more than 22.000 probe sets and has 11 probe pairs (about 500,000 probes). The feature size is 18 $\mu$m and probes from the same probe set are distributed randomly over the chip.

For more details about the GeneChip platform see the Affymetrix website `http://www.affymetrix.com`.

### 2.2.2 DNA Microarray Analysis Process

Microarray analysis is a complex process and starts long before computational statistical analysis will be performed. Details can be found in numerous books for microarray data analysis [BDG03, Ste03]. Figure 2.4 visualizes a typical microarray data analysis process as described in [BDG03]. Steps 1 and 2 are very essential for the outcome of an experiment, because scientific aims and the experiment design will be defined. Steps 3 and 4 run in the laboratory, are complex and can have a strong influence on the results. Step 5 is mainly concerned with the analysis of the digitized image arising out of Step 4. Image analysis permits to convert the pixel intensities in the scanned images into probe-level data. The result is a collection of numerical estimates representing the measured expression levels. For reasons of limited space Steps 1 to 5 will not be discussed in detail. Instead the focus lies on the preprocessing (Step 6 and 7) of the integrated data matrix, which is often called *low level analysis*, dealing with removing measurement errors and data transformation. This is discussed in the next paragraph. *High level analysis* is the 'real' statistical analyses and interpretation of the data (Step 8 and 9). As far as required for the analyses of the large cancer study in Chapter 7 differential gene expression and network based methods are



1 scientific aims/tasks
start
info
- choose scientific problem
- ask scientific question
- formulate hypothesis
- check literature (text mining?)
revise
2 design experiment
info
- select factors; define conditions
- decide on replications
- define decision criteria
- define analysis tasks/methods
- consider statistical issues
3 design/make array
- obtain/design probe DNA sequences
- arrange probes on array
- obtain/track info on probe sequences
next sample/array
chip
4 hybridize/scan spots
- get condition/sample
- obtain/prepare target RNA
- obtain/prepare reference RNA
- run competitive hybridization
- produce digital image data
- track target/reference info
raw image data
5 process image
- collect/store images
- analyze spots
- derive numerical measurement estimates
- normalize/standardize
- track other info
numerical data
6 derive data matrix
- collect/store numerical data sets
- integrate numerical data from multiple arrays
- integrate any other info
- derive data matrix
matrix
7 pre-process matrix
- missing value handling
- normalization
- transformation
- variable/feature selection
transformed matrix
feedback
8 analyze and model
- visualization
- correlation analysis
- classification
- regression/approximation
- cluster analysis
- pathway/regulatory network modeling/analysis
results
9 interpret and validate
- cross-validation
- statistical tests
- visual inspection of results
- biological validation
* against existing knowledge
* further experiments
new hypotheses, new insights, new knowledge
feedback
feedback

Figure 2.4: Microarray data analysis process - the big picture [BDG03].

described. There exists a multitude of other tools for analysing and interpreting microarray data, but they are not focused in this work.

## Low Level Analysis - Preprocessing

Preprocessing starts after the image processing. Data derived from image analysis are called *raw data* and stored in so called CEL files. These specially coded ASCII files containing fluorescence intensities for each probe on one microarray. Many preprocessing methods have been proposed for high-density oligonucleotide array data [IHC$^+$03]. Methodology for preprocessing Affymetrix GeneChip probe-level data is discussed in this section. However, many of these procedures apply to high-density oligonucleotide platforms in general.

The data recorded by means of the microarray technique are characterized by high levels of noise induced by the preparation, hybridization and measurement processes (Step 3-5). These perturbations have to be removed, therefore, preprocessing of raw-data has been a research topic of high priority over the past few years. In Section 2.2.3, the variations between arrays will be described with a statistical error model. Preprocessing usually involves three steps: background correction, normalization and summarization [GCH$^+$05].

**Quality Assessment and Control:** *Quality assessment* (QA) and *quality control* (QC) are an essential part of the data analysis process. The term *quality assessment* deals with the computation and interpretation of metrics that are intended to measure quality. The term *quality control* is used for possible subsequent actions, such as removing data or redoing experiments. Bad arrays should be identified and removed as early as possible in the process. Mostly this step will be conducted before preprocessing.

**Background Correction:** The term *background correction*, also referred to as *signal adjustment*, describes a wide variety of methods. More specifically, a background correction method should perform some or all of the following:

1. Corrects for background noise and processing effects.

2. Adjusts for cross hybridization, which is the binding of non-specific DNA (i.e. non-complementary binding) to the array.

3. Adjusts expression estimates so that they fall on the proper scale, or are linearly related to concentration.

It is important to note, that this definition is somewhat broader than is often used in the wider community. Many times only methods dealing with the first problem have been referred to as background correction methods [BIAS03].

**Normalization:** *Normalization* is the task of manipulating data to make measurements from different arrays comparable. The purpose of normalization is to adjust for effects, which arise from variation in the microarray technology rather than from biological differences between the RNA samples or between the printed probes. Different efficiencies of reverse transcription, labeling or hybridization reactions among different arrays, physical problems with the arrays, reagent batch effects, and laboratory conditions cause systematic technical biases and need to be corrected. [HGJY01] discusses these possible sources in more detail.

Probe-level normalization for high-density oligonucleotide arrays has been investigated in [BIAS03], which compares the bias and variability of expression measures computed using different normalization methods. The work defines two classes of normalization methods: complete data methods and baseline methods. Complete data methods use information from across all arrays to produce the normalization. The baseline methods select one array to represent the typical array, and then all of the other arrays are normalized to that array. It has been discovered, that complete data methods are preferable to methods choosing a baseline array.

**Summarization:** GeneChip arrays work by using 11 different PM spots to target 11 separate 25 base long sections of a target gene's mRNA. The final step in preprocessing microarray data is the *summmarization*. It summarizes the data from these 11 separate probes into an expression value for the gene in question. There are a number of different ways how this can be achieved, but the final result is always a single expression value for each gene on each chip.

More details and adaptions for parallelization of preprocessing steps can be found in Chapter 3 and 5.

### High Level Analysis

High level analysis is the 'real' statistical analysis and interpretation of microarray data. There are many statistical approaches to analyze preprocessed data. *High level analysis* methods mostly have the goal to identify differential gene expression, which describes differences in the mean values between different genes, and gene-gene correlation, which represents correlation between different genes using graphs. Differential gene expression analysis of microarray data is fraught with many classical statistical issues, such as appropriate test statistics, replicate structure, sample size, outlier detection and statistical significance of results. The original and simplest approach to identify differentially expressed genes is to use a fold change criteria.

For an overview and introduction to different high level analyses tools see [GCH+05].

## 2.2.3 Microarray Error Model

Subtle variations between arrays, the reagents used, and the environmental conditions lead to slightly different measurements even for the same sample. The effects of these variations may be grouped into two classes:

**Systematic Effects:** *Systematic effects* affect a large number of measurements simultaneously and can be estimated and approximately removed.

**Stochastic Components:** *Stochastic components* or *noise* are completely random effects with no well-understood pattern.

An *error model* is a description of the possible outcomes of a measurement. It depends on the true value of the underlying quantity that is measured and on the measurement apparatus. Describing the errors with a stochastic model will be useful for preprocessing, for construction of inferential statements about experimental results, for quality assessment and other tasks. For example a model-based approach allows pooling information across multiple arrays.

### Additive-multiplicative Error Model

Most measurement technologies require a linear calibration curve to estimate the actual concentration of an analyte in a sample for a given response. Incorporating into the linear calibration the two types of errors, a generic model for the value of the measured intensity $y$ of a single probe on a microarray is given by

$$y = \alpha + \mu e^{\eta} + \epsilon$$

where $\alpha$ is the mean background (mean intensity of unexpressed genes), $\mu$ the expression level in arbitrary units, $\epsilon \sim N(0, \sigma_\epsilon)$ an error term for the standard deviation of the background, and $\eta \sim N(0, \sigma_\eta)$ an error term. This is the *additive-multiplicative error model* for microarray data, which was first proposed by [RD01]. Very similar models were used to develop algorithms for microarray data, for example [HvHS+02] proposes a variance stabilization method for the preprocessing.

**DChip: Model-based Analysis of Oligonucleotide Arrays**

Another model was proposed in [LW01]. The estimation procedure is based on a model of how the probe intensity values respond to changes of the expression levels of the gene. It models the perfect match (PM) and mismatch (MM) signal from each probe pair. The average of the PM-MM differences for all probe pairs in a probe set is used as an expression index for the target gene. $\theta$ denotes an expression index for the gene, $\phi$ is an additional rate of increase and $\epsilon$ is a generic symbol for a random error.

$$y = PM - MM = \theta\phi + \epsilon$$

The paper [LW01] argues, that there is a strong preference to base all computation directly on the differences between the PM and MM responses in a probe pair. Today analysis only using PM signals are preferred [GCH+05].

Based on this statistical model, it is possible to address several important analysis issues that are difficult to handle by using other approaches. These include accounting for individual probe-specific effects, and automatic detection and handling of outliers and image artifacts [LW01].

## 2.3 Next-Generation Sequencing

The term *DNA sequencing* refers to methods for determining the order of the nucleotide bases – adenine (A), guanine (G), cytosine (C), and thymine (T) – in a molecule of DNA. In essence, the DNA is used as a template to generate a set of fragments, that differ in length from each other by a single base. The fragments are then separated by size, and the bases at the end are identified, recreating the original sequence of the DNA.

In recent years, new sequencing schemes, also called *high-throughput sequencing* (HTS), *massively parallel sequencing*, *flow-cell sequencing* or *next-generation sequencing* have been proposed. The high demand for low-cost sequencing has driven the development of high-throughput sequencing technologies, that parallelize the sequencing process, producing thousands or millions of sequences at once. High-throughput sequencing technologies are intended to lower the cost of DNA sequencing beyond what is possible with standard methods.

### 2.3.1 Standard Methods

Since the introduction of DNA sequencing in the early 1970s, sequencing has become easier and orders of magnitude faster in the last years. The rapid speed of sequencing attained with modern DNA sequencing technology has been instrumental in the sequencing of the human genome in the Human Genome Project (`http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml`).

The most commonly used method of sequencing DNA – the *dideoxy or chain termination method* – was developed by Fred Sanger at the University of Cambridge in 1977

[SNC77]. The key to the method is the use of modified bases called dideoxy bases. When a piece of DNA is being replicated and a dideoxy base is incorporated into the new chain, it stops the replication reaction.

### 2.3.2 High-Throughput Sequencing

Today there are two core differences of HTS to Sanger's capillary sequencing. First of all the library is not constructed by cloning, but by a novel way of doing polymerase chain reaction (PCR). The fragments are seperated by physico-chemical means (emulsion PCR or bridge PCR). Furthermore, very many fragments are sequenced in parallel in a flow cell (as opposed to a capillary) and observed by a microscope with CCD camera. The main advantage of HTS is the ability to process millions of sequence reads in parallel rather than less than 100 at a time. Commercially sequencing machines are available from Roche '454', Illumina (formerly: Solexa) 'Genome Analyzer', Applied Biosystems 'SOLiD system', and Helicos 'Helicoscope'.

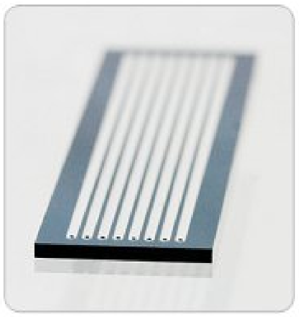


Figure 2.5: Illumina genome analyzer flow cell (`http://www.illumina.com/`). Up to eight samples can be loaded onto the flow cell for simultaneous analysis on the Illumina Genome Analyzer.

More details about high-throughput sequencing can be found in the manuals of the sequencing machines or in [HJ08, SJ08, Mar08, VDD09]. Relevant textbooks do not exist yet.

### 2.3.3 Analyses

At the moment there exist different use-cases for HTS in literature. Due to the cost reduction and ongoing technology development their number is growing daily. Several examples are listed in the following:

- De-novo sequencing and assembly of small genomes.
- Transcriptome analysis (RNA-Seq), especially for identifying transcripted regions and expression profiling.

- Resequencing to find genetic polymorphisms.
- ChIP-Seq, nucleosome positions, etc.
- Environmental sampling (metagenomics)

The applications are all very different. Therefore, it is difficult to define a standardized analysis process. Identical to the microarray data analysis process (see Figure 2.4) there is a feedback loop. But in the Steps 3 to 7 other technology and methods, and for Step 8 and 9 other analysis, interpretation and validation methods are used. For the data processing and analysis there are some similar procedures required in all applications, which are outlined in the following section. These methods are available from old sequencing technologies, but they have to be optimized especially for better performance.

Due to the higher number of sequences and the novel data structures that come along with this development, established procedures may not be suitable and new algorithms are required and will be developed in the future.

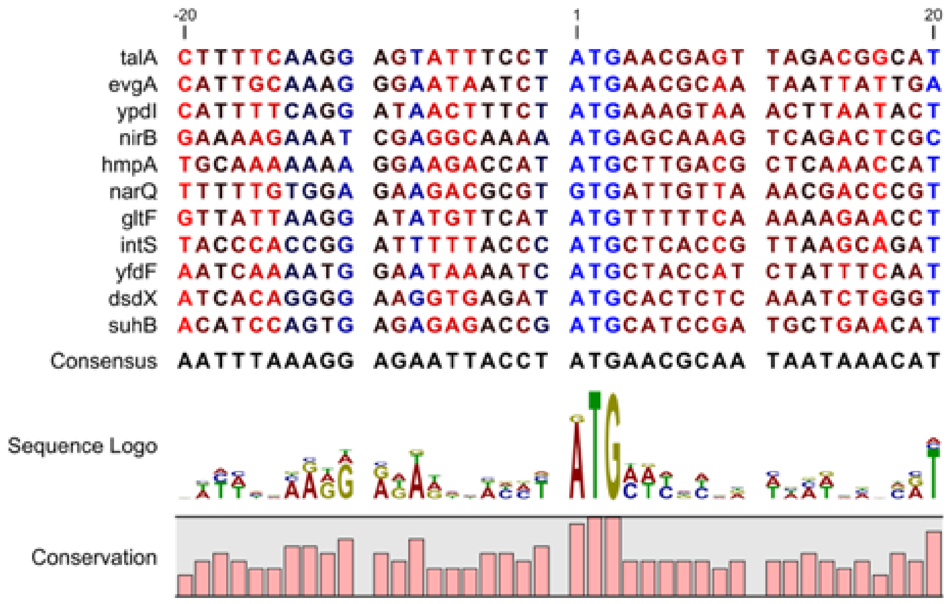
### Alignment

*Sequence Alignment* is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity, that may be a consequence of functional, structural, or evolutionary relationships between the sequences. Aligned sequences of nucleotide or amino acid residues are typically represented as rows within a matrix. Gaps are inserted between the residues so that identical or similar characters are aligned in successive columns (see Figure 2.6).

Computational approaches for sequence alignment generally fall into two categories: *global alignments* and *local alignments*. Global alignments, which attempt to align every residue in every sequence, are most useful, when the sequences in the query set are similar and of roughly equal size. Local alignments are more useful for dissimilar sequences that are suspected to contain regions of similarity or similar sequence motifs within their larger sequence context.

The main differences in alignment of next-generation sequence data to conventional sequences are the millions of very short reads, rather than a few long ones, which have to be mapped to the genome. Furthermore, dominant cause for mismatches are read errors and not substitutions. Base-call quality information is more important, only small gaps are expected, and mate-paired reads require special handling, . . . .

In the last two years, many commercial and open-source tools for short-read alignments have been published: Eland, Maq, Bowtie, Biostrings, BWA, SSAHA2, Soap, RMAP, SHRiMP, ZOOM, NovoAlign, Mosaik, Slider, . . . . The methods use different algorithms and differ in speed, memory requirements, accuracy, user interface and available downstream analysis tools.



| | | | | |
|---|---|---|---|---|
| talA | CTTTTCAAGG | AGTATTTCCT | ATGAACGAGT | TAGACGGCAT |
| evgA | CATTGCAAAG | GGAATAATCT | ATGAACGCAA | TAATTATTGA |
| ypdI | CATTTTCAGG | ATAACTTTCT | ATGAAAGTAA | ACTTAATACT |
| nirB | GAAAAGAAAT | CGAGGCAAAA | ATGAGCAAAG | TCAGACTCGC |
| hmpA | TGCAAAAAAA | GGAAGACCAT | ATGCTTGACG | CTCAAACCAT |
| narQ | TTTTTGTGGA | GAAGACGCGT | GTGATTGTTA | AACGACCCGT |
| gltF | GTTATTAAGG | ATATGTTCAT | ATGTTTTTCA | AAAAGAACCT |
| intS | TACCCACCGG | ATTTTTACCC | ATGCTCACCG | TTAAGCAGAT |
| yfdF | AATCAAAATG | GAATAAAATC | ATGCTACCAT | CTATTTCAAT |
| dsdX | ATCACAGGGG | AAGGTGAGAT | ATGCACTCTC | AAATCTGGGT |
| suhB | ACATCCAGTG | AGAGAGACCG | ATGCATCCGA | TGCTGAACAT |
| Consensus | AATTTAAAGG | AGAATTACCT | ATGAACGCAA | TAATAAACAT |

Figure 2.6: Ungapped sequence alignment of eleven E. coli sequences defining a start codon. The start codons start at position 1. The corresponding sequence logo is shown below the alignment (`http://www.clcbio.com/index.php?id=869`).

### Assembly

*Sequence Assembly* refers to aligning and merging fragments of a much longer DNA sequence in order to reconstruct the original sequence. This is needed as DNA sequencing technology cannot read whole genomes in one go, but rather in small pieces between 20 and 1000 bases, depending on the technology used. Typically the short fragments, which are called reads, result from shotgun sequencing genomic DNA, or gene transcripts (ESTs). Sequence assembly requires specialized software, typically based on so-called de-Brujin graphs. One of the most popular assembly tools is called 'Velvet' [ZB08].

Further general analysis tools for HTS are:

- statistical tests (counting statistics)
- visualizations
- segmentations

## 2.4 Other Biological Data & Analyses

This work focuses on biological data arising from the studies of genomes, especially microarray data and data from next-generation sequencing. But there are many other biological data, which will be focused on in statistical computing. It is not possible to list all available kinds of biological data. There are many examples, where computational tools are required to generate biological knowledge from data to better understand living systems. Following [Xio06], the main analysis applications of bioinformatics can be divided into three subfields (see Figure 2.7): *sequence*, *structural*, and *functional analysis*.

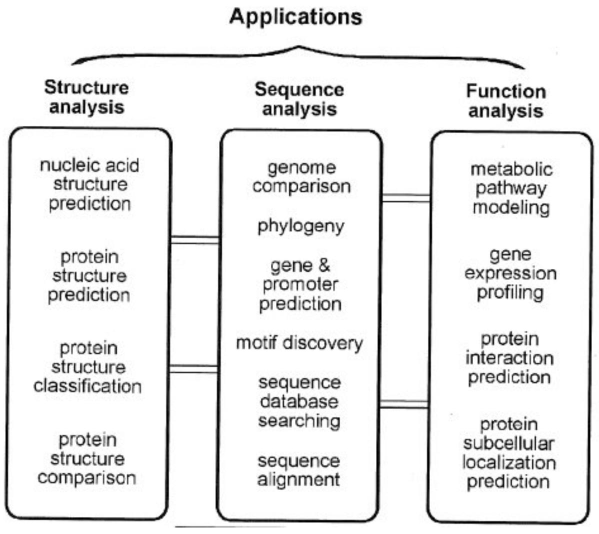

Applications
Structure analysis
nucleic acid structure prediction
protein structure prediction
protein structure classification
protein structure comparison
Sequence analysis
genome comparison
phylogeny
gene & promoter prediction
motif discovery
sequence database searching
sequence alignment
Function analysis
metabolic pathway modeling
gene expression profiling
protein interaction prediction
protein subcellular localization prediction

Figure 2.7: Overview of subfields of bioinformatics [Xio06] with structure, sequence and function analysis.

The are of sequence analysis includes sequence alignment, sequence database searching, motif finding and pattern discovery, gene and promoter finding, reconstruction of evolutionary relationships, and genome assembly and comparison. This thesis focuses on classical tools for next-generation sequence data. Other kinds of data are *ChIP-Sequencing* (ChIP-Seq), used to analyze protein interactions with *DNA* or *RNA-Sequencing* (RNA-Seq) to sequence cDNA in order to get information about a sample's RNA content.

Structural analysis includes protein and nucleid acid structure analysis, comparison, classification and prediction.

The functional analysis includes gene expression profiling, protein-protein interaction prediction, protein subcellular localization prediction, metabolic pathway reconstruction, and simulations. In addition to expression arrays, these types of arrays are for example data from tiling arrays (ChIP-chip), which are very similar to microarray data. Another example for functional analyses is *flow cytometry*. A technique for counting and examining microscopic particles suspended in a stream of fluid. It allows simultaneous multiparametric analysis of the physical and/or chemical characteristics of single cells flowing through an optical and/or electronic detection apparatus. Certain applications may include physical sorting of components.

The analysis of biological data often generates new problems and challenges, that in turn powers the development of new and better computational tools.

# Chapter 3

# Bioinformatics Using R and Bioconductor

The open-source programming language R and the Bioconductor open-source project for the analysis and comprehension of genomic data provide a wide spectrum of computational tools for the analysis of genomic data. This chapter introduces existing methods for DNA microarray analyses and next-generation sequence data. It discusses computational problems and challenges of existing programs. At the end it examines solutions to improve the performance and to allow analyses on huge numbers of biological data.

## 3.1 R and Bioconductor

*R* [R D08a] is an open-source programming language and software environment for statistical computing and graphics. The core R installation provides the language interpreter and many statistical and modeling functions. R was originally created by R. Ihaka and R. Gentleman in 1993 and is now being developed by the R Development Core Team. R is highly extensible through the use of packages. These are libraries for specific functions or specific areas of study, frequently created by R users and distributed under suitable open-source licenses. A large number of packages is available at the *Comprehensive R Archive Network* (*CRAN*) at `http://CRAN.R-project.org` or the Bioconductor repository [GCB$^+$04] at `http://www.bioconductor.org`. The R language was developed to provide a powerful and extensible environment for statistical and graphical techniques.

*Bioconductor* is an open-source and open-development software project and R package repository for the analysis and comprehension of genomic data. Bioconductor is primarily based on the R programming language and a repository for R packages. The Bioconductor project was started in fall of 2001 and is overseen by the Bioconductor core team, based primarily at the Fred Hutchinson Cancer Research Center (FHCRC, Seattle, WA, USA) with other members coming from various US and international institutions. There are currently (release 2.4, 21.April 2009) 320 contributed packages in Bioconductor's development repository. Releases occur twice a year, normally some days after a

| Release | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 | 1.6 | 1.7 | 1.8 | 1.9 |
|---|---|---|---|---|---|---|---|---|---|
| # Packages | 20 | 30 | 49 | 81 | 100 | 123 | 141 | 172 | 188 |
| Release | 2.0 | 2.1 | 2.2 | 2.3 | 2.4 | | | | |
| # Packages | 214 | 233 | 260 | 294 | 320 | | | | |

Table 3.1: Number of contributed packages included in each of the Bioconductor releases.

new release of the R language. The project also maintains more than 400 annotation data packages, that aid in the analysis of data from microarray experiments. Table 3.1 tracks the growth of the project over the semi-annual releases. The download statistic (`http://www.bioconductor.org/packages/stats/`) for Bioconductor software packages reports 150.000 package downloads per year.

The repository is split into three parts:

**Software:** Packages for diverse areas of high-throughput biological analysis.

**Metadata:** Bioconductor 'Annotation' packages contain biological information about microarray probes and the genes they are meant to interrogate, or contain ENTREZ gene-based annotations of whole genomes.

**Experiment Data:** Bioconductor 'Experiment Data' packages contain example data sets directly stored in R variables.

Packages in the software repository mainly address the development of high-quality algorithms for genome data analysis. Packages used for microarray analyses in this work and in connection to the described aspects in Chapter 2.2 will be presented in Section 3.2. Since the beginning of 2008 an ensemble of new or expanded packages introduces tools for next-generation DNA sequence data. Details for these packages are presented in Section 3.3.

## 3.2 Analyses of DNA Microarray Data

The open-source programming language R and the open-source project Bioconductor support the rapid developments in microarray technologies. Table 3.2 gives an overview of some commonly used packages for DNA microarray low and high level analysis including the corresponding references. All packages and download statistics are available at the Bioconductor website.

### 3.2.1 Low Level Analysis - Preprocessing

In Chapter 2.2 the general pipeline to preprocess Affymetrix microarray data is introduced: background correction, normalization and summarization. This section outlines the operation and theory of some commonly used and serial preprocessing methods implemented with R and Bioconductor. More details and code examples can be found in [GCH+05, BIAS03]

| Package | Functionality | Reference |
|---|---|---|
| *Low Level Analysis* | | |
| **affy** | Functions for exploratory oligonucleotide array analysis, especially for preprocessing. | [GCBI04, BIAS03] |
| **gcrma** | Main function converts background adjusted probe intensities to expression measures, using the same normalization and summarization methods as `rma`. | [WIG$^+$06] |
| **affyPLM** | Extends and improves the **affy** package and makes heavy use of compiled code for speed. Focus is on methods for fitting probe-level models and tools using these models. | [Bol04] |
| **vsn** | Implements a robust variant of the maximum-likelihood estimator for the stochastic model of microarray data. | [HvHS$^+$02] |
| **simpleaffy** | High level functions for reading CEL-files, phenotypic data, and then computing simple things with it, such as t-tests, fold changes, etc. Some basic scatter plot functions and mechanisms for generating high resolution journal figures. | [WM05] |
| **arrayQualityMetrics** | Performs quality metrics on microarray data, one or two channels. Results are designated to allow the user to rapidly assess the quality. | [KGH09] |
| *High Level Analysis* | | |
| **limma** | Data analysis, linear models and differential expression for microarray data. | [Smy04] |
| **genefilter** | Some basic functions for filtering genes. | [HHGF08] |
| **multtest** | Non-parametric bootstrap and permutation resampling-based multiple testing procedures. | [GCH$^+$05] |
| **GlobalAncova** | Support of the GlobalAncova approach. | [HMM08, MM05] |
| **pcalg** (CRAN) | Standard and robust estimation of the skeleton and the equivalence class of a Directed Acyclic Graph via the PC-Algorithm. | [KB07] |
| **GeneNet** (CRAN) | Analyzing gene expression data with focus on the inference of gene networks. | [SS05] |

Table 3.2: Overview about some R packages for microarray low and high level analysis.

or the corresponding references. Most of these methods are parallelized in the **affyPara** package.

### Quality Control & Assessment

Quality assessment is an important procedure, that detects divergent measurements beyond the acceptable level of random fluctuations. As microarray data quality can be affected at each step of the microarray experiment processing, quality assessment is an integral part of the analysis. The **affy** package implements tools for graphical quality assessment of microrarray data: `image()`, `boxplot()`, `hist()`, `MAplot()`, .... The **arrayQualityMetrics** package [KGH09] provides a comprehensive tool, that works on all expression arrays and platforms and produces a self-contained quality report, which can be web-delivered. Additional there are metrics included for automatical outlier detection of arrays.

### Background Correction

*Background correction* (BGC) methods are used to adjust intensities observed by means of image analysis to give an accurate measurement of specific hybridization. Therefore, BGC is essential, since parts of the measured probe intensities are due to non-specific hybridization and the noise in the optical detection system. The BGC methods *RMA*, *MAS 5.0*, and *Ideal Mismatch* are implemented in the function `bg.correct()` in the **affy** package and commonly used. Further methods are available in other packages (e.g., **affyPLM**) and can be used with the function `bg.correct(object, method, ...)`, too.

**RMA Background Correction:** *RMA background correction* is done by fitting a normal-exponential mixture model array by array and subtracting a background estimate from the PM value of each probe that is estimated in such a way, that the result is guaranteed to remain positive. Subsequently the data are logarithm transformed [IHC$^+$03]. RMA is motivated by the empirical distribution of probe intensities and ignores the different propensities of probes to undergo non-specific binding. Therefore, the background is often underestimated.

The characteristics of each probe can be determined by its sequence and the background noise can be described with a statistical model. Using this model the improved RMA background correction method GCRMA incorporates probe sequence composition into background adjustment [IHC$^+$03].

**MAS 5.0 Background Correction:** The *MAS 5.0 background correction* breaks the array into regions, within each grid it picks a background and noise value for that grid. The background-noise adjustment for each probe intensity is given by taking a weighted average of the grid background-noise values. The weights are dependent on the distance from the probe location to the center of each of the grids [Aff02].

**Ideal Mismatch Background Correction:** This method is discussed in [Aff02] and uses the MM probes to correct the PM probes. Originally PMs were corrected by subtracting MMs. However, many MM's are greater than the corresponding PM and therefore, simple subtraction creates difficulties. To sidestep this problem the MAS 5.0 algorithm uses an *Ideal Mismatch*, which is defined as the MM, when it is physically possible and a quantity smaller than the PM in other cases.

### Normalization

Normalization is required to compare measurements from different array hybridizations due to many obscuring source variations. Most of the commonly used functions are implemented in the **affy** package and can be used with the function `normalize(object, method="...", ...)`.

Two classes of normalization methods can be defined. *Complete data methods* use information from across all arrays to produce the normalization. The *baseline methods* select one array to represent the typical array, and then all of the other arrays are normalized to that array. It has been discovered, that complete data methods are preferable to methods choosing a baseline array [IHC$^+$03].

**Complete Data Methods - Quantile Normalization:** The goal of *quantile normalization* is to give the same empirical distribution of intensities to each array [BIAS03] and can be motivated using a quantile-quantile plot, which will have a straight diagonal line, with slope 1 and intercept 0, if two data vectors have the same distribution. The quantile normalization method is a specific case of the transformation $x'_i = F^{-1}(G(x_i))$ , where $G$ is estimated by the empirical distribution of each array and $F$ using the empirical distribution of the averaged sample quantiles.

**Complete Data Methods - Cyclic Loess Normalization:** The *cyclic loess normalization* is a generalization of the global loess method for two color arrays [YDL$^+$02]. When dealing with single channel array data, pairs of arrays are normalized to each other by using MA plots. The cyclic loess method normalizes intensities for a set of arrays by working in a pairwise manner. The procedure cycles through all pairwise combinations of arrays, repeating the entire process until convergence. One drawback is, that this procedure requires $O(n^2)$ loess normalizations. Usually only one or two complete cycles through the data are required.

**Baseline Methods - Scaling / Constant Normalization:** The *constant normalization* method, which was proposed by Affymetrix and used in both versions 4.0 and 5.0 of their software [Aff02, Aff04], chooses a baseline array and all the other arrays are scaled to have the same mean intensity as this array. This is equivalent to selecting a baseline array and then fitting a linear regression without intercept term between each array and the chosen array. Then, the fitted regression line is used as the normalizing relation. [BIAS03] describes four different methods for selecting the baseline array.

**Baseline Methods - Non-linear / Invariantset Normalization:** The *invariantset normalization* uses a non-linear relationship between each array and the baseline array can be used for normalization. Different relations have been proposed. For example, in [BIAS03] a loess smoother is discussed and implemented in the **affy** package.

### Summarization

Typically, Affymetrix GeneChip microarrays have hundreds of thousands of probes. These probes are grouped together into probesets. Within a probeset each probe interrogates a different part of the sequence for a particular gene. Summarization is the process of combining the multiple probe intensities for each probeset to produce an expression value for each probeset on the array.

Similar to the normalization methods, two classes of summarization methods can be defined: *Single-Chip Summarization* methods use only probe information on an individual array to compute expression summaries for that array. The expression values for each array are computed in isolation from information in other arrays. *Multi-Chip Model Summarization* is motivated by examining probe response patterns across arrays.

Most of the functions are implemented in the **affy** package and can be used with the function `computeExprSet(x, pmcorrect.method, summary.method, ...)`.

**Single-Chip Summarization - `avgdiff`:** This is an unrobust single chip method. It takes the mean of the log preprocessed PM probes for a particular probeset as the expression value for that probeset on that array. This is the approach, that was taken in [Aff02].

**Single-Chip Summarization - `mas`:** As documented in [Aff02], this method is a robust average using 1-step Tukey biweight on $log_2$ scale. The alogrithm combines probe values for a particular probeset from a single array to compute the expression value for that array.

**Multi-Chip Model Summarization - `liwong`:** This is an implementation of the methods proposed in [LW01]. The Li-Wong Model Based Expression Index (MBEI) is based upon getting the following multi-chip model to each probeset

$$y_{ij} = \phi_i \theta_j + \epsilon_{ij}$$

where $y_{ij}$ is $PM_{ij}$ or the difference between $PM_{ij} - MM_{ij}$. The $\phi_i$ parameter is a probe response parameter and $\theta_j$ is the expression on array $j$.

**Multi-Chip Model Summarization - `medianpolish`:** In the *medianpolish summarization*, used in the RMA methods, a robust multichip linear model is fitted to the log of the preprocessed PM probes for a particular probeset [IHC+03]. In particular for a probeset $k$ with $i = 1, \ldots, I_k$ probes and data from $j = 1, \ldots, J$ arrays, the following model is fitted

$$log_2(PM_{ij}^k) = \alpha_i^k + \beta_j^k + \epsilon_{ij}^k$$

where $\alpha_i$ is a probe effect and $\beta_j$ is the $log_2$ expression value. This method allows to combine information across chips. Please note, using this summary measure the expression values are in $log_2$ scale.

**Multi-Chip Model Summarization - FARMS:** 'Factor Analysis for Robust Microarray Summarization' (FARMS) is presented in [HCO06] and is a model-based technique for summarization. The method is based on a factor analysis model for which a Bayesian maximum a posteriori method optimizes the model parameters under the assumption of Gaussian measurement noise. Thereafter, the RNA concentration is estimated from the model. The code is available at the website `http://www.bioinf.jku.at/software/farms/farms.html`.

### Composite Preprocessing

There is a number of popular composite preprocessing algorithms, to combine the presented methods. As discussed, the procedure for generating expression measures can be considered as a three step process. Let X be raw probe intensities across all arrays and E be the probe set expression measures. Let B be the operation, which background corrects probes on each array, N be the operation, which normalizes across arrays and S be the operation, which combines probes together to compute an expression measure. The process of computing measures of expression can be written as:

$$E = S(N(B(X)))$$

**expresso:** The `expresso()` function provides quite general and simple facilities for computing expression summary values. B, N and S can be chosen arbitrary out of the existing methods. The trade-off is, that the function is often considerably slower than the functions, that have been optimized for producing specific expression measures.

**threestep:** The `threestep()` function is provided by the **affyPLM** package and has facilities for computing expression summary values. B, N and S can be chosen arbitrary out of the existing methods. It is primarily implemented in compiled code and therefore, typically faster than `expresso()`.

**RMA:** The `rma()` function is an expression measure consisting of three particular preprocessing steps: B is the RMA background correction process, N is the quantile normalization and S is an operation, which takes $log_2$ of the probes and fits a robust linear model, using the median polish algorithm.

**GCRMA:** The `gcrma()` function is an expression measure consisting of three particular preprocessing steps: B is the GCRMA background process, N is the quantile normalization and S is an operation, which takes $log_2$ of the probes and fits a robust linear model, using the median polish algorithm.

**vsn - Variance Stabilization Normalization:** The *vsn* [HvHS+02] method combines background correction and normalization into one single procedure to share information across arrays for background correction and normalization. The additive-multiplicative error model, described in Chapter 2.2.3, is used to estimate the perturbations in microarray data. Therefore, a complex and computer intensive robust variant of the maximum-likelihood estimator for the stochastic model has to be solved. The model incorporates data calibration, a model for the dependence of the variance on the mean intensity, and a variance stabilizing data transformation. For the normalization transformation a so called *generalized logarithm* 'glog' is used

$$x_{ki} \mapsto h_i(x_{ki}) = glog\left(\frac{x_{ki} - a_i}{b_i}\right)$$

where $x_{ki}$ is the intensity matrix, $b_i$ is the scale parameter for array $i$, $a_i$ is a background offset.

The code is available in the **vsn** package and can be applied using the `vsn2()` function. To get the expression values, additional a summarization method has to be used or the `vsnrma()` function, which calles `vsn2()` on the perfect match values only and calculates probeset summaries with the medianpolish algorithm of RMA.

### 3.2.2 High Level Analysis

High level analysis is the 'real' statistical analysis and interpretation of microarray data. There are many statistical approaches to analyze preprocessed data. Commonly used are differential gene expression, which describes differences in the mean values between different genes, and gene-gene correlation, which represents correlation between different genes using graphs. Different packages in the Bioconductor repository exist for these two methods and many other methods.

#### Differential Gene Expression

The simplest approach to identify differential expressed genes is to use a fold change criteria. However, single genes are not, in general, the primary focus of gene expression experiments. The researcher might be more interested in relevant pathways, functional sets, or genomic regions consisting of several genes. For example using the **GlobalAncove** package [HMM08, MM05], gene-wise linear models are used to formalize the relationship of gene expression with phenotypic or genomic covariates. An ANOVA-based sum of squares summarizes the individual gene-wise linear models to a group statement. A permutation test and an asymptotic distribution of the test statistics under the null hypothesis are available to calculate P-values.

#### Network Analyses - Correlation

The estimation of graphs or networks for genomic data is a very ongoing technology and a lot of studies address questions about the interaction between genes. In R and Bioconductor three useful methods – PC-Algorithm [KB07], Gene Correlation Networks (GeneNet) [SS05] and Graphical Lasso [FHT08] – are available.

**PC-Algorithm:** The *PC-Algorithm* is a method for estimating the skeleton and equivalence class of a very high-dimensional *Directed Acyclic Graph* (DAG) with corresponding Gaussian distribution [KB07]. It uses the PC-Algorithm, presented in [SGS01], to estimate a graph defined through conditional dependencies on any subset of the variables. The PC-Algorithm starts from the complete graph and deletes recursively edges based on conditional independencies. In the R language the method is available in the **pcalg** package. The algorithm is computationally feasible and it is difficult to evaluate the computational complexity of the PC-Algorithm exactly, but the worst case is bounded by $O(p^q)$ as a function of dimensionality $p$ (variables) and $q$ the maximal size of the neighbourhoods [KB07].

**GeneNet:** The **GeneNet** package is a R package for analyzing high-dimensional (time series) data obtained from high-throughput functional genomics assays, such as expression microarrays or metabolic profiling. Specifically, **GeneNet** allows to infere large-scale gene association networks. These are *Graphical Gaussian Models* (GGMs), that represent multivariate dependencies in biomolecular networks by means of partial correlation. Therefore, the output of an analysis – conducted by GeneNet – is a graph, where each gene corresponds to a node and the edges included in the graph portray direct dependencies between them [SS05].

**Graphical Lasso:** Another R package is the **glasso** package to estimate a sparse inverse covariance matrix using a lasso (L1) penalty. It can be used for estimating a sparse undirected graph. Using a coordinate descent procedure for the lasso, a simple and fast algorithm - the *Graphical Lasso* - is available [FHT08].

**Comparison:** A comparison and simulation study for Graphical Gaussian Models is available in [VSBH08]. It reports strong differences between the available methods and for the PC-Algorithm a good control of the FDR (False Discovery Rrate), when the parameter $\alpha$ is suitably chosen. Comparing the computation time of the methods for estimating graphs, the PC-Algorithm is about two times faster than the other ones. The comparison and improvement of the existing methods is an ongoing work. Especially the choice of the $\alpha$ parameter of the significance level for the individual partial correlation tests and the convergence criteria. A fix value of $\alpha = 0.05$ is commonly used, but for more than 60 nodes and more than 50% sparseness in the graph there is a huge bias. Figure 3.1 shows the structural hamming distance (SHD) between the original graph and the estimated graph. For the simulation 1500 normal distributed samples were generated from the original graph. The result is independent of the used number of observations and only small imporvements with a smaller $\alpha$ can be achieved.

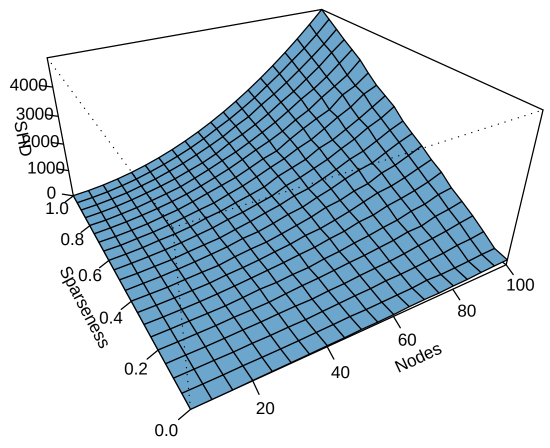


Figure 3.1: Visualization of the bias in the PC-Algorithm. Graphic plots the structural hamming distance between original and estimated graph for different numbers of nodes and sparseness in the original graph. 1500 normal distributed observataions and $\alpha = 0.05$ are used.

The PC-Algorithm was designed for estimating sparse graphs. KEGG graphs (pathways) analyzed in the large cancer study (see Chapter 7) have a nearly sparse graph structure. They have less than 20% of the maximal number of edges and the average number of nodes is 53 (average of edges: 173). In the large cancer study existing graph structures should be confirmed, therefore, a good control of the FDR is required. Due to the mentioned reasons, in this thesis the PC-Algorithm available in the **pcalg** package with a fix $\alpha = 0.05$ is used.

**Methods for Comparing Graphs:** There are only a few methods to compare graphs especially arising from microarray data. The **pcalg** package provides two useful methods to compare graphs.

**Structural Hamming Distance:** The *Structural Hamming Distance* (SHD) between two graphs is the number of edge insertions, deletions or flips in order to transform one graph to another graph. The smaller the SHD the bigger is the similarity between the two graphs. The SHD is symmetric and can be calculated using the function `shd(graph1,graph2)`.

**Rates:** The *True Positive Rate* (TPR) is the number of correctly found edges in the estimated (first graph parameter) divided by the number of true edges in the true (second graph parameter) graph. The *False Positive Rate* (FPR) is the number of incorrectly found edges in the estimated (first graph parameter) divided by the number of true gaps in the true (second graph parameter) graph. Therefore, a high TPR and a low FPR show a good similarity between the graphs. The rates can be calculated with the function `compareGraphs(graphEstimated, graphTrue)`.

**Graphical Comparison:** Another way to demonstrate differences or similarities between two graphs is the visualization of the graphs next to each other. As you can see in the graphs in this work, the visualization of graphs with more than 50 nodes and edges gets very complex. Especially there are no tools for automatic graphical comparison of two graphs. Highlighting same nodes or edges is possible, using the difficult code structure from the **Rgraphviz** package. But, there are no tools to plot two graphs with the same node structure. Therefore, a graphical comparison of graphs is not yet possible and a package for graphical comparison of graphs should be developed.

### 3.2.3 Computational Problems & Challenges

Independent from the used software for genetic analyses, actual research and computations are limited by the available computer hardware. Another challenge is the fact that microarray experiments are becoming increasingly popular, experiments are growing in the number of used arrays, and methodological advances have led to more computational demanding solutions.

#### Problem: Memory Limit

Actual research and computations are limited by the available computer hardware. For many users the available main memory - mostly 1 or 2 GB at a workstation - limits the number of arrays that may be quantified. In the `R` language the main memory limits are caused by the structure of the `AffyBatch` class. An `AffyBatch` will be created by importing CEL files into the `R` software and is a container for storing probe-level data, related phenotypic information and MIAME. The number of arrays, which can be imported strongly depends on the architecture of the computer system and microarray chip type. Table 3.3 shows some maximum numbers of CEL files of the Human Genome U133A

| System | max. CEL files |
|---|---|
| 64-bit linux system with 4 GB main memory | 400 |
| 32-bit linux system with 4 GB main memory | 160 |
| 32-bit Microsoft Windows XP system with 1 GB main memory | 60 |

Table 3.3: Maximum number of CEL files of the Human Genome U133A Affymetrix GeneChip® for creating an `AffyBatch` object at different computer systems.

Affymetrix GeneChip (HG-U133A), which can be used for creating an `AffyBatch` object at different computer systems. The same machine can yield results differing by up to 5%. Because of the different amount of 'outliers' the CEL file size can change by up to 3-5%. Using more arrays, a segmentation fault occurs in the R function `ReadAffy()`. Differences in system architecture and configuration render any prediction of the maximum number of microarrays impossible. Calculations like preprocessing methods on the `AffyBatch` require more main memory, which means that the amount of usable arrays will be smaller (see Table 3.4).

## Problem: Computation Time

Furthermore, most of the existing preprocessing methods are very time consuming and thus not useful for first and fast checks in laboratories. *Computation time* is the time a computer needs to complete a program. This closely depends on the speed of the computer processor. Different measurements show a nearly linear relation between computation time and the amount of arrays. The gradient depends on the kind of method used and the speed of the processor. Figure 3.2 shows the computation time in relation to the amount of microarrays

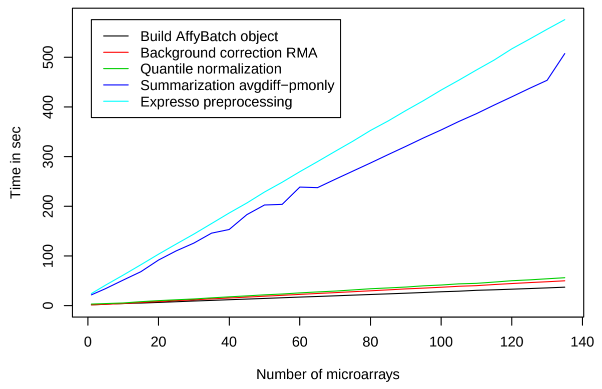


Figure 3.2: Computation time in relation to the amount of microarrays for several preprocessing methods. [expresso(..., bgcorrect.method='rma', normalize.method='quantiles', pmcorrect.method='pmonly', summary.method='avgdiff'); computeExprSet(..., pmcorrect.method='pmonly', summary.method='avgdiff')]

for several preprocessing methods. The time measurements were done at one node of the IBE computing poll cluster with 2.66 GHz and 6 GB main memory as described in Chapter 4.7. Summarization of the probes to probesets takes the most time. This is a very complex process and strongly depending on the size of the chip type, because in a loop over all probesets (HG-U133A: 22.000) the corresponding probe pairs (HG-U133A: 11) have to be summarized.

Similar results are presented in a simulation study on a website from Ben Bolstad (`http://bmbolstad.com/misc/ComputeRMAFAQ/size.html`). The simulations were performed at three different computer systems with a maximum main memory of 1 GB. The maximum number of CEL files for creating an `AffyBatch` object was about 100. For the computation time there are similar linear degrees. In the discussion the swapping problem is mentioned: 'There is some upwards bias in the times due to the way the simulation was run. Because of the way the operating system moves memory pages in and out of swap, if a large amount of memory was allocated previously, the following routines may also suffer.'

## Challenges

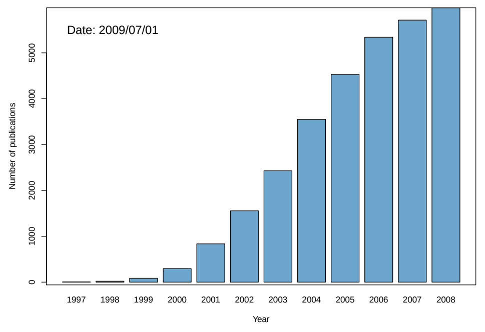
A further challenge is the fact, that microarray experiments are becoming increasingly popular. The large number of publications with the keyword 'microarray' published in the PubMed database (`http://www.ncbi.nlm.nih.gov/pubmed/`) shows the rapid development in this field during the last ten years (see Figure 3.3(a)).

In addition, microarray chips are becoming cheaper and the number of chips used in experiments is growing. The technology for creating chips is improving daily and the number of probes per chip is growing, too. Therefore, more and more data have to be managed and processed. Figure 3.3(b) shows the box-and-whisker diagrams[1] for the size of experiments published in the database ArrayExpress [PKS$^+$07] between 2003 and 2009. The mean size of the experiments stays stable about 10 arrays; however, the number of big experiments is growing. The red line visualizes the number of experiments with more than 150 microarrays. For instance, in 2007 the EMBLs European Bioinformatics Institute (EMBL-EBI, Cambridge, UK) launched an experiment containing 5896 CEL files (Affymetrix HG-U133A) to create a human gene expression atlas [PKS$^+$07, E-TABM-185]. Since 2003 there are 190 experiments with more than 150 microarrays and 8 experiments with more than 1000 arrays.

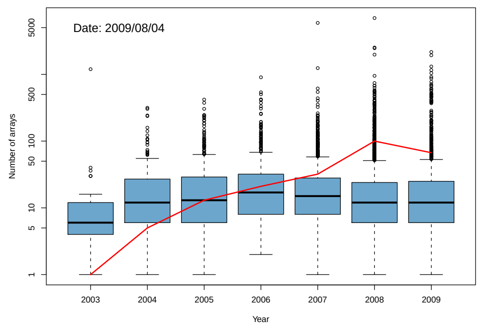
Advances in scanner technology, array manufacturing, analysis algorithms, sequence selection, probe design, and assay conditions have all contributed to improve the amount and quality of data obtained from a single GeneChip array. Additionally, a reduction in feature size has increased the data density on GeneChip arrays. The latest iteration of the human expression arrays is represented by the Human Genome U133 Plus 2.0 Array. Enhancements in array manufacturing, new scanner technology, and improvements in data acquisition allowed the further reduction in feature size from 18 microns to 11 microns. The Human Genome U133 Plus 2.0 Array contains over 54,000 probe sets representing approximately 38,500 genes on a single array [Aff04]. Figure 3.4 visualizes the evolution of feature size per array and number of probe sets per array for Affymetrix human genome chips.

Next to the microarray developments, methodological advances have led to more computationally demanding solutions in statistics. A common thread among these recent methods is the use of simulation and resampling techniques. For example in microarray data, building classification or prognostic rules, which uses resampling of the original data

[1]In descriptive statistics, a boxplot (also known as a box-and-whisker diagram) is a convenient way of graphically depicting groups of numerical data through their summaries: smallest observation, outliers (circles), whiskers (extend to the most extreme data point which is no more than 1.5 times the interquartile range from the box), lower quartile, median, upper quartile, and largest observation.



(a) Publications with the keyword 'microarray' published in the PubMed database between 1997 and 2008.



(b) Size of experiments published in ArrayExpress between 2003 and 2009. The red line visualizes the number of experiments with more than 150 microarrays.

Figure 3.3: Graphical visualization of the popularity of microarray experiments.

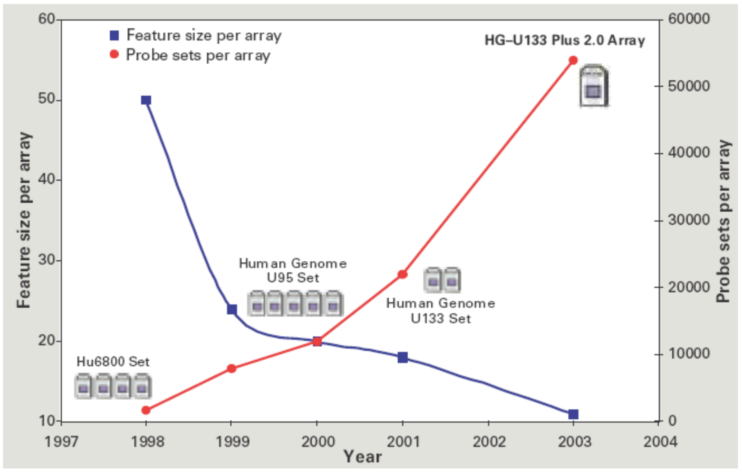


Figure 3.4: Evolution of feature size per array and number of probe sets per array from 1998 to 2004 [Aff04].

to estimate the classification or prognostic error [RHPM04], will be very time consuming. Here, preprocessing has to be part of the cross-validation and resampling strategy which is necessary to estimate the rule's prediction quality honestly.

## 3.3 Next-Generation Sequence Data

The open-source programming language R and the open-source project Bioconductor support the rapid developments in next-generation sequencing. The packages in the Bioconductor repository focus especially on down-stream (after alignment) analysis, quality assessment, data manipulation, ChIP-seq and other peak calling, and visualization problems. Due to the very new technology most packages are in development and the code structure is very unstable. At the moment (August 2009) no publications about the new Bioconductor packages and furthermore, no relevant textbook about high-troughput sequence analyses exist. This section gives an overview of the latest packages for next-generation sequence data in the Bioconductor repository.

### 3.3.1 Available Bioconductor Packages

All presented packages are available at the Bioconductor website. For more details see the vignettes or help files of the packages.

#### The Biostrings Package

The **Biostrings** package offers memory efficient string containers, string matching algorithms, and other utilities for fast manipulation of large biological sequences or set of sequences. Especially for the representation of DNA, RNA, amino acid, and general biological strings. For the sequence manipulation it provides functions for sequence summary (e.g., `alphabetFrequency()`), pattern matching (`matchDNApattern()`), subsequences and 'Views' and 'masks', and alignments (global, local, ends-free, . . . ).

#### The BSgenome Package

The **BSgenome** package provides an infrastructure for Biostrings-based genome data packages. Using this package new packages with genome data stored in classes of the **Biostring** package can be provided. To build the packages files containing the sequence data and files containing the mask data are required. The package provides a foundation for representing whole-genome sequences. At the moment there are 13 model organisms represented by 20 distinct genome builds available:

```
R> library(BSgenome)
R> available.genomes()
```

```
 [1] "BSgenome.Amellifera.BeeBase.assembly4"
 [2] "BSgenome.Amellifera.UCSC.apiMel2"
 [3] "BSgenome.Athaliana.TAIR.01222004"
 [4] "BSgenome.Athaliana.TAIR.04232008"
 [5] "BSgenome.Btaurus.UCSC.bosTau3"
 [6] "BSgenome.Btaurus.UCSC.bosTau4"
 [7] "BSgenome.Celegans.UCSC.ce2"
 [8] "BSgenome.Cfamiliaris.UCSC.canFam2"
 [9] "BSgenome.Dmelanogaster.UCSC.dm2"
[10] "BSgenome.Dmelanogaster.UCSC.dm3"
[11] "BSgenome.Drerio.UCSC.danRer5"
[12] "BSgenome.Ecoli.NCBI.20080805"
[13] "BSgenome.Ggallus.UCSC.galGal3"
[14] "BSgenome.Hsapiens.UCSC.hg17"
[15] "BSgenome.Hsapiens.UCSC.hg18"
[16] "BSgenome.Hsapiens.UCSC.hg19"
[17] "BSgenome.Mmusculus.UCSC.mm8"
[18] "BSgenome.Mmusculus.UCSC.mm9"
[19] "BSgenome.Ptroglodytes.UCSC.panTro2"
[20] "BSgenome.Rnorvegicus.UCSC.rn4"
[21] "BSgenome.Scerevisiae.UCSC.sacCer1"
```

Additional there are some functions to manipulate and process the whole genome data. The `bsapply()` function applies a function FUN to each chromosome in a genome using the parameters contained within the `BSParams` object. This object holds the various parameters needed to configure the `bsapply()` function.

### The ShortRead Package

The **ShortRead** package offers base classes, functions, and methods for representation of high-throughput, short-read sequence data. Especially for data management, I/O, manipulating, and quality assessment of short read data of single-end Solexa data. For data management and I/O the package provides functions to navigate in the output directory structure of the Solexa Genome Analyzer sequencing machine and to read and filter the raw data. Additional there are functions (e.g., `qa()`) to summarize read and alignment quality and to create quality reports.

### Further Packages

Next to the three mentioned packages there are several other packages for next-generation sequence analysis in the Bioconductor repository and some others in development:

**IRanges & genomeIntervals:** These packages offer an emerging infrastructure for representing very large data objects, for rangebased representations, and for manipulating intervals on sequences.

**rtracklayer:** Extensible framework for interacting with multiple genome browsers (currently UCSC built-in) and manipulating annotation tracks in various formats (currently GFF, BED and WIG built-in).

**HilbertVis & HilbertVisGUI:** These packages provide creative approaches for visualization of long vectors of integer data (or sequence data), using space-filling (Hilbert) curves that maintain, as much as possible, the spatial information implied by linear chromosomes.

**Chipseq:** (in development) A package with tools for helping to process short read data for Chip-Seq experiments.

### 3.3.2 Computational Problems & Challenges

The computational problems are very similar to the mentioned problems for micorarray data (see Section 3.2.3). The continuing exponential accumulation of full genome data, including full diploid human genomes, creates new challenges not only for understanding genomic structure, function, and evolution, but also for the storage, navigation and privacy of genomic data. Independent from the used software for next-generation sequence data, actual research and computations are limited by the available computer hardware.

Data from high-throughput sequencing experiments are very large. They consist of 10s to 100s of millions of 'reads' (each 10s to 100s of nucleotides long) and are coupled with whole genome sequences (for example, 3 billion nucleotides in the human genome). Currently, publicly available genomes are typically stored as flat text files in the GenBank (`http://www.ncbi.nlm.nih.gov/Genbank/`), but this approach is unlikely to scale up in many ways. The storage of the diploid genomes of all currently living humans using this simple approach would take 'GenBank', without counting headers or any additional annotations, on the order of $36 \times 10^{18}$ bytes, or 36 Petabytes, an amount difficult to store or download over the Internet, even using standard compression technologies (e.g., gzip) [BWB09]. First developments in data structures and algorithms addressing these problems are in progress.

Furthermore, first generation approaches with relatively short reads, restricted application domains, and small numbers of sample individuals are being supplanted by newer technologies producing longer and more numerous reads. New protocols and the intrinsic curiosity of biologists are expanding the range of questions being addressed, and creating a concomitant need for flexible and high-performance software analysis tools. The increasing affordability of high-throughput sequencing technologies means that multi-sample studies with non-trivial experimental designs are just around the corner.

## 3.4 Solutions

Both increased data size and increased simulation demands have to be solved. Most of the observed problems could be removed by using faster computer processors and bigger main memories (e.g., 128 GB). But high-throughput experiments are becoming increasingly popular, buying a better computer can only be a temporary solution.

### 3.4.1 Existing Bioconductor Solutions

For microarray data, there are a number of preprocessing methods included in the **affy** and **affyPLM** packages which try to solve these problems. For example, the `justRMA()` function reads CEL files directly in the working directory and converts the raw data - without using an `AffyBatch` object - to an expression measure using robust multi-array average [IHC+03, IBC+03]. With the `rma()` function about 250 microarrays can be preprocessed on the described computer. Using the `justRMA()` function about six times more arrays (1500) can be analyzed. The `threestep()` function [BCS+04] is primarily implemented in C code and is typically faster than the `expresso()` or `rma()` function. It is an alternative method of computing expression measures using the three described preprocessing steps. Table 3.4

| | 100 CEL files | 150 CEL files | 200 CEL files |
|---|---|---|---|
| `expresso` | 9.3 min | 29.6 min | segmentation fault |
| `threestep` | 0.8 min | 1.2 min | 1.6 min |

Table 3.4: Computation time improvement generated by special methods for preprocessing (system specific).

shows the improvements, which may be achieved for HG-U133A chips. The results were calculated at one node of the IBE computing poll cluster described in Chapter 4.7.

Unfortunately there are only a few methods available, which are designed for fast computations on a large amount of data and they are limited by the available computation hardware.

### 3.4.2 Further Solutions

Several different solutions were and are discussed in the user community to solve the mentioned problems. In the following five approaches are shortly discussed.

1) **Faster and Bigger Computers:** One of the simplest but expensive solutions is to buy faster and – in view of main memory – bigger computers. Today main memory is available in nearly unlimited size, but for linear increase in size, there is an exponential increase in costs and often a main memory limitation by the operation system. In the last years the computer chip development could only achieve a small profit in frequency and speed. Due to technical problems the frequency will stay stable and

the number of processors in computers will be growing. Therefore, code has to be (manually) adapted for the new generation of multiprocessor machines.

2) **Business Applications:** Another expensive solution is to buy commercial software from business partners. Several software companies offer software for high-dimensional genomic data and promise high-performance analyses on single computers. For example the 'Partek Genomics Suite' promises to be 'fast, memory efficient and will analyze large data sets on one personal computer'. Most commercial software do not provide latest algorithms and analyses tools. But using optimized GUIs they are often more userfriendly than the command line oriented R and Bioconductor packages.

3) **Databases:** A database is a structured collection of data that is stored in a computer system. The structure is achieved by organizing the data according to a database model. Standards for storing biological data exist and could be used to develop appropriate database models. First approaches using flat table[2] databases for microarray annotation data (e.g., **hgu133a.db** package) exist, using SQLite and the **RSQLite** package. Due to the high-dimensionality of biological data, more complex and efficient software (database management systems) will be required for efficient use and have to be additionally installed at the users workstation. Furthermore, tools for indexing, concurrency, security, etc. will be required. Databases are the preferred storage method for large multiuser applications, where coordination between many users is needed. If there are more read than write operations, databases perform best for large multiuser applications.

4) **Hard Drive as Main Memory:** Using efficient data structure approaches for reading data from a file on request, instead of loading the file to the main memory, exist. Thereby the main memory problems can be solved, but additional I/O will be generated. These approaches are typically slow and special data structures are required.

   For microarray data this solution is implemented in the **aroma.affymetrix** package [Ben04]. It provides memory-efficient methods to perform basic preprocessing analyses, such as normalization and probe set summarization on Affymetrix data. An `AffymetrixDataset` object defines a set of Affymetrix data files (typically CEL files) on the file system, for which there are methods to access the data by probes or probe sets across arrays. The data is accessed as in memory, but is internally read from file on request. With this approach there is in theory no limitation in the number of arrays. But the data have to be stored in a complex data structure and the R code structure is more complex than the structure from the **affy** package. A detailed comparison to the developed **affyPara** package is available in Chapter 5.

5) **Distributed Computing:** Using the potential of parallel computing, where calculations are carried out simultaneously, is another option. For biological data and sta-

[2]A *flat file database* or *flat table* describes any of various means to encode a database model as a plain text file.

tistical computing, parallel computing does not appear to have been used extensively up to now [Sev03]. Message passing methods are most frequently used for parallelization on multicomputers. In the R language, basic libraries like the **Rmpi** [Yu02] and **snow** [RTL03] packages are still available for parallelization on process layer. First packages for multiprocessor machines are available, too. A detailed description about parallel computing with R can be found in Chapter 4.

Parallelization will increase the speed of execution. Connecting several computers can be used to increase the total available main memory. Computer cluster environments are available for many research institutes today.

Due to the costs and expected hardware limitations, the solutions one and two were dropped. The huge amount of expected data could be critical for database applications and the installation of additional database software is not user friendly. Existing algorithms using efficient data structures on the hard drive level already have performance problems. Distributed computing is the most promising solution, because it accelerates the methods and can solve the main memory problems. Parallel computing is, therefore, the solution, which was picked and implemented in this thesis.

# Chapter 4

# Parallel Computing using R

R is an open-source programming language and software environment for statistical computing and graphics. The core R installation provides the language interpreter and many statistical and modeling functions. The R language was developed to provide a powerful and extensible environment for statistical and graphical techniques. However, the development of the R language was not aimed at providing a software for parallel computing. Nonetheless, during the last decade a great deal of research has been conducted on parallel computing techniques with R.

*High-Performance Computing* (HPC) is a part of computer based computing. It includes tasks which require a huge amount of computing power and of memory. In most cases it uses supercomputers and computer clusters to solve advanced computation problems. But even the use of multi-core systems or *graphics processing units* (GPUs) or of optimized database applications or optimized C class structures belong to HPC. In the R user community and mailing-lists – especially the following topics of HPC – are of high interest.

- Parallel computing approaches for R.
- The use of batch or queue management systems with R.
- Profiling and debugging R code.
- Large-scale automation and scripting with R.

Due to the described problems and solutions in Chapter 3 this work focuses on parallel computing approaches.

*Parallel computing* is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ('in parallel'). In *distributed computing* a program is split up into parts that run simultaneously on multiple computers communicating over a network. Hence distributed computing is a form of parallel computing, but the word *parallel computing* is used to describe program parts running simultaneously on multiple processors in the same computer.

Section 4.1 is a general introduction to parallel computing. The following two sections deal with software and hardware environments for parallel computing and end with an overview figure (4.4) for notations and relations in HPC. Section 4.4 discusses important aspects of manual parallel coding, followed by a chapter about performance analyses for parallel algorithms. The next section introduces parallel computing with R and describes the two most promising and most frequently used packages. The chapter ends with the description and comparison of the three cluster environments used for this thesis.

## 4.1 Introduction to Parallel Computing

In general, parallel computing deals with hardware and software for computation in which many calculations are carried out simultaneously. Traditionally, software has been written for serial computation and runs on a single computer having a single *Central Processing Unit* (CPU). Therefore, a problem is broken into a discrete series of instructions and they are processed one after another. Only one instruction is processed at any moment in time. A simple serial processing process with $n+1$ instructions (tasks) is visualized in Figure 4.1.

In the simplest sense, parallel computing is the simultaneous use of multiple compute resources to solve a computational problem. Using multiple CPUs a problem is broken into discrete parts, that can be solved concurrently. Each part is further broken down to a series of instructions, which run simultaneously on different CPUs. Figure 4.1 shows a simple parallel processing pipeline with $n + 1$ tasks and $m$ sub-tasks. Depending on the algorithms every parallel task can have a different number ($m$) of sub-tasks.

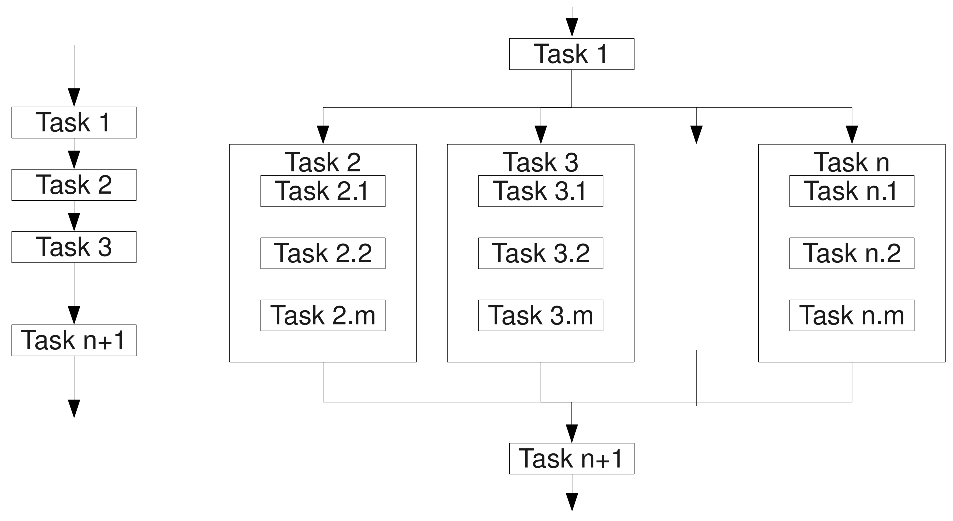


Figure 4.1: Simple illustration of serial (left) and parallel (right) processing with $n + 1$ tasks and $m$ parallel subtasks.

The compute resources of parallel computing can include ...

- a single computer with multiple processors,
- an arbitrary number of computers connected by a network,
- and a combination of both.

Parallel computing is an evolution of serial computing, that attempts to emulate what has always been the state of affairs in the natural world: many complex, interrelated events happening at the same time, yet within a sequence. Historically, parallel computing has been considered to be 'the high end of computing', and has been used to model difficult scientific and engineering problems found in the real world:

- Atmosphere, Earth, Environment
- Physics
- Chemistry, Molecular Sciences
- Geology, Seismology
- Mechanical Engineering - from Prosthetics to Spacecraft
- Electrical Engineering, Circuit Design, Microelectronics
- Computer Science, Mathematics

Today, commercial applications provide an equal or greater driving force in the development of faster computers. These applications require the processing of large amounts of data in sophisticated ways.

### 4.1.1 The Use of Parallel Computing

There are several main goals for the use of parallel computing.

**Save Time and/or Money:** Using $n$ processors could reduce the computation time by a maximum of factor $n$. Theoretically using more resources for a task will shorten time to completion with potential cost savings. Additional computer clusters can be built from cheap and commodity components.

**Solve larger Problems:** Many problems are so large and/or complex, that it is impractical or impossible to solve them on a single computer, especially given limited computer memory. For example "Grand Challenges" (`http://en.wikipedia.org/wiki/Grand_Challenge`) problems requiring PetaFLOPS and PetaBytes of computing resources.

**Provide Concurrency:** A single computing resource can only do one thing at the same time. Multiple computing resources can do many things simultaneously.

**Use of non-local Resources:** Using computing resources on a wide area network, or even the Internet when local compute resources are scarce. For example SETI@home (`http://setiathome.berkeley.edu`) uses about 300.000 computers for a compute power over 600 TeraFLOPS (`http://boincstats.com/stats/project_graph.php?pr=sah`).

**Limits to Serial Computing:** Both physical and practical reasons pose significant constraints of simply building ever faster serial computers:

- Transmission speeds - the speed of a serial computer is directly dependent upon how fast data can move through hardware. Absolute limits are the speed of light (30 cm/nanosecond) and the transmission limit of copper wire (9 cm/nanosecond). Increasing speed necessitates increasing proximity of processing elements.
- Limits to miniaturization - processor technology is allowing an increasing number of transistors to be placed on a chip. However, even with molecular or atomic-level components, a limit will be reached on how small components can be.
- Economic limitations - it is increasingly expensive to make a single processor faster. Using a larger number of moderately fast commodity processors to achieve the same (or better) performance is less expensive.

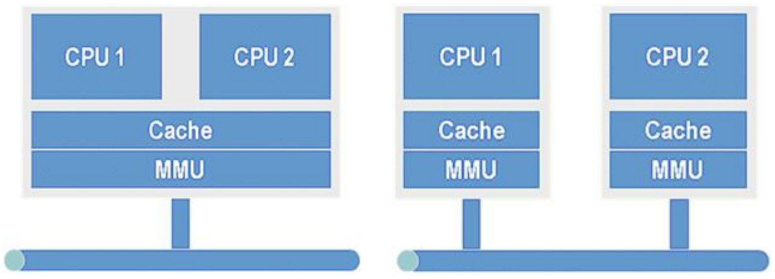
During the past 20 years, the trends indicated by ever faster networks, distributed systems, and multi-processor computer architectures (even at desktop level) clearly show, that parallelism is the future of computing.

## 4.2 Parallel Hardware Environments

There are several hardware types which support parallelism. For the work presented here, however, the software part is more important. Therefore, only the most popular hardware types are mentioned.

**Multi-core Systems:** In a *multi-core system* a multi-core processor combines two or more independent cores (normally a CPU) into a single package (see Figure 4.2(a)). These processors are typically installed in desktop computers or notebooks and in the year 2009 quad-core processors - containing four cores - are becoming a standard. The main memory is usually shared between all processing elements (*shared memory*).

**Multi-processor Systems:** In a *multi-processor system* two or more processors are installed in a single machine (see Figure 4.2(b)). This configuration is typically used for *server* machines and nowadays several multi-core processors will be used. These machines have a lot of main memory and can be used simultaneous from several users. The main memory is usually shared between all processing elements. In huge machines with a lot of processors *distributed memory* architectures are used. Each



(a) Multi-core system (b) Multi-processor system

Figure 4.2: Simple illustration of a multi-core and a multi-processor system. CPU: Central Processing Unit; MMU: Memory Management Unit

processor has its own private memory and a key issue is the distribution of the data over the memories.

**Multi-computer - Computer Cluster:** The idea of *multi-computer* environments – often called *distributed computers* – is the use of multiple computers to work on the same task. A distributed computer is a computer system in which the computers (processing elements with their own memory) are connected by a network.

Operation gets more difficult on heterogeneous multi-computers, where all the nodes do not have the same architecture, operating system and key component libraries (e.g., same R version). But in most cases multi-computer environments are homogeneous. If the computers are located in a local area network (LAN), they may be called a *computing cluster*.

**Multi-computer - Grid Computing:** In *grid computing*, machines are connected in a wide area network (WAN) such as the Internet. Next to the parallelization main aspects in grid computing are heterogeneous environments and resource allocation and management.

There are several other hardware environments which support parallel computing, but so far they are not used for parallel computing with the R language. For example *general-purpose computing on graphics processing units* (GPGPU) is a fairly recent trend in computer engineering research.

## 4.3 Parallel Software Environments

Several programming languages and libraries have been created for programming parallel computers. Kernel-level facilities targeted for high-performance computing (e.g., BProc, MOSIX) are well-developed on Linux operating systems, with additional implementations for Windows or Mac OS X. System-level tools focus especially on the management and monitoring of parallel tasks. A list and short description of some resource management systems is available in [SME+09]. User-level libraries are most important for the end user and parallel code designer. They can be classified based on the underlying memory architecture.

### 4.3.1 Shared Memory

Shared memory programming languages communicate by manipulating shared memory variables. Widely used shared memory application programming interfaces (APIs) include *OpenMP* [DM98] and *POSIX Threads* (Pthreads) [But97]. These are implementations of multi-threading, a method of parallelization whereby the manager (master) thread forks a specified number of worker (slave) threads and a task is divided among them. Each thread executes the parallelized section of code independently.

**OpenMP:** Open Multi-Processing is an API that supports multi-platform shared memory multiprocessing programming in C/C++ and Fortran. Commercial and open-source compilers for many architectures including Windows are available. OpenMP is a portable, scalable model that gives programmers a simple and flexible interface for developing parallel applications. The section of code, that is meant to run in parallel is marked accordingly by using a preprocessor directive, that will cause the threads to form before the section is executed.

**Threads:** POSIX Threads is a POSIX standard for threads. Libraries implementing the standard are often named Pthreads. For many architectures including Windows open source implementations exist. Pthreads defines a set of C programming language types, functions and constants. Programmers can use Pthreads to create, manipulate and manage threads, as well as to synchronize between threads using mutexes and signals.

OpenMP is under active development. Recent work indicates that it is easier to program with OpenMP than Pthreads and that OpenMP delivers faster execution times [BL00, Bin08].

### 4.3.2 Distributed Memory

*Message-passing* APIs are widely used in distributed memory systems. Message passing is a form of communication which is made by sending messages to recipients. A *manager-worker* architecture - often called *master-slave* - is most common. In this model of communication one device or process (manager) controls one or more other devices or processes (workers) (see Figure 4.3). Once a manager-worker relationship between devices or processes is established (spawned), the direction of control is always from the manager to the workers. Well known implementations of the standard are MPI (*Message-Passing Interface*) [For98] and PVM (*Parallel Virtual Machine*) [GBD+94].

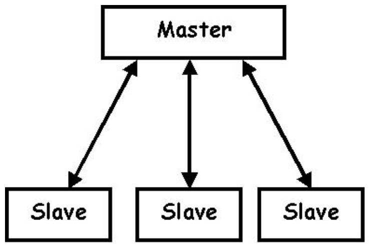


Figure 4.3: Illustration of master-slave or manager-worker architecture.

**MPI:** Message-Passing Interface is a standardized and portable message-passing system designed by a group of researchers for use on a wide variety of parallel computers. The MPI interface is meant to provide essential virtual topology, synchronization, and communication functionality between a set of processes in a language-independent way. Open-source implementations for many architectures including Windows exist. MPICH2 and DeinoMPI currently provide a Windows implementation and OpenMPI and MPICH2 are popular on Unix. OpenMPI is a project combining technologies and resources from several other projects (e.g., LAM/MPI) with the stated aim of building the best MPI library available.

**PVM:** The Parallel Virtual Machine is designed to allow a network of heterogeneous Unix and/or Windows machines to be used as a single distributed parallel computer. Open-source implementations for many architectures including Windows exist.

A comparison of both approaches is available in [GL02]. For computer clusters PVM is still widely used, but MPI appears to be emerging as a de-facto standard for parallel computing.

### 4.3.3 Comparison

The choice of the appropriate user-library is very difficult and often depends on the application and available hardware. Shared memory programming languages were developed for computer systems with shared memory to share or exchange data. These programming languages (especially OpenMP) do not work in distributed memory environments (computer clusters). In these environments message passing will be used for communication and coordination of data between processes. MPI is a de-facto standard and message-passing APIs (especially MPI) work on shared and distributed memory systems. In special systems shared and distributed programming languages (especially OpenMP and OpenMPI) can be used simultaneously. This is called *hybrid parallelization* and MPI will be used for the coarse grain parallelization and OpenMP for fine scaling of single tasks.

In most cases for existing, serial code and new developments the parallelization is more simple and faster with shared memory programming languages. However, MPI is more flexible, runs on both hardware systems and can be used for a more complex program structure. The hybrid parallelization could be very efficient, but parallelization gets very complex.

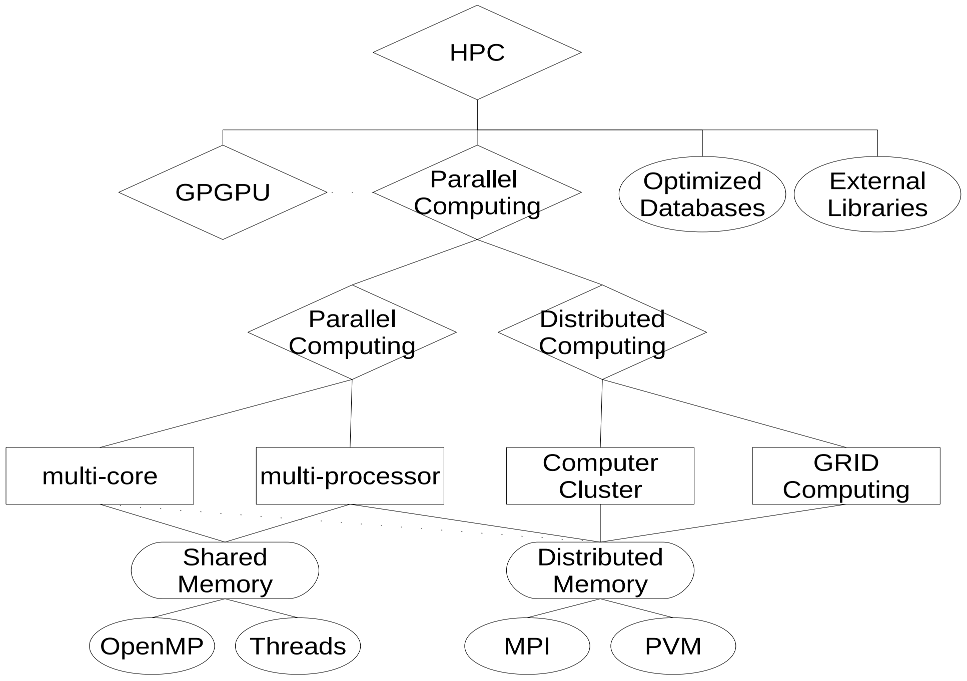


Figure 4.4: Illustration of notations and relations in high-performance computing.

The notations and relations in HPC can be confusing and due to the fast developments in chip technology they are changing very often. Figure 4.4 gives an overview of the relations in actual HPC. The figure does not illustrate the full bandwidth of HPC, but the important aspects for this work.

Due to the available R packages and the compatibility to different hardware systems parallel programming languages – especially MPI – for distributed memory systems will be used in this work. Therefore, developments will be working on all architectures for parallel computing.

## 4.4 Parallel Program Design

Designing and developing parallel programs has characteristically been a very manual process. The programmer is typically responsible for both identifying and implementing parallelism [GKKG03, Slo04]. Very often, manually developing parallel codes is a time consuming, complex, error-prone and iterative process. For a number of years, various tools have been available to assist the programmer with converting serial programs into parallel programs. The most common type of tool used to automatically parallelize a serial program is a parallelizing compiler or pre-processor. However, there are several important reasons to prefer manual to automatic parallelization: Wrong results may be produced, performance may actually degrade, much less flexible than manual parallelization, code is too complex for automatic parallelization, etc..

Therefore, the remainder of this section deals with several important manual methods of developing parallel code.

### 4.4.1 Analysing the Serial Code

The first step in developing parallel software is to understand the problem, that has to be solved in parallel. If a serial program is available, it necessitates to understand the existing code. Before spending time in the attempt to develop a parallel solution for a problem, determine whether or not the problem is one that can actually be parallelized and if there are methods to accelerate the serial code. Several profilers and performance analysis tools exist to identify program's hotspots[1] or bottlenecks[2]. A number of tools are available to profile R code for memory use and evaluation time. How to use these tools and how to detect probable bottlenecks is described in [R D08b], [Ven01], [Gen08], and in the R help page `?Rprof`. The CRAN packages **proftools** [Tie07] and **profr** [Wic08] provide more extensive profiling tools.

A classical example of a non-parallelizable problem is the calculation of the Fibonacci series (1,1,2,3,5,8,13,21,...) by use of the formula:

$$F(k+1) = F(k) + F(k-1)$$

This is a non-parallelizable problem, because the calculation of the next Fibonacci (k+1) number depends on the ultimate (k) and penultimate (k-1) sequence. These three terms cannot be calculated independently and therefore, not in parallel.

### 4.4.2 Partitioning

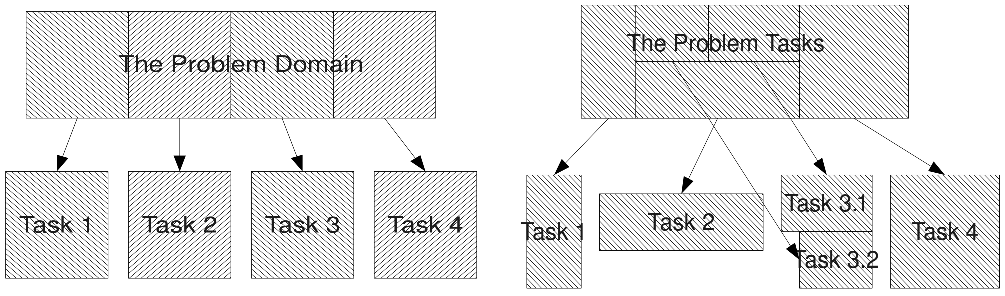
One of the first steps in designing a parallel program is to break the problem into discrete 'chunks' of work, that can be distributed to multiple tasks. This is known as *decomposition* or *partitioning*. There are two basic ways to partition computational work among parallel tasks [GKKG03]. Both are visualized in Figure 4.5 for a decomposition into four task for four different processors.

#### Data Decomposition

In *data decomposition* or *domain decomposition* the data associated with a problem will be decomposed. Each parallel task then works on a portion of the data. This decomposition is useful for problems, where data is static, dynamic data structure tied to a single entity and the entity can be subsetted (e.g., large multi-body problems), and domain is fixed but computation within various regions of the domain is dynamic (e.g., fluid vortices models).

[1]A hotspot is a code part where most of the real work is being done.

[2]A bottleneck is a code area, that is disproportionately slow.



(a) Data Decomposition (b) Task Decomposition

Figure 4.5: Graphical illustration of the two basic decomposition strategies: data and task decomposition.

There are many ways to decompose data into partitions. For example for two dimensional data there are 'Block Block Distribution', 'Block Cyclic Distribution' and 'Cyclic Block Distribution'.

### Task Decomposition

In *task decomposition* or *functional decomposition* the focus is on the computation, that is to be performed rather than on the data manipulated by the computation. The problem is decomposed according to the work that must be done. Each task then performs a portion of the overall work and the different tasks can be distributed to multiple processors for simultaneous execution. This partition is useful, if there is no static structure or fixed determination of calculation numbers to be performed.

## 4.4.3 Further Aspects for Parallel Program Design

There are several further aspects which have to be considered in the design of parallel programs. Some interesting topics for this work are listed below. For more details and aspects see relevant technical literature [Slo04, GKKG03, DFF$^+$03].

**Communication:** The need for *communication* between tasks depends upon the problem. In general calculation is faster than communication and therefore, communication should be reduced.

**Data Dependency:** A dependence exists between program statements when the order of statement execution affects the results of the program. A *data dependence* results from multiple use of the same location(s) in storage by different tasks. Dependencies are important to parallel programming because, they are one of the primary inhibitors to parallelism.

**Load Balancing:** *Load balancing* refers to the practice of distributing work among tasks so that all tasks are kept busy all of the time. It can be considered a minimization of task idle time. Load balancing is important to parallel programs for performance reasons.

**Granularity:** In parallel computing, *granularity* is a qualitative measure of the ratio of computation to communication. Periods of computation are typically separated from periods of communication by synchronization events. In fine-grain parallelism relatively small amounts of computational work is done between communication events. The opposite is called coarse-grain parallelism. The most efficient granularity is dependent on the algorithm and the hardware environment in which it runs.

**Random Numbers:** Generating random numbers presents a particular problem for parallel programming. For example, if you are using a large number of random numbers on a number of different processors and using the same random number generator on each, there is a chance that some of the streams will overlap. However, there are tools available to fix these problems, e.g., SPRNG.

## 4.5 Parallel Performance Analysis

Performance analysis and tuning for parallel algorithms is very difficult. As with debugging, monitoring and analyzing parallel program execution is significantly more of a challenge than for serial programs. A number of tools for monitoring, and program analysis for parallel code are available. For debugging – especially of code running at the workers – only a limited number of tools exists.

### 4.5.1 Computation Time

First of all the *computation time* for different numbers of processors and different sizes of input data can be measured and visualized. Typically a

$$T_N \approx 1/N$$

trend can be seen for the computation time (T) plotted over the number of processors (N). In theory a $N$ times acceleration in computation is expected using $N$ processors.

### 4.5.2 Speedup

In parallel computing, *speedup* (S) refers to how much a parallel algorithm is faster than a corresponding sequential algorithm:

$$S_N = \frac{T_1}{T_N}$$

Where $N$ is the number of processors, $T_1$ the execution time of the sequential algorithm and $T_N$ the execution time of the parallel algorithm with $N$ processors. The speedup is called *absolute speedup* when $T_1$ is the execution time of the best sequential algorithm, and *relative speedup* when $T_1$ is the execution time of the same parallel algorithm on one processor.

### Amdahl's Law

One of the best rates for describing the limits and costs of parallel programming is *Amdahl's Law* [Amd67]. It states that the potential program speedup is defined by the fraction of code that can be parallelized (P):

$$S \leq \frac{1}{1-P}$$

If none of the code can be parallelized ($P = 0$) then the speedup is 1 (no speedup). If all of the code is parallelized ($P = 1$), the speedup is infinite (in theory). If 50% of the code can be parallelized, maximum speedup is 2, meaning the code will run twice as fast (see Figure 4.6). Introducing the number of processors (N) performing the parallel fraction of work, the relationship can be modeled by

$$S_N \leq \frac{1}{\frac{P}{N} + S}$$

where P is the parallel fraction and S the serial fraction. As visualized in Figure 4.6 there are limits to the scalability of parallelism. Due to parallelization, additional costs for

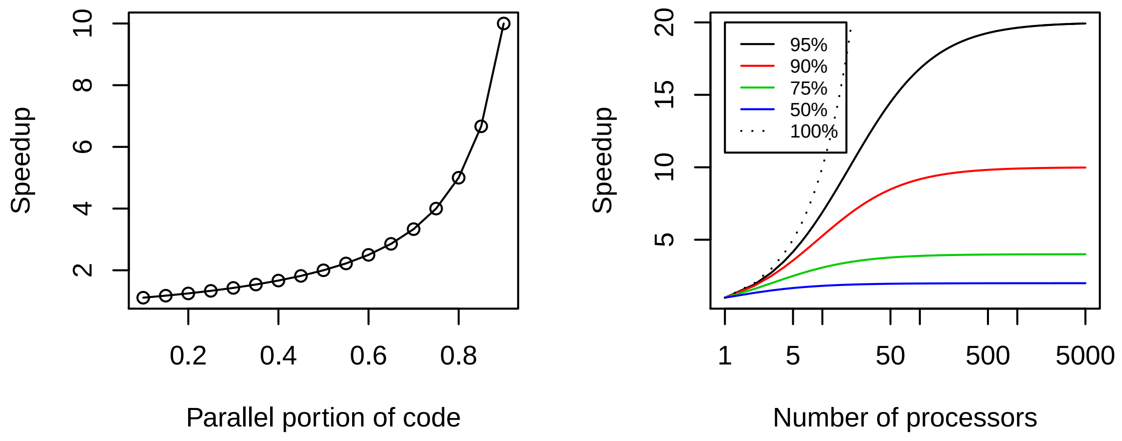


Figure 4.6: Visualization of theoretical speedup for parallel computing plotted for the parallel portion of code (left) and number of processors (right).

communication or synchronisation between processes are possible. Therefore, it is useful to add a parameter $o(N)$, which grows with increasing N.

$$S_N \leq \frac{1}{\frac{P}{N} + S + o(N)}$$

The speedup curves do not more convergence to $\frac{1}{1-P}$. They reach a maximum and then fall off (visualized in Figure 5.11). This effect commonly can be observed in practical examples. If the number of processors is big enough, the costs for communication exceed the computing time.

Amdahl's law was written in 1967 and new technologies – especially caching – have not been considered. Therefore, a super-linear speedup is sometimes possible. Sometimes a speedup of more than $N$, when using $N$ processors, is observed in parallel computing, which is called *super linear speedup*. Super linear speedup rarely happens, that could have different reasons: Bad serial code, cache effects, . . . .

### 4.5.3 Efficiency

Another performance metric is called *efficiency* and is defined as

$$E_N = \frac{S_N}{N}.$$

It is a value – typically between zero and one – estimating how well-utilized the processors are in solving the problem, compared to how much effort is wasted in communication and synchronization. Algorithms with linear speedup and algorithms running on a single processor have an efficiency of 1, while many difficult-to-parallelize algorithms have efficiency such as $\frac{1}{\log N}$ that approaches zero as the number of processors increases.

### 4.5.4 Karp-Flatt Metric

The *Karp-Flatt Metric* is a measure of parallelization of code in parallel processor systems. This metric exists in addition to Amdahl's Law as an indication of the extent to which a particular computer code is parallelized [KF90]. The experimentally determined serial fraction $e$ is defined as

$$e = \frac{\frac{1}{S_N} - \frac{1}{N}}{1 - \frac{1}{N}}.$$

The lower the value of $e$ the better the parallelization. In case of super-linear speedup the value becomes negative.

### 4.5.5 Resource Requirements

The primary intent of parallel programming is to decrease execution wall clock time. However, in order to accomplish this, more CPU time is required. For example, a parallel code that runs in one hour on eight processors actually uses eight hours of CPU time. The amount of memory required can be greater for parallel codes than serial codes, due to the need to replicate data and for overheads associated with parallel support libraries and subsystems. For short running parallel programs, there can actually be a decrease in performance compared to a similar serial implementation. The overhead costs associated with setting up the parallel environment, task creation, communications and task termination can comprise a significant portion of the total execution time for short runs.

Table 4.1 shows the CPU time and used main memory which was consumed for this PhD thesis at the IBE. The monitoring by the batch system 'Sun Grid Engine' is available since March 2009. In May 2009 the permutation test described in Chapter 7.4 was calculated.

| | March | April | May | June | July |
|---|---|---|---|---|---|
| CPU time in days - non parallel | 0.6 | 4.0 | 6.0 | 0.9 | 0.8 |
| CPU time in days - parallel | 246.3 | 118.5 | 1143.9 | 162.6 | 267.4 |
| Memory in GB - non parallel | 0.1 | 0.79 | 2.1 | 0.1 | 0.1 |
| Memory in GB - parallel | 7.2 | 3.9 | 55.2 | 12.3 | 13.8 |

Table 4.1: Used computer resources for this PhD thesis at the cluster at the IBE.

## 4.6 Parallel Computing using R

During the last decade more and more research has focused on using parallel computing techniques with R. The first available package was **rpvm** by Li and Rossini. This package provides a wrapper to the Parallel Virtual Machine software. Early papers describing parallel computing with R are [LR01], [Yu02], [Sev03] and [RTL03]. Interest in high-performance computing with R has been increasing particularly in the last two years. The R mailing lists (`http://www.R-project.org/mail.html`) now frequently host discussions about using R for parallel computing. The UseR!2008 Conference in Dortmund, Germany, and UseR!2009 Conference in Rennes, France contained tutorials, as well as several sessions on HPC with R where several new packages were presented.

The paper [SME+09] presents an overview of techniques for parallel computing with R on computer clusters, on multi-core systems, and in grid computing. It reviews sixteen different packages, comparing them on their state of development, the parallel technology used, as well as on usability, acceptance, and performance. An overview containing hyperlinks to all packages is available in Table 4.2.

Two packages (**snow**, **Rmpi**) stand out as particularly useful for general use on computer clusters. Both have acceptable usability, support a spectrum of functionality for parallel computing with R, and deliver good performance. Other packages try to improve usability, but so far usability gains have usually been achieved at the expense of lower functionality. Packages for grid computing are still in development, with only one package currently available to the end user. For multi-core systems four different packages exist, but a

| Package | Websites | Technology |
|---|---|---|
| *Computer Cluster* | | |
| **Rmpi** | `http://CRAN.R-project.org.org/package=Rmpi` | MPI |
| **rpvm** | `http://CRAN.R-project.org.org/package=rpvm` | PVM |
| **nws** | `http://CRAN.R-project.org.org/package=nws` | NWS and socket |
| **snow** | `http://CRAN.R-project.org.org/package=snow` | **Rmpi**, **rpvm**, **nws**, socket |
| **snowFT** | `http://CRAN.R-project.org.org/package=snowFT` | **rpvm**, **snow** |
| **snowfall** | `http://CRAN.R-project.org.org/package=snowfall` | **snow** |
| **papply** | `http://CRAN.R-project.org.org/package=papply` | **Rmpi** |
| **biopara** | `http://CRAN.R-project.org.org/package=biopara` | socket |
| **taskPR** | `http://CRAN.R-project.org.org/package=taskPR` | MPI (only LAM/MPI) |
| *Grid Computing* | | |
| **GridR** | `http://CRAN.R-project.org.org/package=GridR` | web service, ssh, condor, globus |
| **multiR** | `http://e-science.lancs.ac.uk/multiR` | 3 tier client/server architecture |
| Biocep-R | `http://biocep-distrib.R-forge.R-project.org` | java 5 |
| *Multi-core System* | | |
| **pnmath(0)** | `http://www.cs.uiowa.edu/~luke/R/experimental` | OpenMP, Pthreads |
| **fork** | `http://CRAN.R-project.org.org/package=fork` | Unix: fork |
| **multicore** | `http://CRAN.R-project.org.org/package=multicore` | fork |
| **rparallel** | `http://www.rparallel.org` | C++, file |
| **romp** | `http://code.google.com/p/romp` | OpenMP |

Table 4.2: Overview about parallel R packages for computer clusters, grid computing, and multi-core machines, including the corresponding hyperlinks and technologies.

number of issues pose challenges to early adopters. In January 2009 a new and promising package for the use of multi-core systems with R was released: **multicore**

A short introduction to the two most promising and commonly used packages is given in the following. A performance evaluation using a bootstrap example is available in Section 4.7. The new developed **affyPara** package uses the **snow** package.

### 4.6.1 The snow Package

The **snow** package (Simple Network of Workstations) [RTL07] supports simple parallel computing in R. The interface is intended to be simple, and is designed to support several different low-level communication mechanisms. Four low-level interfaces have been implemented: PVM (via the **rpvm** package), MPI (via **Rmpi**), NetWorkSpaces (via **nws**), and raw sockets that may be useful if PVM, MPI or NWS are not available. This means it is possible to run the same code at a cluster with PVM, MPI or NWS, or on a single multi-core computer.

The **snow** package includes scripts to launch R instances on the slaves. The instances run until they are closed explicitly via the stop command. The package provides support of high-level parallel functions like `apply()` and primitive error-handling to report errors from the workers to the manager. The following example code starts a cluster and calls the function `sum()` with the first element of the list on the first node, with the second element of the list on the second node, and so on. Additional the function `sum()` gets the parameter 3, therefore, it calculates 1+3, 2+3 and 3+3.

```
R> library(snow)
R> cl <- makeCluster(3, type = "MPI")

        3 slaves are spawned successfully. 0 failed.

R> res <- clusterApply(cl, 1:3, sum, 3)
R> unlist(res)

[1] 4 5 6

R> stopCluster(cl)

[1] 1
```

### 4.6.2 The multicore Package

The **multicore** package [Urb09] is a very new R package (released in January 2009) that provides functions for parallel execution of R code on machines with multiple cores or CPUs. It can not be used for multi-computer environments. Unlike other parallel processing methods all jobs share the full state of R when spawned, so no data or code needs to be initialized. The actual spawning is very fast as well, since no new R instance needs to be started. The package uses the `fork` system call to spawn a copy of the current process, which performs the computations in parallel. Modern operating systems use the copy-on-write approach, which makes this very appealing for parallel computation since only objects modified during the computation will be actually copied and all other memory is directly shared.

This is one of the most promising package developments for the integration of multiprocessor environments to R. Problems appear due to the absence of the `fork` command on Windows systems. The code for the same example as above has the following structure:

```
R> library(multicore)
R> res <- mclapply(1:3, sum, 3)
R> unlist(res)

[1] 4 5 6
```

## 4.7 Used Cluster Environments

For the evaluation of the new **affyPara** package different cluster environments were used. All systems are Unix based and the latest R and Bioconductor versions are installed. There are differences in the computation time on single processors due to different clock rates and software compilers.

**HLRB II: SGI Altix 4700:** The system commenced operation in 2006 in the new LRZ building in Garching, Germay. The peak performance is more than 62 TFlop/s, which is delivered by 9,728 Intel Itanium Montecito cores (1.6GHz). The main memory size is 39 TByte. The processors are connected with NUMAlink4 network architecture (Bandwidth of 6.4 GByte/s). In April 2009 the HLRBII was on place 44 in the TOP500 (`http://www.top500.org`) list, a ranking of supercomputers according to the LINPACK benchmark. The HLRB2 can be used as multiprocessor machine with up to 510 processors, or as multi-computer machine with 9728 machines, or as a hybrid-architecture. PBS Pro (the Portable Batch Queuing System) is used for the job management. (`http://www.lrz-muenchen.de/services/compute`)

**IBE - Computing Pool:** The system commenced operation in 2007 and is a computing pool with 32 computers. Each machine runs on 4 CPU cores (2 dual-core Intel Xeon DP 5150, 2.66 GHz) and 8 GB main memory and they are connected with a 1 Gbit Network. There is a maximum number of 128 processors. As master node the machine 'lis06' described bellow can be used. SGE (Sun Grind Engine) is used for the job management.

**IBE - lis06:** This system is a multiprocessor machine with 8 CPU cores (4 dual-cores Intel Xeon E7220, 2.93GHz) and 64 GB main memory. SGE is used for the job management.

### 4.7.1 Comparison of Used Cluster Environments

To compare the speed and performance of the three different cluster environments a simple bootstrap - benchmark was implemented. Bootstrapping is a classic example of a time-consuming but simple to parallelize computation. Bootstrap replicates of a generalized linear model fit for data on the cost of constructing nuclear power plants were generated. Available code from the **boot** package and the proposed parallelization from [RTL03] were used. For the parallelization the in Section 4.6 proposed packages **snow** and **multicore** were used. 5000 bootstrap replicates were calculated on different numbers of processors and the calculation was replicated 10 times to adjust for external factors (network traffic, other computations). The results are plotted in Figure 4.7, for the computation times the boxplots with the highest IQRs are added.

Due to the architecture of the cluster environments the computation times on the single processor should be comparable. The HLRBII with 1.6GHz has the slowest processors. Other differences in computation time are due to differences in the compilation of the R language (used compiler, profiling, . . . ). The step (64-65) in the speedup curve for the IBE computing pool occurs, if there are more than two processors running on one computer (32 available computers). Due to hardware architecture the main memory communication becomes a bottleneck. On the multi-processor machines (lis06 and HLRB2) both packages show very similar behavior and nearly linear speedup for the bootstrap benchmark up to 20 nodes. After 20 nodes a strange behavior occurs at the HLRB2. Due to multiple processes running on the same processor, there is no more linear speedup. This is a load balancing problem of the batch system PBS and operating system. If the computation times are long enough, a better performance can be reached. In all cluster environments there are only small differences (small IQRs, no outliers) in the time measurements of the 10 replicates. This is a good indicator for no foreign network traffic or load.

## 4.8 Summary

In summary there are many high-performance computing technologies available today. Most of them were not developed or optimized for applications in (Bio-)Statistics or Bioinformatics. Due to the flexible package structure, the R language provides a good interface to different high-performance computing resources. Satisfying solutions exist, especially for parallel computing in computer clusters and mulit-core environments.

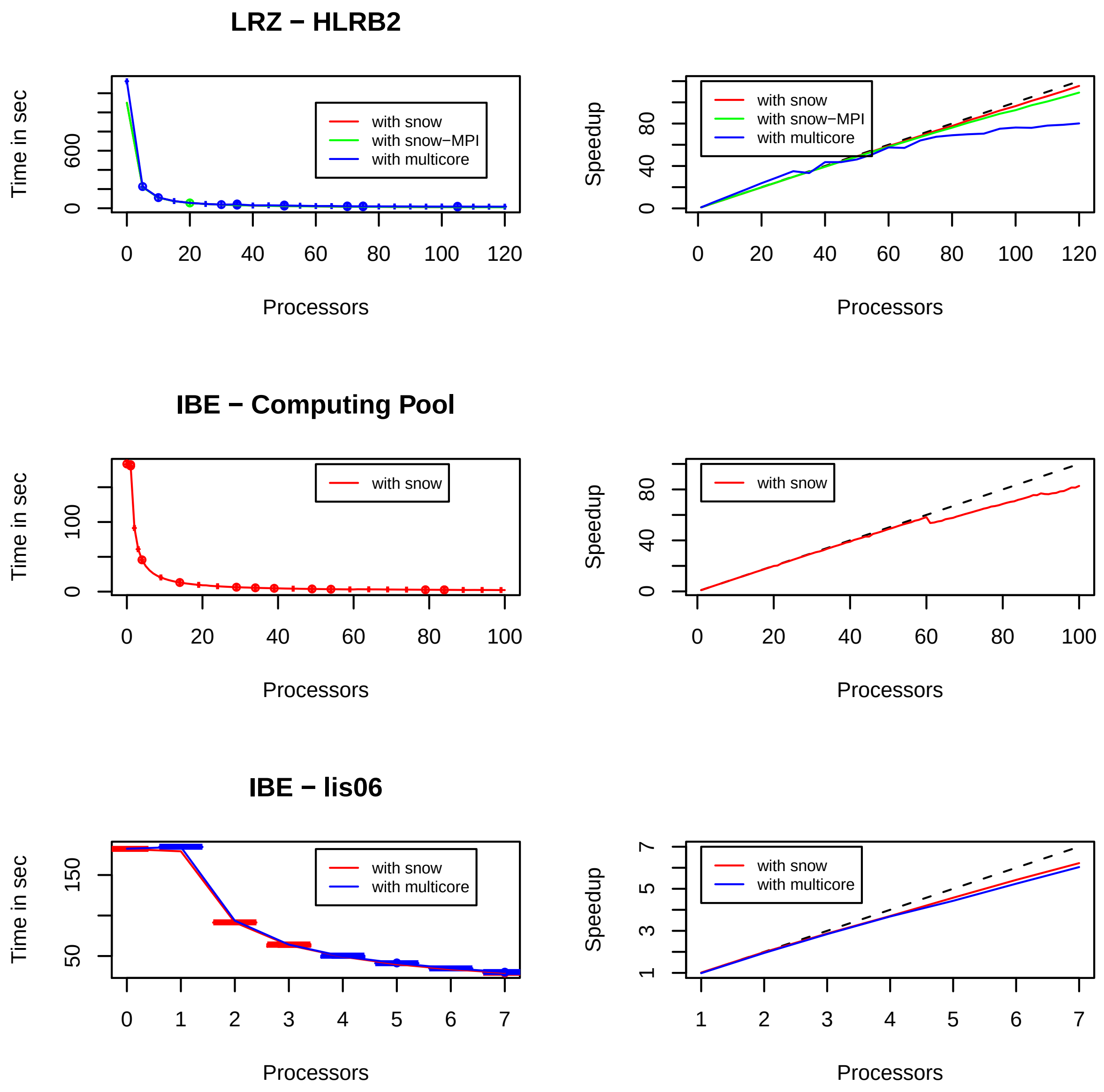


Figure 4.7: Performance (computation time and speedup) for used computer environments.

# Chapter 5

# Parallel Computing in Microarray Data: affyPara

The power of parallel computing is used to solve the mentioned problems and new challenges in microarray data preprocessing. Microarrays can be distributed to different processors or computers, thereby main memory problems are solved and methods accelerated. Existing statistical algorithms and data structures are adjusted and reformulated for parallel computing. Using the parallel infrastructure the methods are enhanced and new methods are developed. A new Bioconductor package called **affyPara** for preprocessing huge amounts of microarray data is presented in [SM08, SM09].

Section 5.1 outlines the main idea and aspects of the parallelization of preprocessing for microarray data. Section 5.2 describes the implementation of the **affyPara** package in detail and new approaches in parallel low level analysis. The chapter ends with results, speedup curves and a comparison to other existing solutions.

## 5.1 Idea

The main idea of parallel computing for microarray data is to use a block cyclic distribution to distribute arrays to different processors. In general, microarray data are stored in a matrix structure. In the columns there are the arrays or samples and in the rows there are the probe sets, genes or features (see Figure 5.1). Using the block cyclic distribution the arrays are distributed equally to all processors, therefore, the amount of required main memory per processor gets smaller. All existing data structures and objects (`AffyBatch`, `ExpressionSet`) for microarray data in the R language and in basic Bioconductor packages are array oriented. The block cyclic distribution is chosen to reuse existing code and therefore, all data (probes) from one array are available at one processor. Other decomposition strategies and distribution details are discussed in Section 5.3.

The **snow** package is used for implementation due to the available cluster environments, the compatibility of the **snow** package to multi-processor and multi-computer systems, and the user friendly code interface.

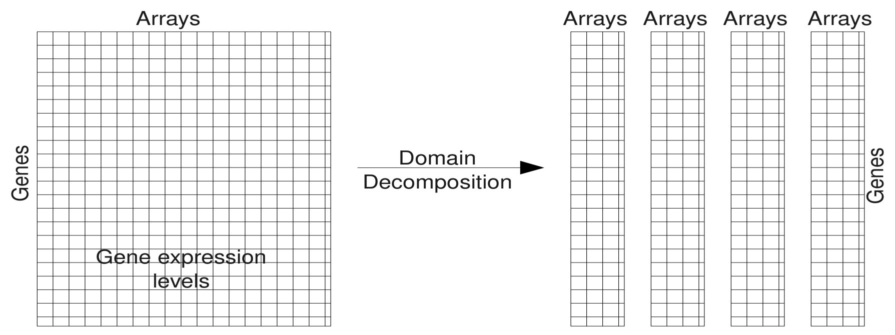


Figure 5.1: Block cyclic distribution for microarray data. The intensity matrix is split into four small matrices using domain decomposition.

## 5.2 Implementation

This section describes the development and implementation of parallel algorithms for preprocessing microarray data. For the visualization of the implementation structure flowcharts and for the computation time Gantt charts are used. A *flowchart* is a common type of chart, that represents an algorithm, showing the steps as boxes of various kinds, and their order by connecting these with arrows. A *Gantt chart* is a type of bar chart that typically illustrates a project schedule, but could be used for visualizing computation and communication time, too. In the **snow** package (Version > 0.3-3) Gantt charts can be created using the following functions:

```
R> library(snow)
R> t <- snow.time(expr)
R> plot(t, xlab = "Elapsed Time", ylab = "Node", title = "Cluster Usage")
```

The graphic represents active computation with green rectangles, blue horizontal lines represent a worker waiting to return a result, and red lines represent manager-worker communications. Ten nodes and 100 microarrays were used in this chapter for the Gantt charts. As example for the two different graphics see Figure 5.2.

### 5.2.1 Basic Architecture

All functions in the **affyPara** package can get the raw data in three different forms of input objects. Depending on the object class, there are differences in the parallel process:

**'AffyBatch':** The functions can get the raw data as an `AffyBatch` object. Therefore, the complete `AffyBatch` object has to be built at the master node and the mentioned main memory problems are not removed. In the parallel process the expression matrix in the `AffyBatch` object is partitioned (block cyclic) and distributed to the nodes.

**’CELfileVec’:** The functions can get the raw data as a list or vector of names (links to the location) of the CEL files. Therefore, the CEL files have to be available in the same directory at all nodes. Typically in cluster environments this is available with a shared file system (e.g., SAMBA or nfs) and as default in multi-processor environments. In the parallel process the list of CEL file names is partitioned, distributed to the nodes, and then the small `AffyBatch` objects are built at the nodes. In cluster environments with a shared file system the communication with the file system could be a bottleneck, because first all nodes connect to the file system to get the raw data.

**’partCELfileList’:** The functions can get the raw data as a partitioned list of names of the CEL files. CEL files can be distributed to the nodes (e.g., to temporary directories) in advance with an optimized data distribution function. This function generates a partitioned list of the names of the distributed CEL files. In the parallel process only the small `AffyBatch` objects have to be built at the nodes. The data distribution step is not required in multi-processor environments, while in big cluster environments it can take some time to distribute the data to all nodes.

Due to the experiences using the computing pool at the IBE the second solution is the best one. No connection problems with the shared file system could be detected with up to 120 nodes and about 7000 microarrays.

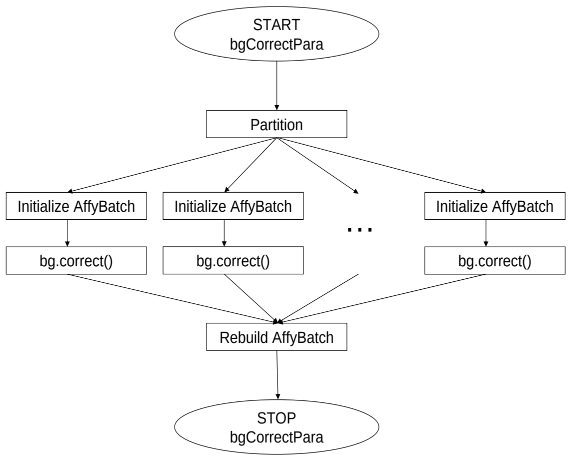
After the distribution of the raw data all parallel functions have the same code structure. First of all the small `AffyBatch` objects are initialized at the workers. Then depending on the kind of preprocessing method the parallel algorithm is executed. At the end the expression matrices are collected by the master node and the `AffyBatch` or `ExpressionSet` object is built at the master.

There are several internal checks to guarantee fail over and in every function debug options can be switched on using the `verbose=TRUE` parameter.

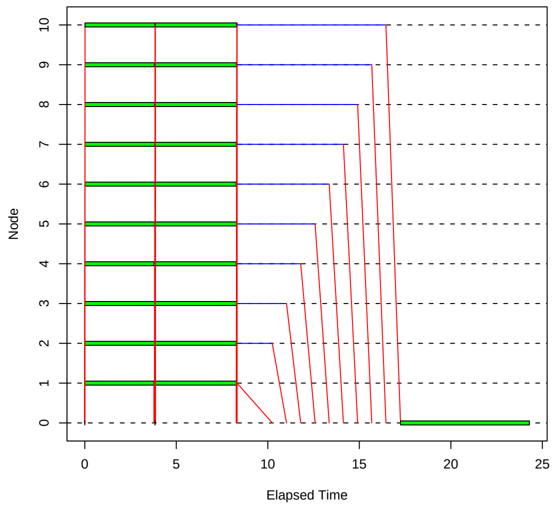
### 5.2.2 Background Correction

Background correction (BGC) methods are used to adjust intensities observed by means of image analysis to give an accurate measurement of specific hybridization. The existing methods are dependent on the actual sample only, therefore, they are easy to parallelize and existing serial code can be reused: partition of the data, initialization of the small `AffyBatch` objects at the workers, execution of the serial BGC methods with the small `AffyBatch` objects at the workers, sending results back to master and rebuilding the `AffyBatch` object. Figure 5.2 shows the programming flowchart and Gantt chart of the parallelized background correction function. In both graphics the implementation steps are visible. The Gantt chart shows a computation time of four seconds for the initialization process and about five seconds for the parallel BGC. The long communication and `AffyBatch` rebuilding time is discussed in Section 5.3.

Parallelized BGC methods are implemented in the function `bgCorrectPara()`. Available methods can be listed with the command `bgcorrect.methods()`. An example code



(a) Flowchart



(b) Gantt chart with CEL file list as input object.

Figure 5.2: Flowchart and Gantt chart for parallelized rma background correction. Both graphics visualize the implementation steps.

using a cluster with two MPI workers and the Dilution data set (four arrays) from the **affydata** package has the following structure:

```
R> library(affyPara)
R> library(affydata)
R> data(Dilution)
R> bgcorrect.methods()

[1] "mas"  "none" "rma"
```

```
R> makeCluster(2, type = "MPI")
```

```
R> bgc <- bgCorrectPara(Dilution, method = "rma", verbose = TRUE)

Partition of object 2.924 sec DONE
Object Distribution: 2 2
Initialize AffyBatches at slaves 4.341 sec DONE
BGC on Slaves 8.308 sec DONE
Rebuild AffyBatch 3.064 sec DONE
```

```
R> stopCluster()
```

### 5.2.3 Normalization

Normalization methods make measurements from different arrays comparable. Similar to BGC methods the baseline normalization methods only depend on the actual arrays and are easy to parallelize. The serial code can be reused at the workers. Complete data methods have proved to perform very well, but due to their multi-chip dependency they are more complex for parallelization.

#### Baseline Normalization Methods

These methods select one array to represent the typical arrays, and then all the other arrays are normalized to that array. The Gantt charts for the two baseline normalization methods constant and invariantset are visualized in Figure 5.3.

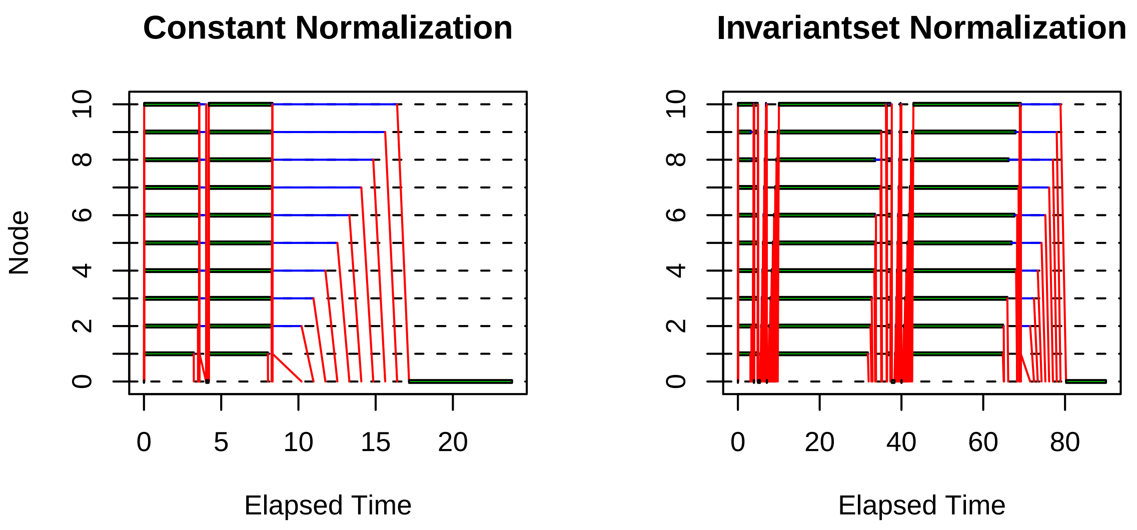


Figure 5.3: Gantt chart for parallelized constant and invariantset (PM and MM separate) normalization with CEL file list as input object.

**Constant Normalization:** After the `AffyBatch` object's initialization at the workers (four seconds), about 0.5 second is required to get the reference arrays from one of the workers to the master and to distribute this array to all workers (only red communication lines in the Gantt chart). Then about five seconds are required for the parallel constant normalization. The serial code from the **affy** package is reused at the workers. Parallelized constant normalization is available in the function

```
R> norm <- normalizeAffyBatchConstantPara(Dilution, refindex = 1,
+      FUN = mean)
```

**Invariantset Normalization:** After the `AffyBatch` object's initialization at the workers, a parallel procedure with three steps is started. If a PM and MM separate invariantset normalization is used, then there is a second three step procedure (see Gantt Chart). In the first step the reference array is calculated, then the reference array is copied from one worker to all workers. In the third step the invariantset normalization is executed using the serial code from the **affy** package at the workers. Parallelized invariantset normalization is available in the function

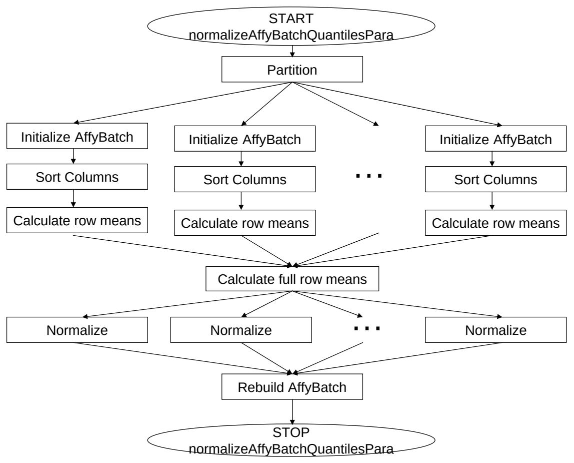
```
R> norm <- normalizeAffyBatchInvariantsetPara(Dilution,
+          baseline.type = "mean", type = "pmonly" )
```

### Complete Data Normalization Methods

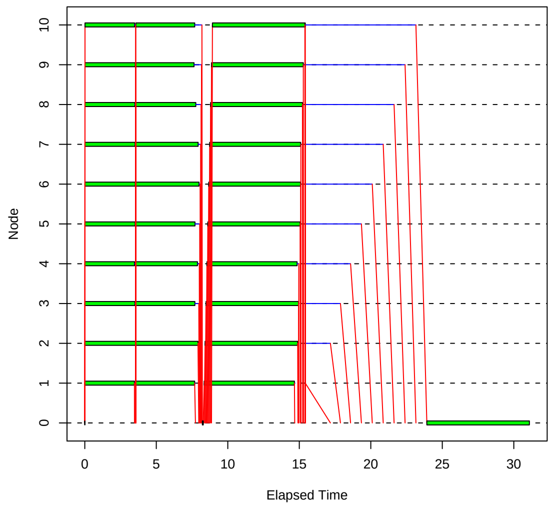
These methods use information from across all arrays to produce the normalization. Therefore, first of all the data have to be distributed to the workers, and some model parameters have to be calculated. Sending these parameters back to the master, the parameters for the whole normalization model can be computed. Back at the workers, the normalization is done with the complete parameters.

**Quantile Normalization:** Quantile normalization gives the same empirical distribution of intensities to each array. First of all the row (probes) means over all arrays have to be calculated. Therefore, the row means are calculated at the workers. With these results the full row means are calculated at the master. The vector of row means is distributed to all workers and the arrays are normalized at the workers. Figure 5.4 shows the flowchart and Gantt chart for the parallelized quantile normalization function. Parallelized quantile normalization is available in the function `normalizeAffyBatchQuantilesPara(object, type = c("separate", "pmonly", "mmonly", "together"),...)`. An example code using a cluster with two MPI workers and the Dilution data set (four arrays) from the **affydata** package has the following structure:

```
R> makeCluster(2, type = "MPI")
```



(a) Flowchart



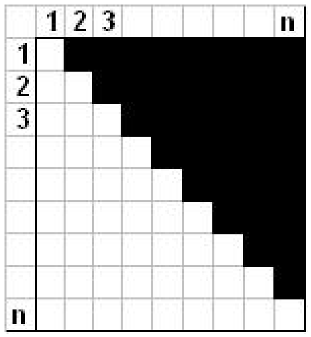
(b) Gantt chart

Figure 5.4: Flowchart and Gantt chart for parallelized quantile normalization with CEL file list as input object.

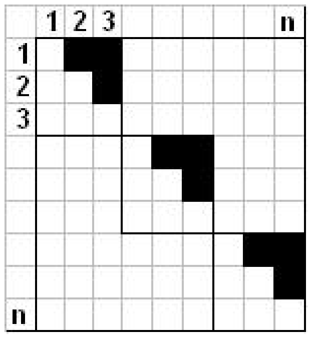
```
R> bgc <- normalizeAffyBatchQuantilesPara(Dilution, type = "pmonly",
+                  verbose = TRUE)

Partition of object 1.406 sec DONE
Object Distribution: 2 2
Initialize AffyBatches at slaves 4.929 sec DONE
PM normalization 13.413 sec DONE
Rebuild AffyBatch 2.765 sec DONE

R> stopCluster()
```

For performance reasons the quantile normalization in the **affyPara** package is implemented in C. Due to the use of the **snow** package as communication API for parallel computing and the support of different communication layers (socket, mpi, pvm, nws) in the **snow** package, it is not possible to use C code for the parallelization. In detail for MPI, it is not possible to export the global MPI-communication object (pointer), which is created in R from the **snow** package, into the C language. A complete reimplementation of the **snow** and **Rmpi** would be required. At the workers it is possible to use C code, but for quantile normalization the benefit would be only small. Therefore, the computation time of the parallel code (implemented in R) has to be compared to the serial code in C.

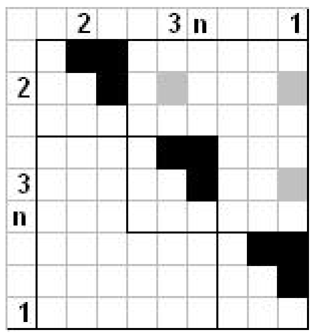
**Cyclic Loess Normalization:** In the cyclic loess normalization for single channel array data, pairs of arrays are normalized to each other by using MA-plots. Usually only one complete cycle through the data is required. Figure 5.5(a) visualizes the $\frac{A(A-1)}{2}$ pairs ($A$ is the number of arrays), which will be normalized.



(a) Cyclic Loess Complete Normalization



(b) Cyclic Loess Normalization at workers



(c) Cyclic Loess Normalization at workers with Permutation

Figure 5.5: Visualization of cyclic loess normalized array pairs for $A$ arrays $(1,\ldots,n)$ and $N$ processors.

The pairwise parallelization for cyclic loess is difficult. Due to the block cyclic distribution of the arrays, at each worker only some pairs are available, visualized in Figure 5.5(b). If the normalization is only executed at the workers, only $\frac{A(ceiling(A/N)-1)}{2}$ pairs are normalized to each other ($N$ is the number of processors) or $\frac{A^2}{2}(1-\frac{1}{N})$ are not normalized to each other. This means, the bigger the number of processors, the less pairs are normalized. Furthermore, there has to be more than one array at each node ($\frac{A}{N} > 1$).

**Cyclic Loess Standard:** Due to the block cyclic distribution of the arrays, first of all the available pairs at the workers are normalized to each other. To get the same results as from the original loess normalization function, the normalization between all arrays has to be guaranteed. Therefore, array one from worker one is distributed to all other workers, normalized against all available pairs and removed from the workers. In the same way all arrays are copied to the workers and normalized to each other. It has to be checked, that arrays are not normalized twice to each other. This approach requires a lot of network traffic, but due to the parallel normalization of the pairs, an obvious acceleration can be achieved.

The parallelized cyclic loess normalization is available in the function

```
R> norm <- normalizeAffyBatchLoessPara(Dilution, type = "pmonly", maxit = 1 )
```

**Cyclic Loess with Permutation:** The parallelized cyclic loess normalization requires a lot of network traffic and array permutations. For further improvements the number of normalized pairs can be adapted and the network traffic is smaller. This idea is visualized in Figure 5.5(c) and implemented in the function

```
R> norm <- normalizeAffyBatchLoessIterPara(object, percentPerm = 0.75,
+                    type=c("separate","pmonly","mmonly","together"), maxit=1)
```

The additional parameter `percentPerm` is required, which gives the percentage of normalized pairs. Applied studies demonstrate, that 75% percent of pairs are sufficient for a 'good' normalization. For the quality comparison of the normalization the graphical tools boxplot and histogram (not shown) are used. Figure 5.6 compares the boxplots for 50 arrays normalized with the serial cyclic loess normalization (red boxplots) and the parallel (four workers) cyclic loess normalization with 75% percent of pairs normalized (blues boxplots).

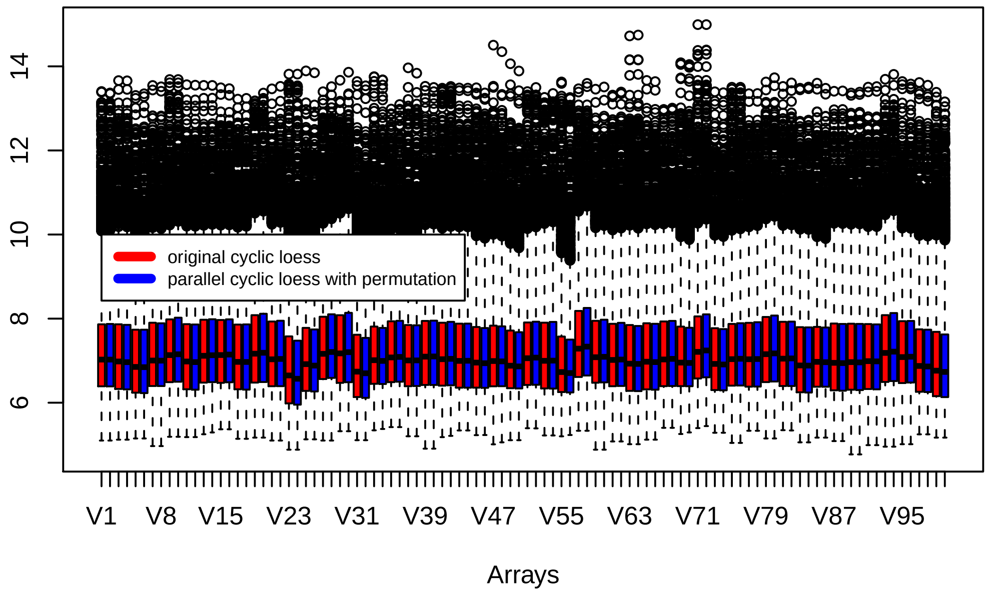


Figure 5.6: Boxplots comparing serial cyclic loess normalization (red boxplots) and the parallel (four workers) cyclic loess normalization with 75% percent of pairs normalized (blues boxplots). Arrays are from the E-GEOD-11121 experiment.

As shown in the Section 5.3 the improvement by the reduced number of normalized pairs yields only in a small improvement in the speedup.

### 5.2.4 Summarization

Summarization is the final step in preprocessing raw data. It combines the multiple probe intensities for each probeset to produce expression values. These values will be stored in the class called `ExpressionSet`. Compared to the `AffyBatch` class, the `ExpressionSet` requires much less main memory, because there are no more multiple data. The different parallel summarization methods are available in the `computeExprSetPara()` function.

#### Single-Chip Summarization

Single-chip summarization methods use only probe intensities of an individual array to compute expression values for that array. Therefore, they are easy to parallelize. The original summarization methods can be executed at the workers, the new intensity matrices (about 10-16 times smaller than the intensity matrices of an `AffyBatch` object) are sent back to the master and the complete `ExpressionSet` object is build. This method works for `avgdiff` and `mas` summarization.

```
R> eset <- computeExprSetPara(Dilution, method = "avgdiff")
```

#### Multi-Chip Model Summarization

Multi-chip model summarization methods build statistical models upon the probe intensities of all arrays. Due to the block cyclic distribution of the arrays, at each worker only some arrays are available. Furthermore, the serial summarization methods are implemented in for-loops over all probes. For the new generations of chips these are more than 500.000 iterations. Therefore, the serial methods are implemented in C. It is not possible to use C code for the parallelization with the **snow** package and the computation time of the parallel code (implemented in R) has to be compared to the serial code in C.

**Original Multi-Chip Model Summarization:** In the parallelized method, the required probes for one probeset are collected from the workers, and by means of the standard summarization methods one expression value is calculated at the master. In the following step the next probes belonging to one probeset are collected from the workers and summarized, and so on. This is a slow parallel implementation for multi-chip model summarization, but produces in view of machine accuracy the same results as the serial methods. This algorithm is available for the summarization methods `playerout`, `farms`, `liwong` and `medianpolish`. In the `computeExprSetPara()` function these methods have to be coded with `method='XXX_orig'`.

```
R> eset <- computeExprSetPara(Dilution, method = "medianpolish_orig")
```

**Partly Multi-Chip Model Summarization:** Similar to the 'cyclic loess normalization with permutation' idea, the multi-chip model summarization can be calculated only at the workers. In this case the $ceiling(\frac{A}{N})(N-1)$ arrays at the other workers do not have any influence on the summarization model. But parallelized summarization is about $N$ times faster than the original multi-chip model summarization. This algorithm is available for the summarization methods `playerout`, `farms`, `liwong` and `medianpolish` and can be called with the following command:

```
R> eset <- computeExprSetPara(Dilution, method = "medianpolish")
```

Applied studies demonstrate, that for more than 10 arrays per worker very similar expression values are generated. For the quality comparison of the summarization the graphical tools boxplot and histogram (not shown) are used. Figure 5.7 compares the boxplots for 50 arrays summarized with the original medianpolish summarization (red boxplots) and the parallel (four workers) medianpolish summarization only at the workers (blues boxplots).

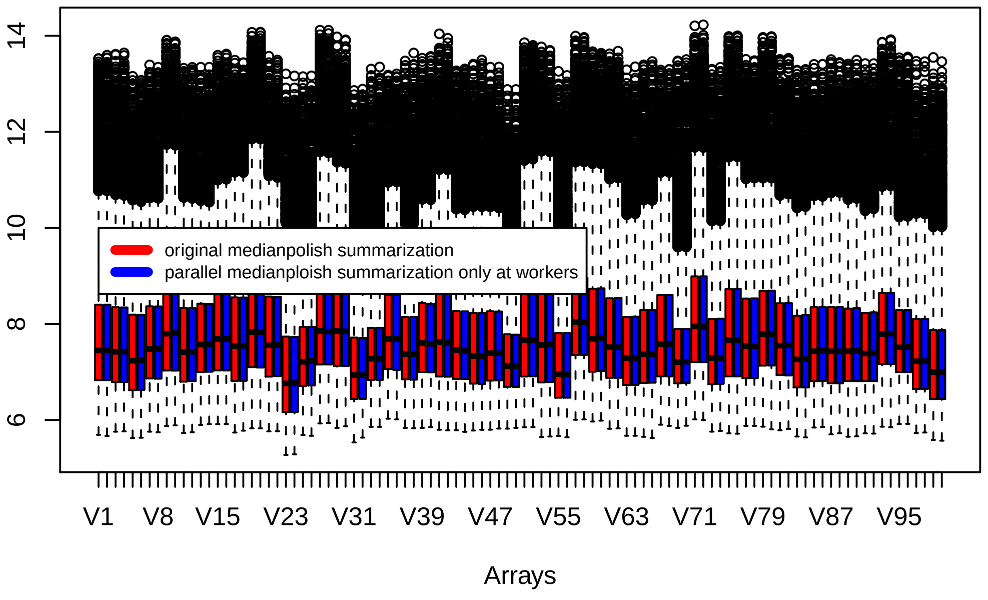


Figure 5.7: Boxplots comparing original medianpolish summarization (red boxplots) and the parallel (four workers) medianploish summarization only at the workers (blues boxplots). Arrays are from the E-GEOD-11121 experiment.

### 5.2.5 Composite Preprocessing

An efficient method for preprocessing can be obtained by combining the background correction, normalization and summarization methods to one single method. For parallelization, the combination has the big advantage of reducing the exchange of data between master and workers. Moreover, at no point a complete `AffyBatch` object has to be built, and the time-consuming rebuilding of the `AffyBatch` object is no longer necessary.

#### preproPara

A parallelized complete preprocessing method is available in the following function:

```
R> eset <- preproPara(Dilution, bgcorrect = TRUE, bgcorrect.method = "rma",
+                     normalize = TRUE, normalize.method = "quantiles",
+                     normalize.param = list(type = "pmonly"),
+                     pmcorrect.method = "pmonly", summary.method = "medianpolish")
```

This function can compute the preprocessing steps proposed for background correction, normalization and summarization. The function was developed according to the `expresso()` function in the **affy** package. Figure 5.8 visualizes the parallel computation time using a Gantt chart. The summarization step takes the most computation time

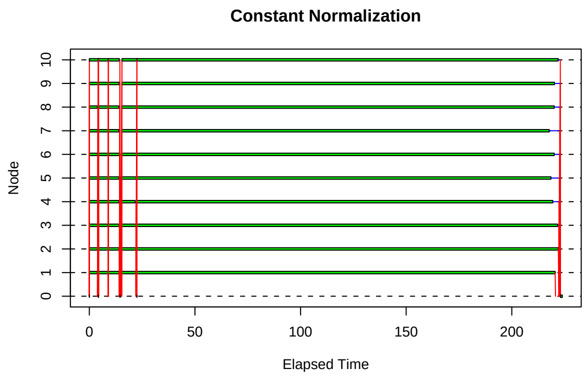


Figure 5.8: Gantt chart for parallelized complete rma preprocessing.

(200 seconds), but there are less communications and there is no expensive `AffyBatch` object rebuilding.

#### rmaPara

One of the most common used complete preprocessing methods is the Robust Multi-Array Average (RMA) expression measure. This function converts an `AffyBatch` object into an `ExpressionSet` object using the robust multi-array average expression measure. This method is parallelized in the function `rmaPara()` and is a wrapper around the `preproPara` function. Due to the discussed problems in parallelization of C code with the **snow** package, the parallel function is not optimized using C code. Furthermore, the parallel function does not use the original `medianpolish` function, the described partly multi-chip model summarization is used. But with the parameter `summary.method="medianpolish_orig"` the original summarization can be used and the same results as in the serial implementation are obtained.

```
R> eset <- rmaPara(Dilution)
```

### vsnPara

Variance stabilization normalization (VSN, [HvHS+02]) combines background correction and normalization into on single procedure and uses the additive-multiplicative error model to estimate the errors in microarray data. Therefore, a complex and computing intensive robust variant of the maximum-likelihood estimator for the stochastic model has to be solved. The parallelization of the maximum-likelihood estimator is complex and ongoing work.

First of all the serial vsn implementation is completely written in the C language. Due to the described problems of parallelization of C code and the use of the `snow` package, the existing C code was translated to R code. In the function `vsn2_optimPara()` the L-BFGS-B [LNZ+94] solver – implemented in the R function `optim()` – requires the function to be minimized and a function to return the gradient. For the evaluation of these functions all arrays are required. Therefore, these functions were reimplemented and adapted for the block cyclic distribution of the arrays. This parallel implementation yields only a small improvement in speed, because the function evaluations and calculations of the gradients are not very time consuming. Furthermore, for every optimization step two parallel communications are required, which produces a lot of network traffic.

This is a first working parallel implementation of the vsn method, which supports the block cyclic data distribution and therefore, the vsn normalization of a nearly unlimited number of arrays. However, the parallel R implementation has the same speed or is slower than the serial C implementation.

```
R> norm <- vsn2Para(Dilution)
```

The parallel vsn method is provided by the `vsn2Para()` function, which additionally supports add-on normalization (parameter `reference`). The commonly used `vsnrmaPara()` function is available for vsn background correction, normalization and rma summarization.

### 5.2.6 Quality Control & Assessment

Quality assessment is an important procedure, that detects divergent measurements beyond the acceptable level of random fluctuations. In most cases different graphical based methods are used to assess the quality of arrays. Using the block cyclic data distribution in the **affyPara** package, the methods have to be adapted to collect the data from the workers and to calculate the plot parameters in parallel. In the bachelor thesis from Esmeralda Vicedo [Vic09] the parallel quality control methods `boxplotPara()` and `MAplotPara()` were implemented. The graphical visualization for more than 150 arrays gets – independent on the used graphical method – very complex and unreadable. Therefore, the parallelized methods were extended for only plotting the interesting (outlier and reference arrays) arrays. Additional a simulation study for the agreement between statistical methods for quality control was started to find the best quality assessment methods. First results show, that the pairs MA-plot - heatmap, spatial density distribution - RLE, and boxplot - NUSE assess the same arrays as outliers. This result can be validated in theory and with realistic examples. Therefore, it can be advised to use only one method from each of these pairs. For example, the combination of the three methods MA-plot, RLE and boxplot is sufficient for the quality assessment of microarray data. A publication with a detailed description of the simulation study, the statistical methods for comparing the quality assessment methods, and the results is in preparation.

For more details on the functions, see the help files or the vignette in the **affyPara** package. An overview of all functions in the **affyPara** package is available in the vignette, too.

## 5.3 Results

The **affyPara** package with parallelized and efficient preprocessing methods for high-density oligonucleotide microarrays was developed. Parallelization of existing preprocessing methods produces, in view of machine accuracy, the same results as serialized methods and new methods in parallel code arose. The partition of data and distribution to several nodes solves the main memory problems and accelerates the methods up to factor 15 for 300 microarrays or more.

The package is open-source, available in the Bioconductor repository since April 2008 (`http://www.bioconductor.org/packages/release/bioc/`), and is accepted and used by the community. Figure 5.9 visualizes the download statistic for the **affyPara** package provided by the Bioconductor Team. The package supports all commonly used functions (in parallel implementation) for microarray preprocessing. An overview of the functions is available in the vignette of the package.

This section outlines a critical discussion for the implementation and results, and compares the package to other solutions.

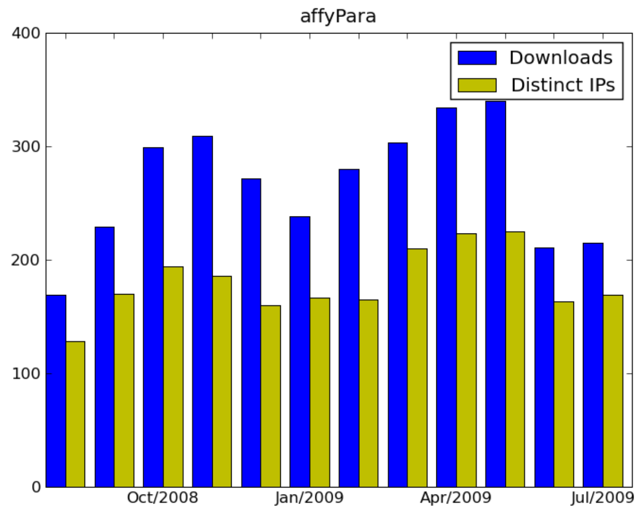


Figure 5.9: Download stats for software package **affyPara** (`http://bioconductor.org/packages/stats/bioc/affyPara.html`).

### 5.3.1 Partition

Main problems for microarray preprocessing are the computer hardware limitations. Especially the available main memory limits the number of arrays that may be quantified. Therefore, mainly domain decomposition is used in the **affyPara** package. For parallelization with domain decomposition, the input data have to be partitioned and the parts of input data have to be distributed to the workers. The easiest and most natural way is a block cyclic distribution. In this process the input data will be partitioned on arrays (columns) and distributed equally to all nodes. This process is visualized in Figure 5.1.

Partition on probes and other partition strategies have not been tested or implemented. In this context, a new data structure in the R programming language has to be developed, all preprocessing functions have to be reimplemented, and the reuse of existing code would not be possible. Due to the array oriented implementation of the existing methods the chosen partition is the best one. Only for summarization (summarizing rows) the partition on probes (rows) should improve the performance. However, redistribution of the data costs a lot of communication time (compare discussion for loess normalization) and should, therefore, be rejected.

#### Avoid Complete `AffyBatch` Object

Partitioning the `AffyBatch` object at the master and distributing split `AffyBatch` objects to the workers does not solve the problem of limited main memory. The complete `AffyBatch` object has to be built at the master and the distribution creates a lot of network traffic. Therefore, the processes start at the workers with a certain delay (see Figure 5.10).

It is more efficient to partition the vector of CEL files and to create the split `AffyBatch` objects at the workers. In the **affyPara** package all functions for preprocessing can get a partitioned list of CEL files, a list of CEL files (list will be splitted at the master and `AffyBatch` objects created at the workers) or an `AffyBatch` object as input data.

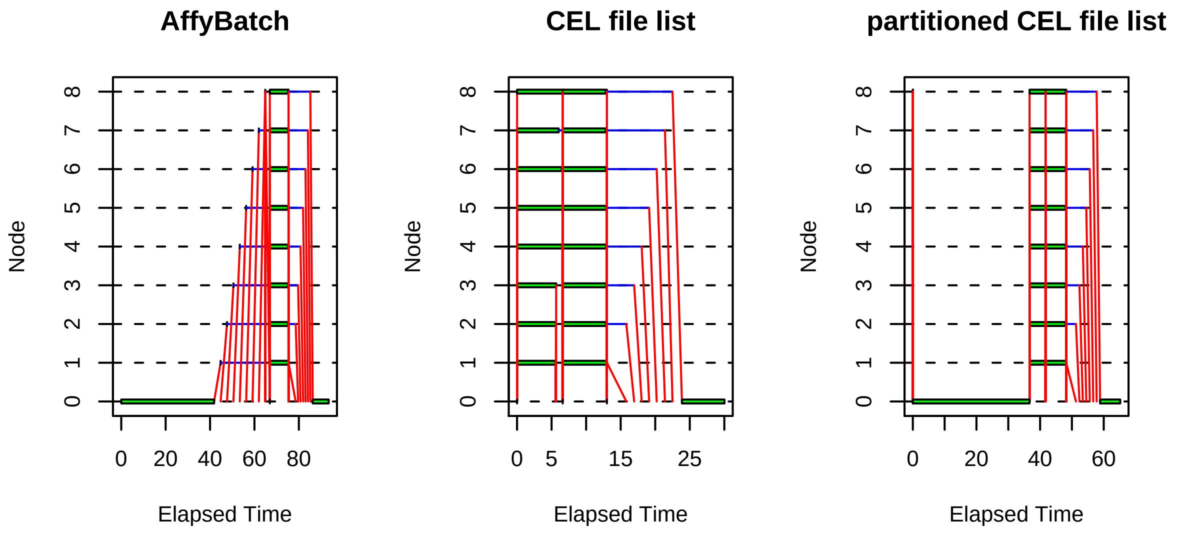


Figure 5.10: Gantt chart for parallelized background correction (`rma`) methods with an `AffyBatch` object, a CEL file list and a partitioned list of CEL files as input data. No worker process runs at the master processor.

In Figure 5.10 the computation and communication time of background correction (`rma`) for 120 microarrays using eight workers and different input objects is visualized with Gantt charts. For the `AffyBatch` input object there is a long computation time at the master to create the `AffyBatch` object (40 seconds) and then a long time for sending the `AffyBatch` objects to the workers and storing them (25 seconds). These two operations take more than 60% of the complete computation time. After these operations, a short time (about 10 seconds) is required for the background correction. At the end all data (`AffyBatch` objects) have to be collected from the workers and the complete `AffyBatch` object has to be rebuilt at the master. Collecting the data and rebuilding the result object is required independent from the input object. For the 'CEL file list' input object there is less time required for distributing data, but there is additional time required for creating the small `AffyBatch` objects at the workers (6 seconds). Using a 'partitioned CEL file list' as input object, first of all the raw data (CEL files) have to be distributed from the master to the workers. In this example it takes up to 40 seconds using the 'RCP' protocol. This process runs on the operation system level and is not monitored from the **snow** package in the R language. Therefore, only computation time at the master node is plotted. In reality this is communication time between master and workers. After the distribution process, some time is required to build the small `AffyBatch` objects at the workers (6 seconds).

In parallel environments the CEL files are often available by a shared memory system. At a workstation cluster, this is often realized with a samba or nfs device, in multi-core systems every processor has a connection to the hard drive. In computer clusters the shared memory system can be the bottle neck for communication traffic. In this case and for distributed memory systems, the function `distributeFiles()` for (hierarchically) distributing files from the master to a special directory (e.g., '/tmp/') at all workers was designed. R or the faster network protocols 'SCP' or 'RCP' can be used for the process of distributing data. As demonstrated in Figure 5.10 the file distribution requires additional time.

Accessing the raw data from a shared memory system and building the small `AffyBatch` objects at the workers is the fastest solution. No problems with the connection to the shared memory system could be detected at the tested computer environments.

Due to the required communication costs for small numbers of arrays the communication time is longer than the calculation time (compare Figure 5.10). Therefore, for small numbers of arrays (less 100) the serial code is faster than the parallel code.

### Equal Data Distribution

In general the size of raw data per worker is essential for the performance of the parallelized methods. Especially using data decomposition, the raw data have to be distributed in a suitable manner to achieve a good load balancing. In our case we assume, that every processor or worker has the same performance and due to the same chip type of all arrays the calculation time per array will be the same at every worker. Therefore, the arrays should be distributed equally to all workers. All functions in the **affyPara** package distribute the arrays equally to all nodes as far as possible.

### Size of Partition and Number of Processors

As described in the previous section at the beginning and in the end of the preprocessing process time for the communication will be required. The **affyPara** package uses the **snow** package as parallel computing API. In the **snow** package the communication for large data objects is not yet optimized, because there are supported communication mechanism (MPI, PVM, NWS, SOCKET) for which no optimized broadcast functions exist. In the **snow** package there is a linear data send operation: object one is sent to worker one, when the first send operation is completed, object two is sent to worker two, and so on. Therefore, the communication time grows linearly using the **snow** package (compare the Gantt chart for the `AffyBatch` object in Figure 5.10). Using optimized communication functions like 'MPI_Isend' or 'MPI_broadcast' from the MPI library logarithmic communication time can be achieved. Assuming that the calculation time typically behaves like $1/N$ (N number of processors) the complete time can be calculated.

Figure 5.11 plots the assumed calculation time (red), possible communication times (green, blue), and the cumulative time (dashed). For the **snow** package there is a linear communication time (blue) and therefore, after a special number of processors the cumu-

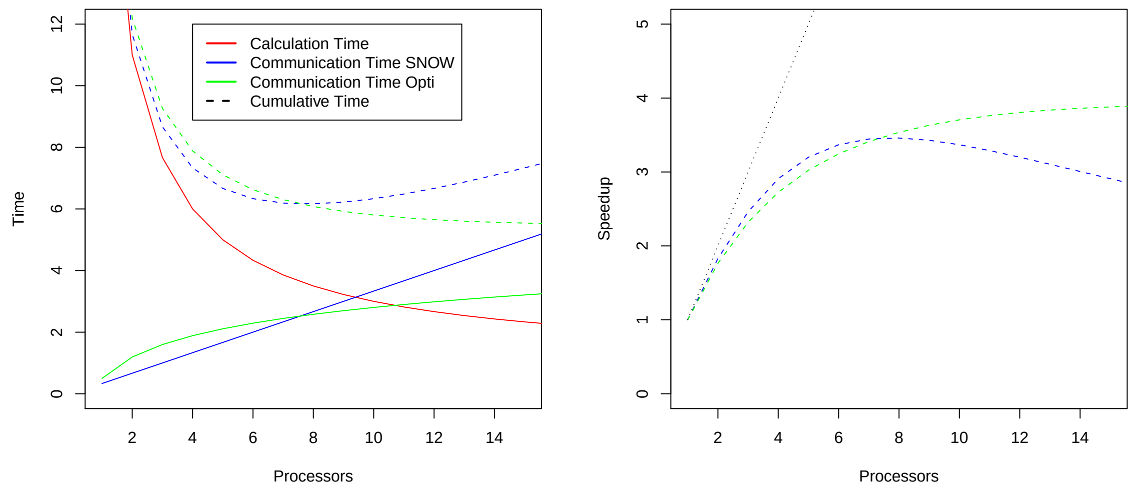


Figure 5.11: Theoretical communication and calculation time and corresponding speedup curves.

lative time starts to grow. The same behavior can be seen in the speedup curves, due to the linear communication time the speedup (dashed blue line in right figure) is decreasing. This behavior can be seen in all speedup curves for the **affyPara** package and is discussed in theory with Amdahl's law in Section 4.5, too.

Using optimized communication functions the optimal number of processors is as much processors as possible. Due to the communication costs generated from the **snow** package the optimal number of processors is not infinite. Therefore, it is very difficult to choose the right size and number of partitions. Simulations (compare speedup curves in this chapter) at the different available hardware environments show, that the best performance can be achieved for 15-20 arrays per node or as much nodes as possible, if there are too many arrays. The optimal number of arrays per node or number of processors can be defined as follows:

$$\text{arrays per node} \approx 20 \tag{5.1}$$

$$\text{number of processors} \approx \frac{\#\ \text{Arrays}}{20} \tag{5.2}$$

## 5.3.2 Performance Analysis

Different tools for the performance analysis are presented in Chapter 4.5. This section discusses the performance of the **affyPara** package at different parallel computing environments using the presented tools.

### Overview

Table 5.1 lists different performance metrics for all developed parallel preprocessing methods. For the performance analysis 300 arrays of the chip type 'HG-U133A' (data from the large cancer study in Chapter 7) and the computer cluster at the IBE were used. For all

| | T seriell | T parallel | N | S abs | S rel | Efficiency | Karp-Flatt |
|---|---|---|---|---|---|---|---|
| rma BGC | 194.4 | 62.86 | 10 | 3.09 | 3.79 | 0.38 | 0.18 |
| constant Norm. | 700.0 | 68.13 | 14 | 10.27 | 14.68 | 1.05 | 0.00 |
| invariant Norm. | 2262.7 | 121.81 | 16 | 18.58 | 15.86 | 0.99 | 0.00 |
| quantil Norm. | 1866.7 | 75.72 | 14 | 24.65 | 6.03 | 0.43 | 0.10 |
| loess Norm. | 84293.3 | 10877.31 | 20 | 7.75 | 7.65 | 0.38 | 0.08 |
| loessIter Norm. | 84293.3 | 9078.84 | 10 | 9.28 | 8.81 | 0.88 | 0.01 |
| VSN | 1065.8 | 849.04 | 12 | 1.26 | 2.70 | 0.22 | 0.31 |
| Summarization | 386.1 | 420.44 | 6 | 0.92 | 1.14 | 0.19 | 0.85 |
| Summ. part | 407.1 | 50.63 | 30 | 8.04 | 7.20 | 0.24 | 0.11 |
| RMA (summ. part.) | 273.6 | 232.62 | 30 | 1.18 | 8.75 | 0.29 | 0.08 |
| readAffyBatch | 64.8 | 46.38 | 8 | 1.40 | 2.62 | 0.33 | 0.29 |

Table 5.1: Overview of the performance of the **affyPara** package: Serial computation time in seconds, parallel computation time in seconds, optimal number of processors (N), absolute speedup, relative speedup, efficiency, and Karp-Flatt Metric. Calculated with 300 microarrays at the computer cluster at the IBE.

functions the default parameters were chosen. To reduce computation time for the invariantset, quantile, cyclic loess normalization and the medianpolish summarization only the PM intensity data were used.

All baseline methods – all background correction methods, constant and invariantset normalization, and `avgdiff` and `mas` summarization – are parallelized very easy and efficient. But some of these methods have very short serial computation times (e.g., rma background correction) and the communication traffic in the parallel implementation limits the acceleration.

All other methods, multi-chip or complete data methods, are more difficult to parallelize, and require more network traffic. If the proportion of communication to calculation time is small (calculation >> communication time), then the methods perform very well in parallel. The Karp-Flatt metric is a measure for the parallelization of code and the lower the value the better the parallelization. Except the complete summarization, VSN, and readAffyBatch all methods show a good level of parallelization and an acceptable efficiency. The VSN parallelization has a medium Karp-Flatt metric, because only the function and gradient evaluation of the solver are parallelized and the serial C code is compared with the parallel `R` code. In the `medianpolish` summarization a loop over all probe sets, collecting the probe intensities from all workers and summarizing them, with a lot of network traffic is required. `read.affybatchPara()` is a parallel function to read the CEL files into an `AffyBatch` object. This function is only useful at a multi-core machine, where no

”real” network traffic is required. Therefore, this function was benchmarked at the IBE - lis06 multi-core machine. Due to the complex rebuilding process for the whole `AffyBatch` object, there is only a low absolute speedup and a bad Karp-Flatt value.

For the normalization methods a relative speedup up to 15 and an absolute speedup up to 24 can be achieved. Several speedup curves are plotted in Figure 5.12 and for all methods the parallel code is faster than the serial code. This even holds for parallel R code compared to serial C code. For quantile normalization the ”old” (inefficient) R code was used, therefore, there is a super-linear absolute speedup.

As described, due to the communication costs for about 20 arrays per processor the best performance is achieved. In this case for 300 arrays $N = 15$ processors are required. The performance analysis verifies this number, but there are differences in the used methods and hardware environment. Therefore, it is not possible to give a more detailed rule for the optimal number of processors.

For the partly multi-chip model summarization the best results can be achieved with as much processors as possible, because the summarization then runs in parallel and the communication costs from the `ExpressionSet` objects are less (10-16 times) than from the `AffyBatch` object. But having less than 10 arrays per worker, the expression intensities are very different to the complete summarization results. In default, the function `rmaPara()` uses the partly summarization and therefore, as much processors as possible, but not less than 10 arrays per processors, should be used.

### Different Computer Environments

The **affyPara** package is tested in different hardware environments:

**Computer Cluster:** IBE, Linux-Cluster (LRZ, Munich, Germany), HLRB2 (LRZ, Munich, Germany), Hoppy (FHCRC, Seattle, WA, USA)

**Multicore:** IBE, HLRB2 (LRZ, Munich, Germany), lamprey (FHCRC, Seattle, WA, USA)

Thanks a lot to the listed institutes for providing access to their computer resources. The package works at all systems and the performance was very similar. There were comparable speedup values and ideal numbers of processors. The main differences occurred due to hardware differences (CPU speed) of the systems. Figure 5.12 shows the relative speedup curves for background correction and quantile normalization at the IBE (2.66 GHz) and the HLRB2 (1.6 GHz) cluster (dashed line). Breaks in the speedup curves are mostly generated by unbalanced data distribution. For example 200 microarrays cannot be equally distributed to 23 nodes, there are some processors that have to calculate with one more array. Furthermore, external network traffic in the workstation cluster at the IBE is a reason for outliers. To avoid these effects for all time measurements the calculations were repeated five times and averaged. Due to the described linear computation time in the **snow** package, the speedup curves are decreasing after the optimal number of processors.

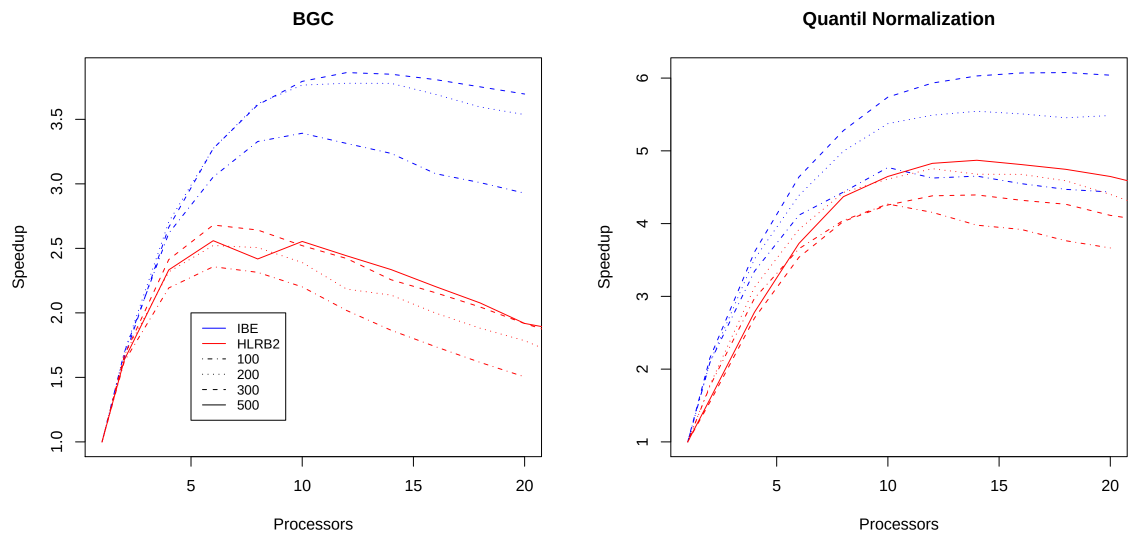


Figure 5.12: Speedup curves for background correction and quantile normalization at the IBE (blue) and the HLRB2 cluster (red).

### 5.3.3 Comparison to other Solutions

The presented **affyPara** package is not the only solution to preprocess a huge number of microarray data. There are optimized functions in some Bioconductor packages. For example in the **affy** package, there is the function `justRMA()` which reads CEL files directly in the working directory and converts the raw data - without using an `AffyBatch` object - to an expression measure using robust multi-array average. In addition, there is the **aroma.affymetrix** package which uses efficient data structure approaches for reading data from a file on request instead of loading the file to the main memory [Ben04]. These three solutions are compared in Table 5.2.

All packages use the R language for the basic operations. The **affyPara** package requires additionally an API for parallel computing (e.g., MPI) and other solutions in the Bioconductor packages use the C language to accelerate the methods. The **aroma.affymetrix** package uses the file system for data handling and is, therefore, the slowest implementation in computation time. The **affyPara** package uses the main memory of several computers and is, therefore, the fastest solution. A simple time comparison of rma background correction and quantile normalization is available in Figure 5.13. For small numbers of arrays the original **affy** code is the fastest solution. Using more than 50 arrays the **affyPara** package is the fastest solution. For 200 arrays the **affyPara** package is up to 13 times faster than the **aroma.affymetrix** package and twice as fast as the code from the **affy** package.

All three solutions are open-source, they work on the standard operation systems and there exist possibilities for third-party extensions. Compared to the other solutions, the **affyPara** package only works for expression arrays. The described parallelization ideas can be extended to other genomic data as SNP chips or exon arrays. For all three solutions

| | **affyPara** | **aroma.affymetrix** | Affymetrix packages in Bioconductor |
|---|---|---|---|
| Programming language | R + MPI | R | R (+ C) |
| Depends on | **affy**, **snow** | **aroma.core**, **affx-parser**, **aroma.light**, **R.huge**, **aroma.apd** | **Biobase** |
| Data handling | Memory | File System | Memory |
| Chip types | expression arrays | expression arrays, SNP chips, exon arrays, ... | expression arrays, SNP chips, exon arrays, ... |
| Usability | easy | difficult | easy |
| Max. number of arrays | limited by number of computers | limited by size of hard disk | limited by main memory |
| Computation time | fast | slow | medium |
| Operating system | Linux, Windows, Mac OS X | | |
| Third-party extensions | yes | | |
| Open-source | yes | | |

Table 5.2: Comparison of the **affyPara** package to the **aroma.affymetrix** package and other solutions.

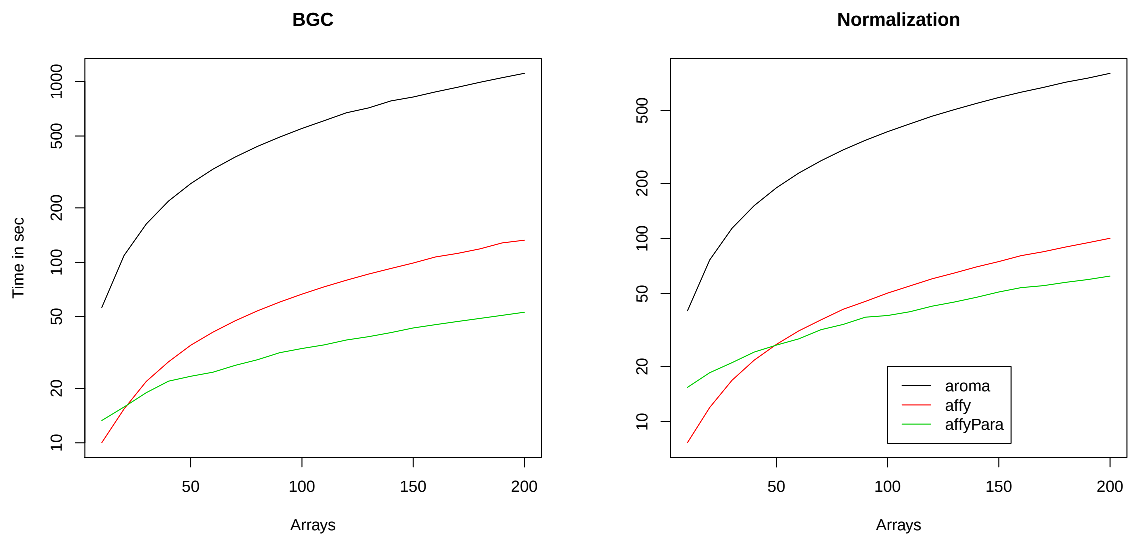


Figure 5.13: Computation time for rma background correction and quantile normalization using code from the **affy**, **affyPara** and **aroma.affymetrix** package.

there is a limitation in the maximum number of preprocessible microarrays. Due to the size of existing supercomputers and grid environments with the **affyPara** package, the biggest amount of microarray data should be preprocessible. Using the cluster pool at the IBE about 16.000 (32 nodes · approximately 500 microarrays) microarrays of the type HG-U133A can be preprocessed using the function `preproPara()`.

The usability of the **affyPara** package is very good for trained R and Bioconductor users. Some simple code examples for creating an `AffyBatch` object and rma background correction with the **affy** and **affyPara** package are visualized in the following lines:

```
R> library(affy)
R> AB <- ReadAffy()
R> AB_bgc <- bg.correct(AB, method = "rma")
```

```
R> library(affyPara)
R> makeCluster(5, tpe = "MPI")
R> AB <- ReadAffy()
R> AB_bgc <- bgCorrectPara(AB, method = "rma")
R> stopCluster()
```

To use the power of parallel computing, the user needs only a working computer cluster and cluster start or administration programs (e.g., SGE). The R syntax is very similar and only two more lines for starting and stopping the cluster are required. For the **aroma.affymetrix** package a complex data structure at hard disk level is required and the commands are very different to other Bioconductor packages.

```
R> library(aroma.affymetrix)
R> cdf <- AffymetrixCdfFile$fromChipType("HG-U133A")
R> cs <- AffymetrixCelSet$fromName(name, tags, chipType = cdf)
R> bc <- RmaBackgroundCorrection(cs)
R> csBC <- process(bc)
R> AB_bgc <- extractAffyBatch(csBC)
```

## 5.4 Summary

For preprocessing of high-density oligonucleotide microarrays the **affyPara** package [SM08, SM09] based on the **snow** package was developed and is available at the Biodoncuctor repository: `http://www.bioconductor.org/packages/release/bioc` Existing statistical algorithms and data structures had to be adjusted and reformulated for parallel computing. Using the parallel infrastructure the methods could be enhanced and new methods are available. Parallelization of existing preprocessing methods produce, in view of machine accuracy, the same results as serialized methods. The partition of data and distribution to several nodes solves the main memory problems and accelerates the method up to factor 15 for 300 arrays or more. Due to communication limitations in the **snow** package good performance can be achieved calculating with about 20 arrays per node. For the complete rma preprocessing as much processors as possible can be used, but there should be not less than 10 arrays per processor.

A new limitation will be imposed by the memory size of the `ExpressionSet` class and the analysis will be performed on the new big size of expression sets. However, no data set is currently available on the market that would cause main memory problems with the `ExpressionSet` class.

# Chapter 6

# Parallel Computing in Next-Generation Sequence Data

This chapter demonstrates the power and problems using parallel computing to solve the mentioned – see Chapter 3.3 – problems and new challenges for next-generation sequence data. Especially the amount of data, data handling in parallel computing environments and communication costs are very critical in the presented examples.

Section 6.1 outlines general ideas for parallel computing in high-throughput sequencing. An existing parallel implementation for data I/O in the **ShortRead** package from Martin Morgan (FHCRC, Seattle, WA, USA) is presented. Section 6.3 discusses a parallel test implementation in the **BSgenome** package. The `bsapply()` function, which applies a function to each chromosome in a genome, was parallelized using the **Rmpi** and/or **snow** package. Additional standard optimization strategies for parallel computing are described, applied and tested.

## 6.1 Ideas

Due to the `apply`-like code structure of existing data analyses methods, the first idea for parallel computing in next-generation sequence data is the use of data decomposition. Depending on the kind of application and used sequencing technique or machine, there will be different ways of data partition, distribution and implementation. This chapter examines data decomposition for next-generation sequence data and will use the **snow** and/or **Rmpi** package for parallel computing.

Parallel computing ideas from other research areas can be adapted to improve the methods especially for pattern matching and alignment. For example [Mut00] presents a simple and parallel algorithm to solve the multi-pattern matching problem with optimal speedup. The implementation uses a Monte-Carlo algorithm based on finger-print functions. Furthermore, parallel implementations exist in literature for pair-wise alignment using the Needleman-Wunsch algorithm [Bar02] or for local alignment using the Smith-Watermann algorithm [RS00]. These methods are used in serial from the **Biostrings** package, but are not yet implemented in parallel in the Bioconductor project or used in next-generation sequence analyses.

Optimized parallel computing paradigm for huge numbers of global variables exists. NetWorkSpaces and Sleigh (R package **NWS**) use a central server to store all data. This could be used to have the sequences available for every worker and data communication is only required for the requested sequences. This solution is not yet tested for next-generation sequence data. In addition first ideas using graphical processing units for high-throughput sequence alignment are published [STDV07]. For example at a single computer workstation the processing speed of GPUs will be used to improve pattern matching. As well there is no integration to the R language available at the moment.

## 6.2 Parallelization in the ShortRead Package

The **ShortRead** package provides base classes, functions, and methods for representation of high-throughput, short-read sequencing data. The **ShortRead** package aims to provide key functionality for input, quality assurance, and basic manipulation of 'short read' DNA sequences such as those produced by Solexa 454, and related technologies, including flexible import of common short read data formats.

Solexa and other short read technologies often include many files. For example, there is one file per tile, 300 tiles per lane, and 8 lanes per flow cell: 2400 files per flow cell. An example request on the data could be to find the average intensity per base at special cycle. A natural way to extract this kind of information from these files is to write short functions. The files are generally large and numerous, so even simple calculations consume significant computational resources. The `srapply()` function in the **ShortRead** package is meant to provide a transparent way to perform calculations like this, distributed over multiple nodes of a MPI cluster. The package uses the **Rmpi** package and therefore, MPI as communication interface. If no **Rmpi** cluster is available, the function `srapply()` evaluates the function as `lapply()`, whereas the following code distributes the calculation over the available workers.

```
R> library("ShortRead")
R> library("Rmpi")
R> mpi.spawn.Rslaves(nsl = 16)
R> srres <- srapply(intFls, calcInt, cycle = 12)
R> mpi.close.Rslaves()
```

In most cases communication costs for sending as well as receiving data are low and results can be reduced at the workers. Therefore, in most examples using the `srapply()` function a speedup approximately proportional to the number of available computer processors can be achieved. However, the file hierarchy has to be available at the hard disk of every node, e.g., provided by a samba or nfs device.

Furthermore, for quality assessment of next-generation sequence data, the `qa()` function in the **ShortRead** package provides a convenient way to summarize read and alignment quality. Evaluating quality assessment for a single lane can take several minutes, due to the mentioned amount of files. If a **Rmpi** cluster is available, the function distributes the task of processing each lane to each of the workers. Again the file hierarchy has to be available at the hard disk of every worker. Using this parallel approach in a multi-core environment with eight processors, all lanes (8) of Solexa sequence data can be read at the same time and there is an acceleration of factor eight.

## 6.3 Parallelization in the BSgenome Package

The **BSgenome** package provides an infrastructure for Biostrings-based genome data packages. In addition there are some functions to manipulate and process the whole genome data. The `bsapply()` function applies a function to each chromosome in a genome. In this case a parallel implementation can distribute every chromosome of a genome to one worker.

### 6.3.1 Parallel Implementation

The `bsapply()` function is very easy to parallelize. In general `apply` like functions call the same function to different data. The data can be distributed to different R sessions – running on different processors – and the function can be applied in parallel.

As implemented in the `srapply()` function in the **ShortRead** package, the **Rmpi** package will be used. If a cluster is available, the function `mpi.parSapply()` will be used, otherwise the serial `sapply()`. Therefore, some adjustments to the internal function `processSeqname()` – executes the `FUN` function to each chromosome – are required:

- The function has to become a global function to avoid sending the whole environment to each node (due to lexical scoping).

- The function now gets the additional input parameter `BSParams`, to have the data from the object (sequence, FUN, ...) available at every node. In the serial implementation the `BSParams` object was available as global object. The sequences will be loaded (cached) from the disk at the workers. Therefore, the genome library with the `Biostrings` object has to be available at all nodes, but not loaded into the R session.

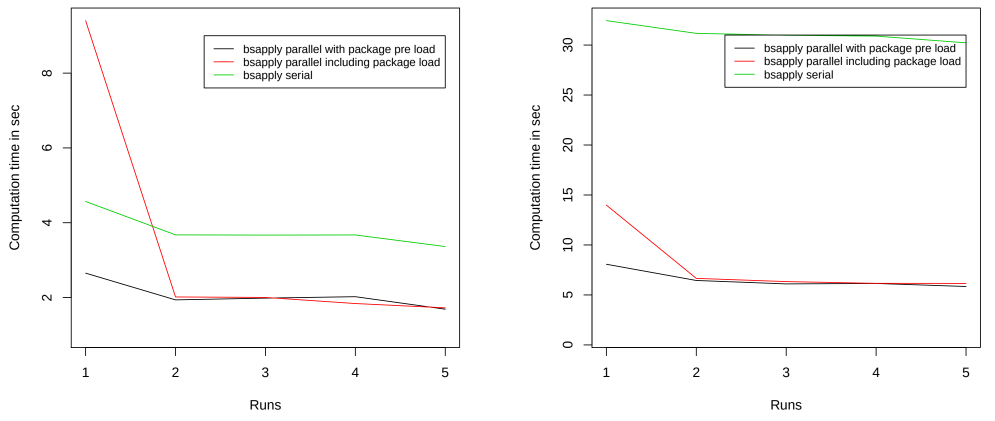
Additional there is a new function `parSFapply()`, which executes the `apply`-like function in parallel or serial. This function is similar to the `srapply()` function in the **ShortRead** package, but with some improvements for error handling.

### 6.3.2 Results & Discussion

The parallel implementation was tested with several examples: In the worm (`BSgenome.Celegans.UCSC.ce2`), and full (all chromosomes - 48) and reduced (only short chromosomes - 38) human (`BSgenome.Hsapiens.UCSC.hg18`) genome the alphabet frequencies for every chromosome and the occurrences of all defined patterns across the whole genome were counted. As expected the serial and parallel code produces the same results in all tested examples.

#### Package and Sequence Loading Times

Using the function `bsapply()` in parallel mode requires the loading of the **BSgenome** (including **IRanges** and **Biostrings** packages) package and the loading (caching) of the sequences from the disk at all workers. For small examples this requires a lot of additional time and the serial method will be the faster one, but in big examples the loading time is irrelevant (see Figure 6.1). In the examples all calculations were replicated five times (runs).



(a) Reduced human genome (38 chromosomes) (b) Full human genome (48 chromosomes)

Figure 6.1: Computation time in parallel (red), in parallel with package preloading (black) and serial (green) for counting the alphabetic frequencies in the full and reduced human genome.

After the first run, all libraries and data structures are initialized and the computation time gets stable. For further time comparisons the packages will be preloaded at all nodes and therefore, the package loading time is not more measured.

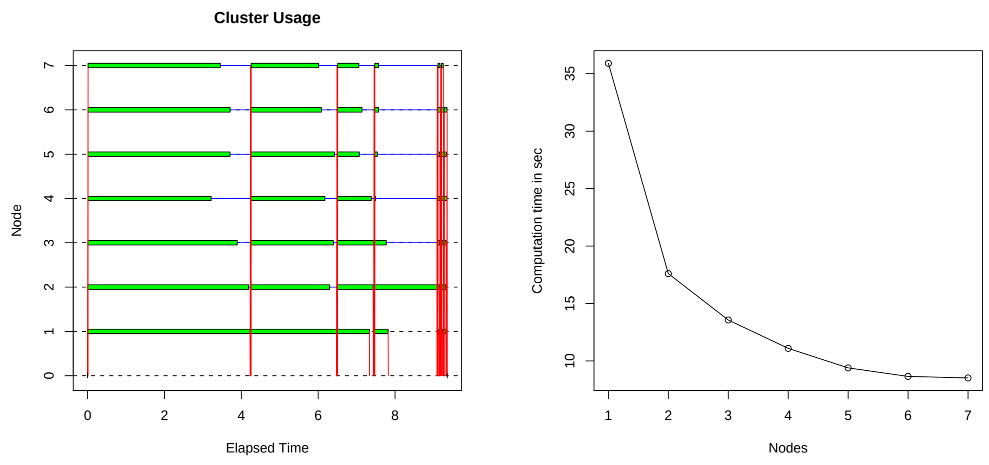
### Visualization of Computation and Communication Times

For the parallelization the **snow** package with MPI as communication layer can be used, too. This library is much more user friendly than the **Rmpi** package. For example there is a function `snow.time()` to visualize the computation and network communication time in a Gantt-Chart. But for huge applications the performance and functionality of this package is not as good as the **Rmpi** package [SME+09].

Figure 6.2(a) shows the Gantt-Chart for measuring the alphabetic frequencies for the full human genome (48 chromosomes) using seven processors. There are more chromosomes than workers, therefore, every worker has to calculate the alphabet frequencies for seven chromosomes. The fist chromosomes are the longest ones (see Figure 6.4(a)), therefore, they have the longest calculation times.

### Number of Processors

Using more processors should show a decrease in computation time. Due to a lot of network traffic (data to send), there is no perfect linear speedup. For most examples using more processors generates smaller computation times (see Figure 6.2(b)).



(a) Gantt-Chart for counting the alphabetic frequencies in the full human genome using seven workers.

(b) Computation time for counting the alphabetic frequencies in the full human genome, calculated at different numbers of processors.

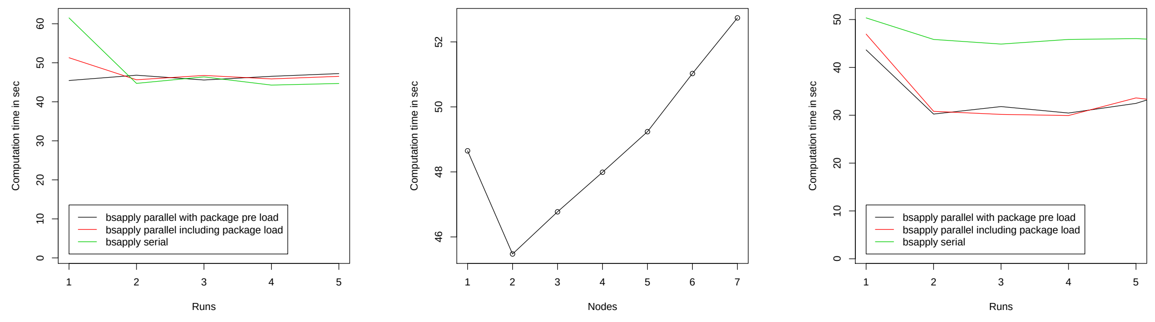
Figure 6.2: Visualization of computation time for counting alphabetic frequencies on full human genome.

### Additional Data Distribution - Pattern Matching

Applying pattern matching on the whole genome requires the sending of additional data – the pattern – to the nodes. For example with the function `countPDict()` the number of occurrences for each pattern can be calculated.

```
R> library(BSgenome.Hsapiens.UCSC.hg18)
R> params <- new("BSParams", X = Hsapiens, FUN = countPDict)
R> library(hgu133plus2probe)
R> dict0 <- DNAStringSet(hgu133plus2probe$sequence)
R> pdict0 <- PDict(dict0)
R> library(Rmpi)
R> mpi.spawn.Rslaves(nslaves = 4)
R> result <- bsapply(params, pdict = pdict0)
R> mpi.close.Rslaves()
```

In detail, the Biostrings object pdict0 has to be sent to every worker. This costs additional time and the parallel implementation is not faster than the serial method (see Figure 6.3(a)). Plotting the computation time over the nodes shows a reduced computation time for two nodes, but then using more processors an increase in the computation time (see Figure 6.3(b)). This is a typical behavior of parallel computing with a lot of (too much) network traffic.



(a) Calculation replicated five times.

(b) Computation time for different numbers of nodes.

(c) Calculation replicated five times and pattern created at worker.

Figure 6.3: Computation time for counting patterns in the reduced human genome.

In most cases this problem can be solved creating the parameters – here the pdict0 object – at the worker. A new function countPDictSF() has to be defined and the pdict0 object will be created at the workers.

```
R> countPDictSF <- function(...) {
+      library(hgu133plus2probe)
+      dict0 <- DNAStringSet(hgu133plus2probe$sequence)
+      pdict0 <- PDict(dict0)
+      return(countPDict(..., pdict = pdict0))
+  }
R> params <- new("BSParams", X = Hsapiens, FUN = countPDictSF, simplify = TRUE)
R> result <- bsapply(params)
```

Therefore, no more data sending is required and the parallel implementation is faster than the serial one (see Figure 6.3(c)). However, the probe sequence data library or the pattern has to be available at every node.

#### Parallel Optimization

There are some further aspects for optimization and performance improvement.

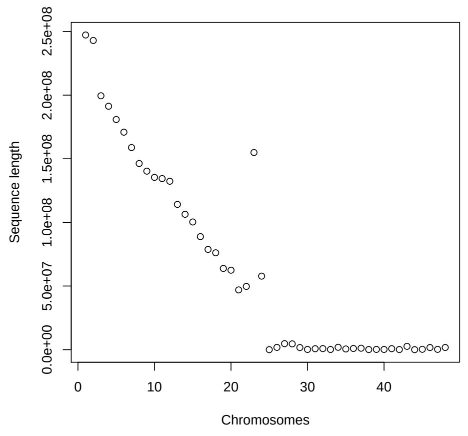
**Sequence Length:** In the `BSParams` class the sequences are stored in the `X` slot (`BSgenome` object). There the sequences are sorted by length (see Figure 6.4(a)). For parallelization the data have to be splitted in $N$ groups with $N$ the number of nodes. For sequence or genome data the first node gets the longest sequences, the second node the second longest one and so on. Therefore, the first worker has the longest computation time and the last node has the shortest sequences and the fastest computation time. This causes a bad load balancing, but can be improved by using a load balanced `apply`-like function and by sending only one sequence to each node. If we have more sequences than nodes, sequences 'wait' at the master node. If the computation at one node is finished, the next sequence will be processed. A computation time improvement of factor two can be achieved (see Figure 6.4(b)).

### 6.3.3 Conclusion

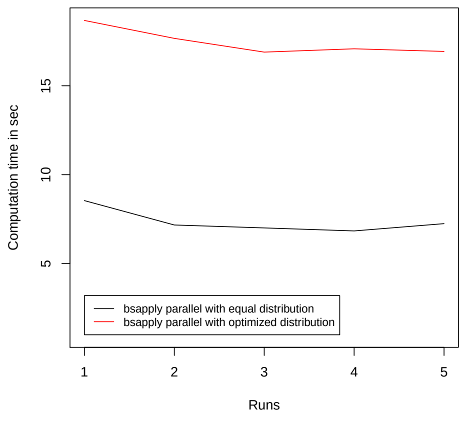
Parallelization of the `bsapply()` or more general of `apply()` like functions is very simple and achieves good performance improvements in most cases. A speedup approximately proportional to the number of available nodes is achievable. But if there is a lot of data to distribute and the communication times are large enough to the relative computation times, no improvement in the overall computation time can be achieved. NetWorkSpaces and the parallel Sleigh environment using a central server to store all data could be a working solution for parallel implementations in next-generation sequence data, but is not yet tested.

For the parallelization of the `bsapply()` function the communication costs are high and only a speedup proportional to the half number of available nodes is realistic. There are some aspects which make the use of the parallel `bsapply()` function difficult and require some notice to the user.

- We do not know which kind of `FUN` functions will be executed and which amount of data will be sent to the nodes or back to the master.



(a) Sequence lenght of chromosomes in human BSgenome object.



(b) Computation time for counting alphabetic frequencies.

Figure 6.4: Sequence length of chromosomes in human genome and computation time for counting alphabetic frequencies in full human genome with improved and load balanced data distribution (black) and equal distribution (red).

- The smaller the data to send and the longer the computation time, the better the performance.
- The genome library with the `Biostrings` object has to be available at all nodes, but not loaded into the R session.
- Already simple examples can have slow parallel computation times.

Therefore, only users familiar with parallel programming standards can achieve good improvements. The parallel implementation was not added to the **BSgenome** package.

### Useful Applications for the Parallel `bsapply()` Function

As demonstrated in this chapter especially communication costs limit the advantage of parallel computing. Due to low communication costs the following functions are useful for the use in the parallel `bsapply()` function: `alphabetFrequency()`, `consensusString()`, `matchPattern()`, `countPattern()`, .... Unsuitable for parallel calculations are the functions `countPDict()`, `matchPDict()`, `pairwiseAlignment()`, `consensusMatrix()`, ....

As described, creating the objects at the workers and reducing the output object, will reduce the communication costs and improves the parallel performance.

## 6.4 Summary

Especially the huge amount of data limits parallelization for next-generation sequence data. In existing parallel implementations all data have to be available at all processors. Due to the amount of data it is not possible to load all data at the master and to distribute the data over the network. In detail the raw data have to be accessible by the hard drive (e.g., a samba or nfs device), which limits the deployment on general computer clusters or grid environments. The `srapply()` function in the **ShortRead** package demonstrates a working parallel solution, in contrast the `bsapply()` function in the **BSgenome** package is a negative example for parallel computing in next-generation sequence data.

New protocols and the intrinsic curiosity of biologists are expanding the range of questions being addressed, and creating a concomitant need for flexible software analysis tools. The increasing affordability of high-throughput sequencing technologies means that multi-sample studies with non-trivial experimental designs are just around the corner. Therefore, innovative new computational tools have to be developed to manage the amount of data and to avoid long computation times. As demonstrated, existing parallel computing techniques show promising outcomes but there are obvious limitations, too.

# Chapter 7

# Large Cancer Study

Public available data sets were collected from public microarray databases, preprocessed and analyzed together. Data from more than 60 experiments and eight different cancer entities were used to demonstrate the power of parallel computing with the **affyPara** package, to discuss the difficulties of data management, and to analyze correlation between genes. Furthermore, the demand of new meta analyses and the unused potential of the existing – public available – data sets is outlined.

This is one of the first projects for analyzing several public available data sets together. Therefore, this chapter presents more technical details and problems of performing microarray analyses with huge numbers of data. Detailed biological interpretations of the results are not yet available. In future it is expected, that there will be single experiments available with more than 2000 arrays. First ideas for genome-wide association studies, prognosis studies or randomized studies to evaluate biological signatures exist. For these projects well working data management, processing and analyses tools have to be available. This chapter presents a proof of principles for a manageable data management and data processing.

Section 7.1 describes the biological idea and some biological background. Section 7.2 contains a critical comment about public available databases and data quality, details about the selected data and the data management. Section 7.3 discusses basic steps of the analysis process and problems occurring with standard analysis procedures. The chapter ends with several figures and tables of results and a small biological interpretation.

## 7.1 Biological Question(s)

The biological idea of the project is to collect microarray data from different human cancer experiments and to compare well-known pathways in different cancer entities. Especially gene-gene correlation and the reproducibility of KEGG pathways is of interest. Furthermore, the influence of standard analysis procedures from huge numbers of microrarrays to the results is not yet explored.

Based on the "Cancer Facts & Figures" statistics of the American Cancer Society (ACS, `http://www.cancer.org`) and on the availability of cancer data in the public microarray databases, eight human cancer entities are chosen. The cancer entities are coarsely grouped into two groups:

**Solid tumors:** Breast Cancer, Colon Cancer, Prostate Cancer, Lung Cancer

**Hemic disease tumors:** Leukemia Cancer (ALL, AML, CLL), Lymphoma

*Solid tumors* are an abnormal mass of tissue that usually does not contain cysts or liquid areas. Solid tumors may be *benign* (not cancer), or *malignant* (cancer). Different types of solid tumors are named for the type of cells that form them or organs where they are located. In contrast *hemic disease tumors* are tumors located in the blood or other liquid areas, e.g., in lymphoma.

In general, statistical analyses for microarray data describe differences in the mean intensity values with differential gene expression methods or use graphs to represent the correlation between different genes. This study mainly analyzes correlation but also tests for differential gene expression. Therefore, all data from one tumor group are preprocessed together. Network anaylsis tools are used to calculate the graphs of gene-gene interaction in a pathway and the estimated graphs are compared to known pathway structures. Additionally, the differential gene expression for every pathway and entity is calculated.

### 7.1.1 Pathways

Strictly speaking, one could argue that pathways do not exist, there are only networks. Furthermore, there exists no exact definition in literature. Nonetheless, pathways are useful and powerful. Biological *pathways* provide intuitive views of the myriad of interactions underlying biological processes. A typical signaling pathway, for example, can represent receptor-binding events, protein complexes, phosphorylation reactions, translocations and transcriptional regulation, with only a minimal set of symbols, lines and arrows.

In practice pathways are a set of related genes, described as lists of gene identifiers. These lists of genes are commonly used to extract some genes from an array and e.g., to test them for differential expression. Pathways are available in *pathway databases*, which enhance and complement ongoing efforts, such as *Kyoto Encyclopedia of Genes and Genomes* (KEGG, `http://www.genome.ad.jp/kegg`), Reactome (`http://www.reactome.org`), or Pathway Commons (`http://www.pathwaycommons.org`).

From the KEGG database the most famous pathways for human cancer are chosen for the large human cancer analysis. Table 7.1 lists the pathways and shows the KEGG ID, the number of genes (nodes) and edges in the pathways, and the number of genes available in the pathway and on the used chip type (HG-U133A). About 90% of the genes in the pathways are available at the used chip type. More information and graphical representations of the pathway structures are available in the KEGG database. For example the graphical structure for the p53 signaling pathway is visualized in Figure 7.1. The tumor-suppressor protein *p53* exhibits sequence-specific DNA-binding, directly interacts with various cellular and viral proteins, and induces cell cycle arrest in response to DNA damage.

| | Kegg IDs | Edges | Genes | Genes on HG-U133A |
|---|---|---|---|---|
| MAPK signaling pathway | hsa04010 | 909 | 272 | 246 |
| Cell cycle | hsa04110 | 660 | 119 | 104 |
| p53 signaling pathway | hsa04115 | 86 | 69 | 62 |
| Apoptosis | hsa04210 | 170 | 89 | 80 |
| Wnt signaling pathway | hsa04310 | 778 | 152 | 127 |
| ECM-receptor interaction | hsa04512 | 575 | 84 | 81 |
| Colorectal cancer | hsa05210 | 104 | 84 | 82 |
| Prostate cancer | hsa05215 | 295 | 90 | 86 |
| Acute myeloid leukemia | hsa05221 | 158 | 59 | 56 |
| Small cell lung cancer | hsa05222 | 223 | 86 | 86 |
| Non-small cell lung cancer | hsa05223 | 122 | 54 | 53 |

Table 7.1: Description, KEGG ID, and number of nodes and edges of the analyzed KEGG pathways.

In response to signals generated by a variety of genotoxic stresses, e.g, UV irradiation or DNA damage, p53 is expressed and undergoes post-translational modification that results in its accumulation in the nucleus. The p53-dependent pathways help to maintain genomic stability by eliminating damaged cells either by arresting them permanently or through apoptosis [JKSW99].

Due to the biological history of origins and the biological information on expression arrays it is difficult to reproduce complete graph structures of pathways. For example, signaling pathways mainly represent phosphorylation reactions, which could not be reproduced on expression arrays. But it should be possible to validate common graph structures and correlations between genes. Using more data gives more information, increases statistical power and should stabilize graph structures. As mentioned in the beginning, it is not the goal of the study to identify new biological relations. A working framework to manage, process and analyze this amount of data is presented.

## 7.2 Data Set

Microarray-based research is a scientific field where an extensive amount of data is generated and published. The ability to combine microarray data sets is advantageous to researchers to increase statistical power, to detect biological phenomena from studies where logistic considerations restrict sample size, or in studies that require the sequential hybridization of arrays.

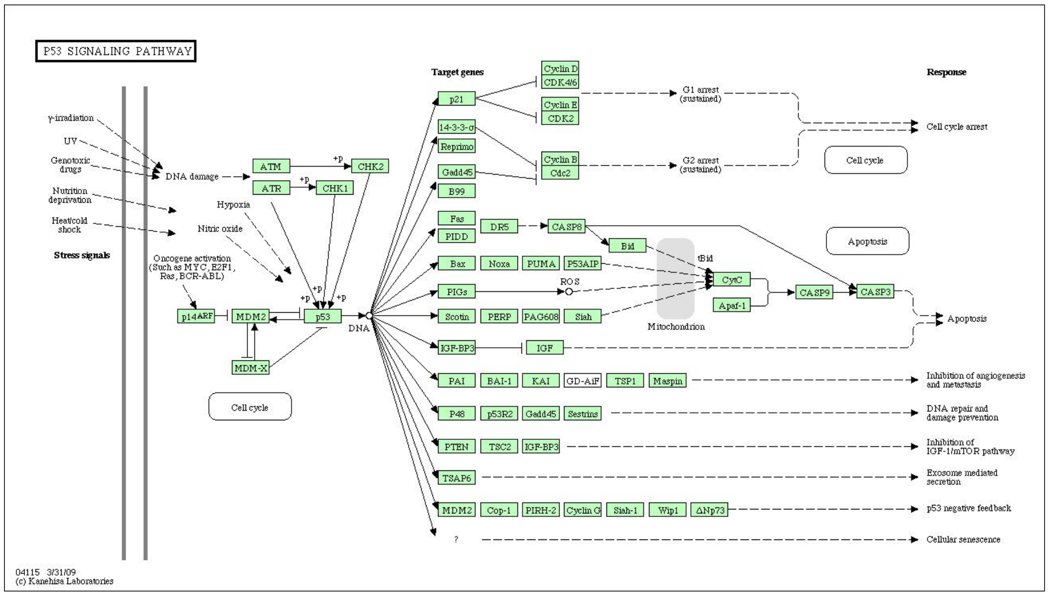


Figure 7.1: Graphical visualization of the p53 signaling pathway from the KEGG database.

### 7.2.1 Public Microarray Databases

Since the year 2000 public microarray databases have been developed and implemented for data management, for reproducibility of research and to provide data to a whole research community. There are several databases available for public data submission and those available for public query. An overview and comparison of microarray databases is available in [GGL01].

In the last years the *Gene Expression Omnibus* (GEO) and *ArrayExpress* (AE) databases have been widely used and are accepted by the Bioconductor user group:

**ArrayExpress:** Is a public archive for transcriptomics data, which is aimed at storing MIAME[1]-compliant data in accordance with MGED[2] recommendations. The ArrayExpress Warehouse stores gene-indexed expression profiles from a curated subset of experiments in the archive. (`http://www.ebi.ac.uk/microarray-as/ae`, [PKS+07, PKK+09])

**Gene Expression Omnibus:** Is a gene expression/molecular abundance repository supporting MIAME compliant data submissions, and a curated, online resource for gene expression data browsing, query, and retrieval. (`http://www.ncbi.nlm.nih.gov/geo`, [BTW+09])

[1] *MIAME* [BHQ+01] describes the Minimum Information About a Microarray Experiment that is needed to enable the interpretation of the results of the experiment unambiguously and potentially to reproduce the experiment.

[2] The *MGED* Society is an international organization of biologists, computer scientists, and data analysts that aims to facilitate biological and biomedical discovery through data integration. (`http://www.mged.org`)

Experiments from the GEO database are imported to AE database by the 'ArrayExpress team' on a weekly basis. Custom automated methods and text mining tools are used to improve data information and quality. Additionally the experiment information is manually curated before it is loaded into the AE database. All imported experiments get an ArrayExpress accession number in the format of 'E-GEOD-n', where n is a number. The import of AE data to GEO is not known or officially confirmed.

GEO is currently the largest fully public gene expression resource. Since its inception, the database has grown exponentially each year. Table 7.2 shows the number of experiments and samples in GEO and AE databases at the end of February 2009. There are nearly 300.000 microarrays available for public query.

| Database | Experiments | Samples | Experiments without FLEO | First Data |
|---|---|---|---|---|
| GEO | 11298 | 286645 | 4362 (39%) | Jan 2001 |
| AE | 7637 | 224947 | 1599 (21%) | Okt 2003 |

Table 7.2: Number of experiments and samples in GEO (published data) and AE database (27/02/2009). FLEO: feature-level extraction output

Table 7.3 and 7.4 show the five biggest chip platforms and experiments in the GEO and AE databases at the end of February 2009. More statistics are available at the database provider websites and related publications [PKS+07, PKK+09, BTW+09].

| Chip Platform | Samples in GEO | Samples in AE |
|---|---|---|
| Affymetrix GeneChip Human Genome U133 Plus 2.0 Array | 21787 | 28817 |
| Affymetrix GeneChip Human Genome U133 Array Set HG-U133A | 19994 | 37916 |
| Affymetrix GeneChip Mouse Genome 430 2.0 Array | 10468 | 13957 |
| Affymetrix GeneChip Mapping 10K 2.0 Array | 8067 | 8070 |
| Affymetrix GeneChip Murine Genome U74 Version 2 Set MG-U74A | 5490 | 10320 |

Table 7.3: Five biggest chip platforms in GEO and AE database (27/02/2009).

### Database and Data Quality

In view of software engineering the implementation and performance of the databases can be improved. There are often long response times, in AE the detail search form was not

| Experiment | Samples | GEO | AE | FLEO |
|---|---|---|---|---|
| Mapping autism risk loci using genetic linkage and chromosomal rearrangements | 6971 | GSE6754 | E-GEOD-6754 | Yes |
| Meta analysis of gene expression data from 6000 human publicly available diseased, normal samples and cell lines obtained from GEO and ArrayExpress | 5896 | - | E-TABM-185 | AE |
| Liver Pharmacology and Xenobiotic Response Repertoire | 5312 | GSE8858 | - | No |
| Combined gene expression and QTL analysis of soybean quantitative resistance to Phytophthora sojae | 2522 | GSE11611 | E-GEOD-11611 | Yes |
| Expression Project for Oncology (expO) | 1973 | GSE2109 | E-GEOD-2109 | Yes |

Table 7.4: Five biggest experiments in GEO and AE database (27/02/2009).

working for more than one month (fixed in August 2008) and GEO is not working correctly with popular web browsers (e.g., Mozilla Firefox).

In order to eliminate bias due to specific algorithms used in the original studies, and to allow consistent handling of all data sets, [RMHA08] recommends to obtain the *feature-level extraction output* (FLEO) files, such as CEL and GPR files. Unfortunately in both databases more than 20% (see Table 7.2) of the experiments do not provide FLEO files. Especially experiments from the origin of the databases (years 2000 to 2004) do not provide raw data. But even new (year 2009) experiments (see Table 7.4) do not provide FLEO files. For big experiments it is technically challenging to provide and to transfer the raw data. For example, 1000 CEL files have a data volume of more than 10GB – depending on the chip type.

The AE and GEO databases support MIAME compliant data, but due to different data archives and implementations additional transformations are required for using data from both databases. For example, in the GEO database the annotation files list the samples in columns and in the AE database they are listed in rows. Furthermore, there are no strict controls of the MIAME-compliant annotation files. Especially errors in variable names ('FactorValue..disease.state.' was found in 8 different spellings in 30 data sets), different coding of variables and unstructured information in the 'Description' variable, make the reusability of annotation files very complicated.

There is a software project called 'Tab2MAGE' (`http://tab2mage.sourceforge.net`), which aims to ease the process of submitting large microarray experiment data sets to the AE public repository database. To this end, Tab2MAGE currently includes tools to convert data and to check or validate files for MIAME-compliant sample annotation. In addition, the package includes tools to validate MAGE-TAB documents, to convert them into MAGE-ML, and to import experiments from the GEO database into Tab2MAGE format: 'GEOImport'.

Due to the mentioned weekly imports, most data sets are available in the AE database (see Table 7.4), and data management gets more feasible when using only one database. At the moment satisfactory tools for the use of data from different database platforms are not yet available. Therefore, this large cancer study only uses data from the AE database.

Similar results and (negative) experiences can be found in [IAB+09], which demonstrates that repeatability of published microarray studies is apparently limited. In the future, stricter publication rules enforcing public data availability, complete and correct data annotation (MIAME), and explicit description of data processing and analysis should be considered.

### 7.2.2 Data Set Description

This section describes the data set and data management used for the large cancer study.

#### Selection Criteria

To omit problems described in the chapter before, and to reduce costs for data handling only data from one public available database (AE) were collected. To preprocess all data together without any special data combination only one chip type is used. In the AE database most samples are of the chip type 'Affymetrix GeneChip Human Genome U133 Array Set HG-U133A' (see Table 7.3).

On February 27, 2009 all experiments from AE were included in the study, that satisfy the following selection criteria:

- Experiment available in AE.
- More than 10 arrays have chip type 'Affymetrix GeneChip Human Genome U133 Array Set HG-U133A.
- Experiment has more than 20 arrays.
- FLEO (raw data) are available.
- 50% of the arrays belong to cancer entities described in Section 7.1.

Some experiments satisfying these criteria contain identical arrays. For example the arrays from the experiments 'E-GEOD-3910' and 'E-GEOD-3911' together are identical to the arrays from the super series experiment 'E-GEOD-3912'. These experiments were not included in the study to avoid duplicate arrays. Thereby 23 experiments were excluded.

In the selection process microarray data from cell line experiments and from human patients were grouped together (data from animals were excluded). Furthermore, data from cancer subtypes were combined to one cancer entity (e.g., childhood ALL is included in the ALL cancer entity group). This coarse grain grouping for the cancer entities is chosen to get groups with more than 200 arrays.

### Selected Public Available Experiments

Selecting more than 60 public available experiments from the AE database, a large cancer data set with more than 7000 microarrays was built. An overview of the selected experiments is available in the Appendix-Tables B.1 to B.11. A detailed statistic of the data set is available in Table 7.5. The table shows the number of experiments and arrays per cancer

| | Experiments | Arrays | HG-U133A | defect | used |
|---|---|---|---|---|---|
| BREAST | 20 | 3595 | 2454 (68%) | 40 (1%) | 1834 (51%) |
| ALL | 12 | 1190 | 1140 (96%) | 3 (0%) | 916 (77%) |
| LUNG | 7 | 537 | 398 (74%) | 12 (2%) | 386 (72%) |
| COLON | 6 | 203 | 203 (100%) | 6 (3%) | 197 (97%) |
| PROSTATE | 5 | 475 | 418 (88%) | 2 (0%) | 416 (88%) |
| AML | 4 | 726 | 563 (78%) | 29 (4%) | 534 (74%) |
| LYMPHOMA | 4 | 335 | 335 (100%) | 4 (1%) | 331 (99%) |
| CLL | 3 | 194 | 182 (94%) | 5 (3%) | 177 (91%) |
| | 61 | 7255 | 5693 (78%) | 101 (1%) | 4791 (66%) |

Table 7.5: Statistic of available arrays for selected ArrayExpress experiments.

entity. Several experiments include other chip types than the HG-U133A chip. Especially in the breast cancer experiments there are more than 30% of other chip types. The tools 'boxplot' and 'MA-plot' were used (see Chapter 5) for quality assessment. If an array was incorrect in both assessments, it was marked as 'defect' and excluded. 60 defect arrays for solid cancer experiments and 41 defect arrays for hemic cancer experiments were excluded, which is about 1% of the data. Due to duplicated arrays in different experiments (from one cancer entity) and several defect arrays, the number of arrays used in the analysis (column 'used') is lower than the number of available arrays. For the breast cancer experiments only 51% can be used in the analysis and about 66% of all arrays. Therefore, the analysis described in Section 7.3 is executed on 4791 cancerous microarrays: 2833 arrays for solid tumors and 1958 arrays for hemic disease tumors.

Due to the mentioned problems in the data quality of the annotation files, it is very difficult to give more information about the cohorts. After manual correction of the annotation files descriptive statistic is possible and listed in Tables 7.6. 94% (0.4% missing) of the data are cancerous, 47% of the patients are female and 18% male (36% missing), and the median age is 55 (50% missing). Figure 7.2 shows the histogram of the age distribution for the hemic and solid cancer group. Expect for acute lymphoblastic leukemia all considered cancer entities mainly occur in older patients, which is expressed in both histograms. The first peak in the histogram for hemic cancer entities shows the group of patients with childhood acute lymphoblastic leukemia.

(a) cancerous

| | non cancerous | cancerous | NA's |
|---|---|---|---|
| hemic | 11 (0.6%) | 1942 (99.2%) | 5 (0.3%) |
| solid | 241 (8.5%) | 2577 (91%) | 15 (0.5%) |
| | 252 (5.3%) | 4519 (94.3%) | 20 (0.4%) |

(b) gender

| | female | male | NA's |
|---|---|---|---|
| hemic | 237 (12%) | 413 (21%) | 1308 (67%) |
| solid | 1997 (70%) | 435 (15%) | 401 (14%) |
| | 2234 (47%) | 848 (18%) | 1709 (36%) |

(c) age

| | Min. | 1st Qu. | Median | Mean | 3rd Qu. | Max. | NA's |
|---|---|---|---|---|---|---|---|
| hemic | 0 | 14 | 47 | 41.83 | 66 | 93 | 1246 (21%) |
| solid | 4 | 47 | 58 | 57.47 | 68 | 93 | 1773 (29%) |
| | 0 | 42 | 55 | 51.19 | 67 | 93 | 3019 (50%) |

Table 7.6: Statistic of cancerous data, gender, and age for selected ArrayExpress experiments.

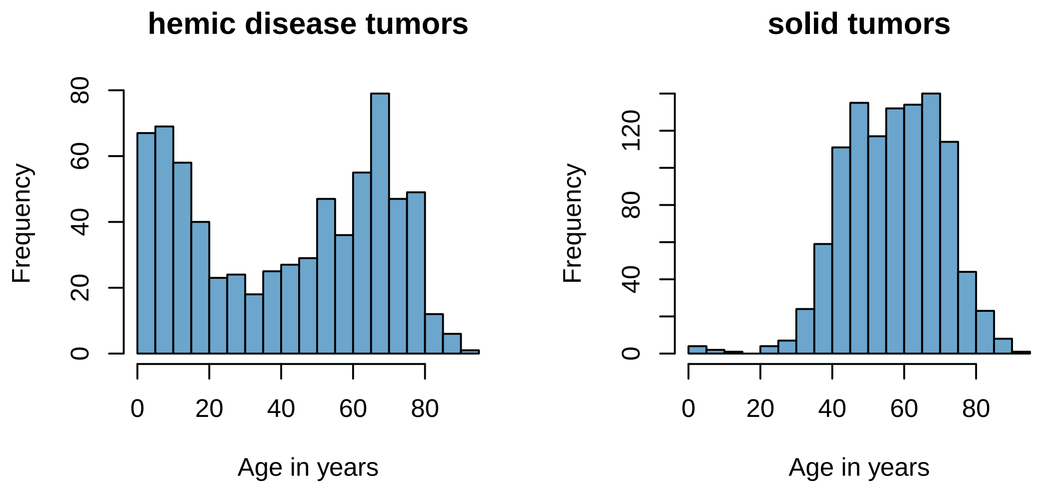


Figure 7.2: Histogram of the age distribution for the hemic and solid cancer group.

All other information (e.g., tumor grading, tumor staging) is in spite of manual correction not available for more than 10% of the complete cohort. For the data cleaning only data in the annotation files were restructured. Additional data from publications or websites were not consulted.

For the analysis of the batch effects and different influences on the data, a small data set of breast cancer data was extracted and is called 'small cancer data set' in the following sections. An overview for this data set is available in Table 7.7. The overview tables (Table 7.7 and Appendix B) list the ArrayExpress ID, the cancer entity, the title and first 300 letters of the experiment description, the number of Arrays (HG-U133A/all), and the Pubmed ID.

### 7.2.3 Data Management: ArrayExpressDataManage

The described large cancer data set consists of more than 60 single experiments and 7000 microarrays. Therefore, data management and storing becomes difficult. A special directory structure at the hard disk and R package for data management of AE experiments was designed. Thereby the data set can be reused from anyone without providing the whole data set, e.g., as new experiment in AE. The dataset can be regenerated very simply from the public available experiments in AE. The R list structure to recreate the large cancer study data is available in the Appendix B. Based on the structure of the new R package for the data management, new experiments can be added at every time point.

#### Directory Structure

To keep a clear data management, the raw data and processed data were saved in a defined directory structure on the hard disk:

**Large-cancer-study** Top directory for the large cancer study.

- **Cancer-entity-XYZ** Every cancer entity has its own directory. All experiments belonging to this entity will be stored in this directory.
  - **Array.adf.txt** Array design description.
  - **Experiment-XYZ.raw.gz** Packed CEL files for one experiment.
  - **Experiment-XYZ.sdrf.txt** Detailed sample and data relationship annotation file for one experiment.
  - **Experiment-XYZ.idrf.txt** Investigation description for one experiment.
  - **Experiment-XYZ_eset.Rdata** R objects of preprocessed experiment data saved as binary Rdata file.
- **Cancer-entity-XYZ_eset.Rdata** R objects of all experiment data, belonging to cancer entity XYZ, preprocessed together and saved as binary Rdata file.
- **complete_eset.Rdata** R objects of all experiment data preprocessed together and saved as binary Rdata file.

| ID | Entity | Title | Description | Arrays | PubMed ID |
|---|---|---|---|---|---|
| E-GEOD-11965 | breast | Contribution of HSD17B12 for estradiol biosynthesis in human breast cancer | 17beta-hydroxysteroid dehydrogenase type12 (HSD17B12) has been demonstrated to be involved in regulation of in situ biosynthesis of estradiol (E2). HSD17B12 expression was reported in breast carcinomas but its functions have remained unknown. Therefore, we examined the correlation between mRNA expre ... | 16/32 | |
| E-GEOD-4917 | breast | Time course microarray data following GR activation in MCF10A-Myc breast cells | This series contain time course microarray data from MCF10A-Myc cells treated with either ethanol or Dexamethasone for 30 min, 2 hr, 4 hr, and 24 hr. This series contains three biological replicates that were analyzed as independent replicate experiments. | 24/24 | 16690749 |
| E-GEOD-6772 | breast | Comparison of gene expression data from human and mouse breast cancers | This SuperSeries is composed of the following subset Series:; GSE6581: Expression data from mammary glands of transgenic mice; GSE6596: Comparison of gene expression data from human and mouse breast cancers: Identification of conserved breast tumor genes Experiment Overall Design: Refer to individua ... | 26/38 | 17410534 |
| E-MEXP-440 | breast | Gene expression changes associated with lapatinib treatment of breast cancer cell lines | Dose and time course response of lapatinib in breast cancer cell lines. | 36/36 | 17513611 |
| E-TABM-43 | breast | CIT-HS-BREAST-CANCER-CHEMOTHERAPY-RESPONSE | 37 samples hybridized on Affymetrix HG-U133A arrays. Analysis of advanced breast cancers treated with a dose-intense epirubicin /cyclophosphamide regimen followed by mastectomy; Validation of TP53-related genes in breast and bladder cancers. We found that a complete response to chemotherapy was only ... | 37/37 | 17388661 |

Table 7.7: Small selection of selected ArrayExpress experiments: 'small cancer data set'.

This file structure is optimized for the data processing with the R language and for reusability of intermediate results. For example, the preprocessed data are stored in the data structure or further results, e.g., graphs, can be stored, too. The CEL files are only stored as packed files to save disk memory. This data structure can be reused for other study types. The grouping into cancer entities can be generalized to all other kinds of groups (for example age or sex).

#### R Package for Data Management

Based on the data structure described in the previous section, a new package called **ArrayExpressDataManage** for data management of AE experiments was implemented. The package uses the Bioconductor package **ArrayExpress** to download data from the AE database and the data are stored in the described data structure. The main function of the new package is the function `dapply()`

```
R> library(ArrayExpressDataManage)
R> res <- dapply(function(x) print(x), path = getwd(), pattern = "*.txt",
+      recursive = TRUE)
```

This is an apply-like function which returns a list of values obtained by applying a function `FUN` to files in a directory. The directory can be defined by the variable `path` and the file type with a regular expression in the variable `pattern`. Some example functions for standard microarray processing steps (e.g., rma preprocessing) are implemented. Additionally, there are functions for data structure cleaning, creating overview tables, etc.. For preprocessing of microarray raw data serial (**affy**) or parallel (**affyPara**) packages can be used.

For more details see the vignette of the package or the help files of the package. The package is available at the R-forge repository: `http://ArrayExpressDataManage.R-forge.R-project.org/`

## 7.3 Analyses

As described in Section 7.1, different network analyses and tests for differential gene expression are performed on the large cancer data set presented in Section 7.2. Before high-level mircorray analysis methods can be used, first of all the data have to be preprocessed and checked for quality. This section presents more technical details and problems in the analysis process of the large cancer study.

### 7.3.1 Analysis Process

The following analysis process was performed on the large cancer data set using the introduced directory structure at the hard disk and the new **ArrayExpressDataManage** package.

1. Preprocessing for each of the tumor groups
   - Extract packed CEL files into temporary directory.
   - Remove arrays with wrong chip type (not HG-U133A).
   - Remove duplicate arrays: CEL files are compared using the md5 check sum.
   - Remove arrays with bad quality: boxplot and MA-plot are used for quality control.
   - Preprocess data with the parallel RMA algorithm (see **affyPara** package).
   - Create annotation data frame: combine annotation files from experiments, correct errors in annotation files and manual add a new column 'FactorValue [cancerous]' (1 = cancerous, 0 = non cancerous), to identify cancerous data.
   - Add annotation data frame to `ExpressionSet` object.
   - Save `ExpressionSet` object in '_eset.Rdata' file.
   - Correct Batch Effect using ComBat algorithm (see Section 7.3.2).
   - Save `ExpressionSet` object in '_eset.Rdata' file and old object in '_esetNoBatchCorrect.Rdata'.
2. Differential Gene Expression in each of the tumor groups
   - Only use cancerous data: 'FactorValue [cancerous]'==1.
   - Calculate GlobalAncova test statistic for all pairs of cancer entities with 10.000 permutations `perm` and for all pathways (`test.genes`).
   - Extract the top ten genes with the most influence to the test statistic.
   - Save genelists in 'diffGeneExp.Rdata' file.
3. Network Analysis with PC-Algorithm for each of the tumor groups and for every used pathway
   - Only use genes (probesets) from one pathway.
   - Only use cancerous data: 'FactorValue [cancerous]'==1.
   - Calculate correlation graph of probesets using the PC-Algorithm with $\alpha = 0.05$.
   - Build gene-gene-graph from probeset-graph: If there are any correlations between probesets from different genes, then a correlation between these genes is expected.
   - Save correlation graph object in 'graphs_PATHWAY.Rdata' file.
4. Comparing Graphs

- Visualize graphs with genes from one pathway and different cancer entities next to each other.
- Compute Structural Hamming Distance, True and False Positive Rates between all graphs, and original KEGG graphs for all pathways.
- Run permutation tests on affiliation of arrays to test for differences in graph structure (500 permutations) for all pairs of cancer entities and all pathways.

### Used Packages

In Step 1 the **ArrayExpressDataManage** and **affyPara** packages with the cluster environment at the IBE with 30 computers was used for the preprocessing of the large number of micorarrays. In Step 2 the **GlobalAncova** package was applied in parallel to test for all pairs. In Step 3 and 4 the **pcalgo** package was used and 88 (11 pathways, 4 entities, 2 tumor groups) gene-gene graphs were estimated and 308 (11 pathways, 28 pairs of entities) pairs compared in parallel. The permutation test was calculated at the computing resources at the LRZ using R version 2.8.0 and the **snow** and **Rmpi** packages for parallel calculations.

### Computation Time

Due to the huge number of microarrays all calculations are very time consuming and parallel computing has to be used to solve the problem in time. Some information about computation time and required main memory in the analysis process is presented in the following.

Using 30 computers at the IBE, the complete rma preprocessing for 2833 arrays of solid tumor patients takes about 35 minutes computation time and about 4.7 GB main memory on each computer. For 1958 arrays of hemic disease tumors the computation time was 25 minutes and a maximum main memory of 5 GB.

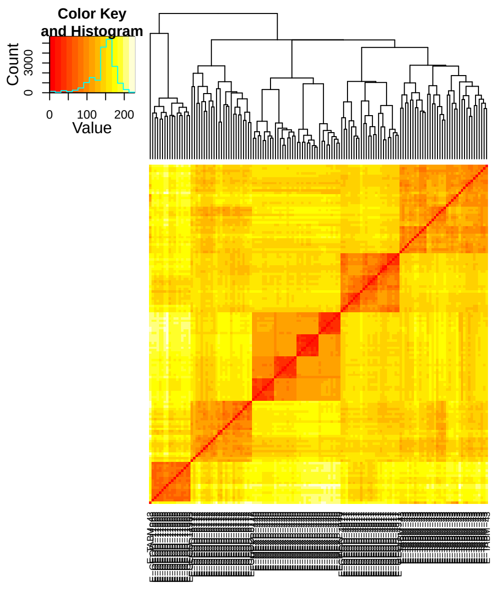
Depending on the number of arrays, edges and genes the (serial) computation time of all 88 graphs with the PC-Algorithm takes a lot of time [KB07]. In the IBE computing pool the mean serial calculation time of one graph is about 20 minutes at one node. But for the biggest pathway (MAPK signaling pathways with 250 nodes) it takes 60 minutes computation time. Therefore, the graphs were estimated and compared in parallel.

In the permutation test in Step 4 two graphs with up to 246 genes and up to 1834 arrays have to be estimated in each run, which takes up to 60 minutes. For four groups of cancer entities, six pairs have to be calculated. This takes up to 123 days (60 min $\cdot$ 500 runs $\cdot$ 6 pairs) for one pathway on a single processor. Using the cluster environment at the IBE the computation time can be reduced to about 15 hours for one pathway. Using 500 processors from the HLRB2 at the LRZ the computation time is about 3 hours and using 1000 processors the number of permutations can be doubled by the same computation time. Due to the computation time, the permutation test in Step 4 was not executed for pathways with more than 100 genes (MAPK, Wnt, p53).

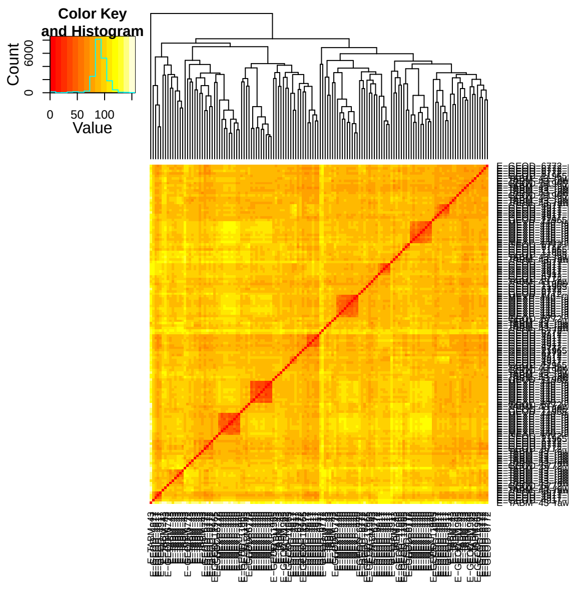
### 7.3.2 Batch Effect

Non-biological experimental variation or *batch effects* are commonly observed across multiple batches of microarray experiments [IWS+05]. Batch effects have been observed from the earliest microarray experiments and can be caused by many factors including RNA isolation batches, hybridization day, wash and reagent batches, operator, and lot-to-lot array variation. In many cases the effects of the processing batches are larger than the biological effects being studied. In general, it is inappropriate to combine data sets without adjusting for batch effects [IWS+05, JLR07].

The batch effect can be seen in the large cancer study data, too. For visualization, Figure 7.3 shows the heatmap of intensity values in the small cancer data set normalized with rma before and after batch effect correction. The orange rectangles show a strong correlation between the arrays from one experiment: *batch effect*



(a) with batch effect



(b) with batch effect corrected

Figure 7.3: Heatmap of intensity values visualizing the batch effect in breast cancer data set (after rma normalization).

As one can see in the heatmap plots or in box-and-whisker plots for the intensities of the arrays (not shown) or in Table 7.8, the batch effect is not caused by different intensity values of the batches. In addition, the batch effect will not be removed from other normalization methods. As well, *variance stabilization normalization* (VSN) [HvHS+02] using a complex error model does not remove this effect. Figure 7.4 shows hierarchical clustering of the vsn normalized small cancer data set before and after batch effect correction.

| | E-GEOD-11965 | E-GEOD-4917 | E-GEOD-6772 | E-MEXP-440 | E-TABM-43 |
|---|---|---|---|---|---|
| Min. | 7.29 | 7.30 | 7.25 | 7.29 | 7.31 |
| 1st Qu. | 7.30 | 7.35 | 7.32 | 7.33 | 7.34 |
| Median | 7.31 | 7.36 | 7.34 | 7.34 | 7.35 |
| Mean | 7.31 | 7.36 | 7.34 | 7.34 | 7.35 |
| 3rd Qu. | 7.32 | 7.37 | 7.37 | 7.36 | 7.36 |
| Max. | 7.33 | 7.49 | 7.39 | 7.37 | 7.39 |

Table 7.8: Summary of intensity values for batch effect (uncorrected) in the small cancer data set.

### Correct Batch Effect

Different methods have been proposed to filter batch effects from data. All of them have different advantages and drawbacks. At the moment it is not proved that they work correctly or do not remove biological information. In the large cancer study a parametric and non-parametric empirical Bayes framework 'Combatting batch effects when combining batches of gene expression microarray data' (ComBat) [JLR07] is used. It is robust to outliers in small (<10) sample sizes and performs comparable to existing methods for large samples. It is the only algorithm correcting batch effects, which is available in `R` code (`http://statistics.byu.edu/johnson/ComBat`). Small changes of the code are required for the application to the latest Bioconductor `ExpressionSet` object. The adjusted code is included in the **ArrayExpressDataManage** package.

**ComBat:** The algorithm is presented in [JLR07] and the method is based on empirical bayes (EB) methods and model based location and scale (L/S) adjustments. EB methods are primarily designed to borrow information across genes and experimental conditions in hope, that the borrowed information will lead to better estimates. L/S adjustments assume, that the batch effect can be modeled out by standardizing means and variances across batches.

The method assumes, that phenomena resulting in batch effects often affect many genes in similar ways (e.g., increased expression, higher variability, ...). The L/S model parameters – that represent the batch effects – are estimated by pooling information across genes in each batch to shrink the batch effect parameter estimates towards the overall mean of the batch effect estimates. These EB estimates are then used to adjust the data for batch effect, providing most robust adjustments for the batch effect on each gene. After background correction, normalization and summarization, the batch effect correction method 'ComBat' is applied to the data. The method consists of three steps:

- Standardize the data.
- EB batch effect parameter estimation using parametric empirical priors.
- Adjust the data for batch effects.

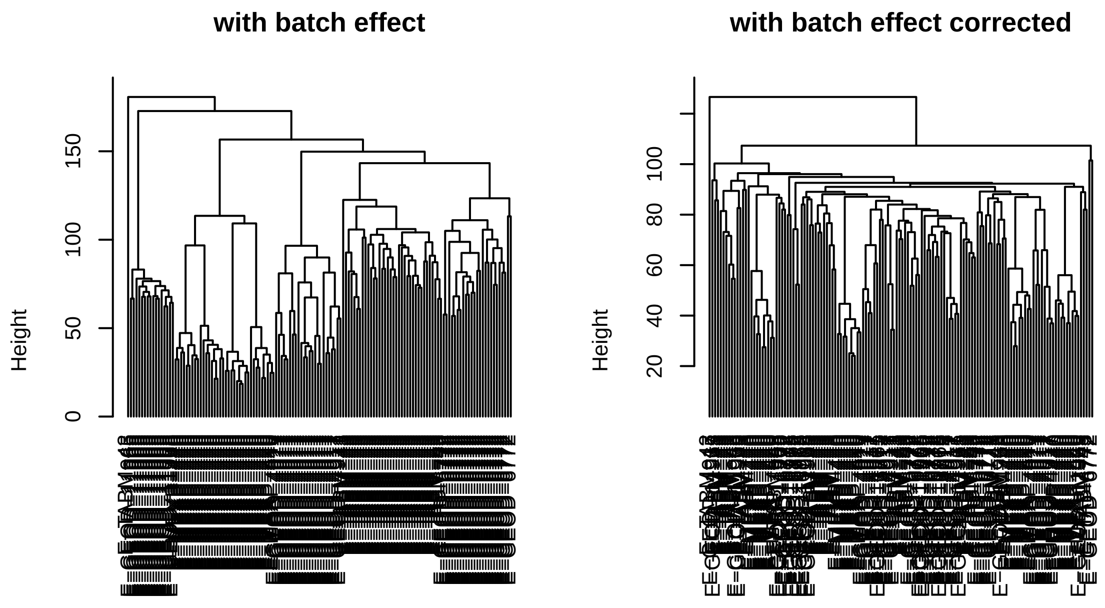


Figure 7.4: Hierarchical clustering graphic visualizing batch effect in small cancer data set (after vsn normalization).

For more details on the ComBat algorithm see [JLR07] or `http://statistics.byu.edu/johnson/ComBat`.

**Discussion:** Due to lacks of information in the annotation data mentioned in Section 7.2 no covariates were used for the batch effect correction. Furthermore, the latest code of ComBat only deals with categorical covariables.

The method is not based on a model that includes all sources of error and it is not the best way to remove sources of error in steps (i.e. take out array effects first, average probes, then remove batch). It can be shown that this approach introduces serious bias into estimates of differential expression.

One thing to keep in mind is the notion of balance. For example, if the batches are not equivalent, in the sense of ER status, HER-2 status, age, etc., then removing the batch effect can also remove or adjust the biology. Therefore, new (unpublished) approaches try to focus on methods that do not adjust biological signal while accounting for confounding (e.g., array and batch effects).

Finally, note that the EB adjustments are dependent on several factors: the standardized batch mean, the empirical prior distribution, the (residual) variance estimate, and the sample size within the batch. Sample size appears to be a very influential factor in this EB method. As sample size increases, the EB adjustment converges to the L/S batch effect parameter estimate and diverges from the empirical prior. In data sets where the sample sizes were relatively moderate (15-25 samples per batch), there is usually not much difference between the EB and the L/S adjustments [JLR07, Data supplement].

### 7.3.3 From Probes to Genes

The last step of preprocessing for Affymetrix microarrays is the summarization step, which combines the multiple preprocessed probe intensities to a single expression value (probeset). For most analyses and biological interpretations the interaction of genes is of interest. For microarray experiments usually several probesets match to one gene (Entrez ID).

$$\text{probes} \Longrightarrow \text{probesets} \Longrightarrow \text{genes}$$

Therefore, the intensities of probesets have to be combined to intensities of genes or a single probeset has to be selected as representative for a gene. In literature several methods are proposed to get from probesets to genes.

**Novel Definition Files for Human GeneChips based on GeneAnnot:** A novel set of custom Chip Definition Files (CDF) and the corresponding Bioconductor libraries for Affymetrix human GeneChips based on the information contained in the GeneAnnot database were developed. In the definitions there is a bijective mapping between genes and probes (not probesets) [FBC+07].

**Summarization for Probesets:** First model based approaches (mean) for summarizing probesets to one intensity value for one gene are published. Several studies identified no differences between the compared models [Rus08].

**Nonspecific Filtering:** Nonspecific filtering removes the probe sets that are believed to be not sufficiently informative (low variance), so that there is little point in considering them further [HHGF08].

No standardized models for summarization of probesets exist and the influence on differential gene expression or correlation between genes is not yet analyzed. A lot of genetic information gets lost using novel definition files or removing probesets with low variance. For differential gene expression this has no or only a low influence on the results. In the Bioconductor community the nonspecific filtering approach is commonly used. But for network analyses and gene correlation genes or probes with correlation are removed and a lot of information is lost.

For this work the following approach is used: Nonspecific filtering is used to remove probesets with no information (e.g., control probes or probes without EntrezID). Genes are extracted from the pathways and matched to the genes on the chip. Then the graph is created on the probesets belonging to the genes in the pathway and on the chip. From this probeset-graph a gene-graph is built. If there are any correlations between probesets from different genes, then a correlation between these genes is expected. Due to correlations between probesets belonging to the same gene, there are loops at some nodes (genes) in the graphs. This gene-graph can be compared to the graph from the KEGG database.

### 7.3.4 Influence of Batch Effect and Preprocessing

The gene-graphs of different experiments with different preprocessing methods are plotted in Figure 7.5. There is a strong influence from the batch effect correction and number of preprocessed data to the graph structure. Same effects can be shown in a lot of different microarray analyses examples, for example differential expression. The PC-Algorithm and the small cancer data set were used to demonstrate the influence.

In the experiments the correction of the batch effect has no influence on the analyses results, but a strong effect to the analyses results of all data together. In Figure 7.5(a) and 7.5(b) all graphs for the experiments have the same structure. The graphs in the last columns are the graphs of the network analysis on the whole data set. Only 40% of the edges in the batch corrected graph are available in the not batch corrected graph. For the interesting edges the strength of the correlation is similar in both graphs. Figure 7.5(b) and 7.5(c) show that preprocessing all data together has an influence on the graph structure. Only less than 50% of the edges are in both graphs but there are less than 2% of false edges (Table 7.9).

| | E-GEOD-11965 | E-GEOD-4917 | E-GEOD-6772 | E-MEXP-440 | E-TABM-43 |
|---|---|---|---|---|---|
| TPR | 0.12 | 0.16 | 0.29 | 0.34 | 0.52 |
| FPR | 0.01 | 0.02 | 0.02 | 0.02 | 0.01 |

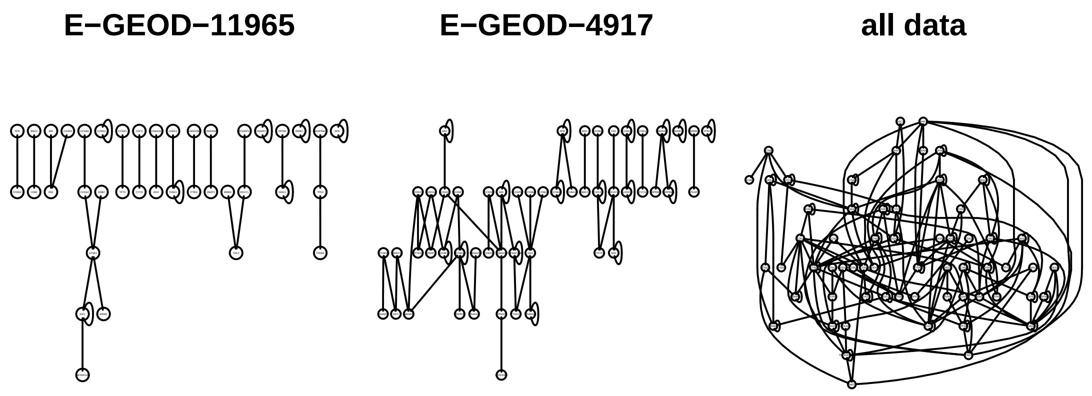
Table 7.9: Comparison between single and complete preprocessed data with the rates TPR and FPR.

Therefore, preprocessing of all data together makes arrays comparable and in view of graph structures it introduces no errors.

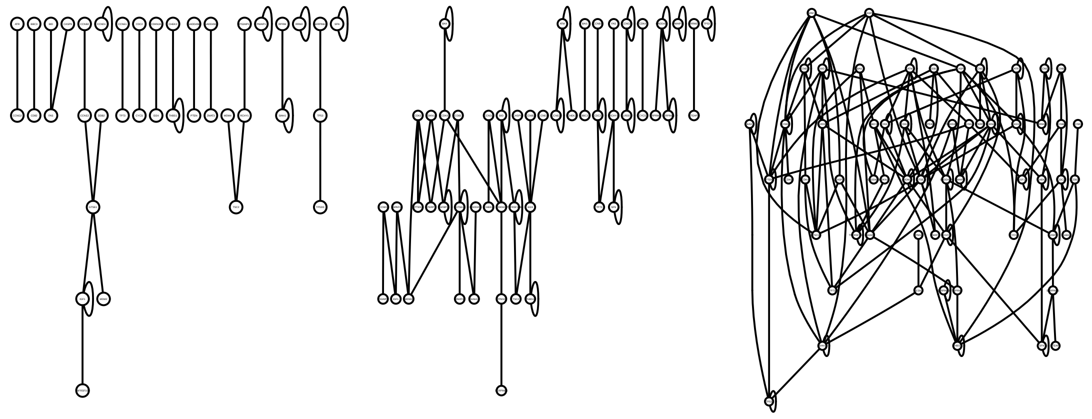
### 7.3.5 VSN Add-on Normalization

Many research institutes do not have powerful computers or computer clusters to preprocess all data together. The add-on variance stabilization normalization is a common way to preprocess huge numbers of microarray data ([KS08] and see Chapter 3.2.1). A maximal number of arrays is normalized in a first run and a so called 'reference data set' (or core data) is created. Additional arrays are added to the reference data in later runs. The new arrays are normalized with saved normalization-error-model parameters without changing the normalized core data. This method is used as well, if arrays are added to the study in different time steps [KS08].

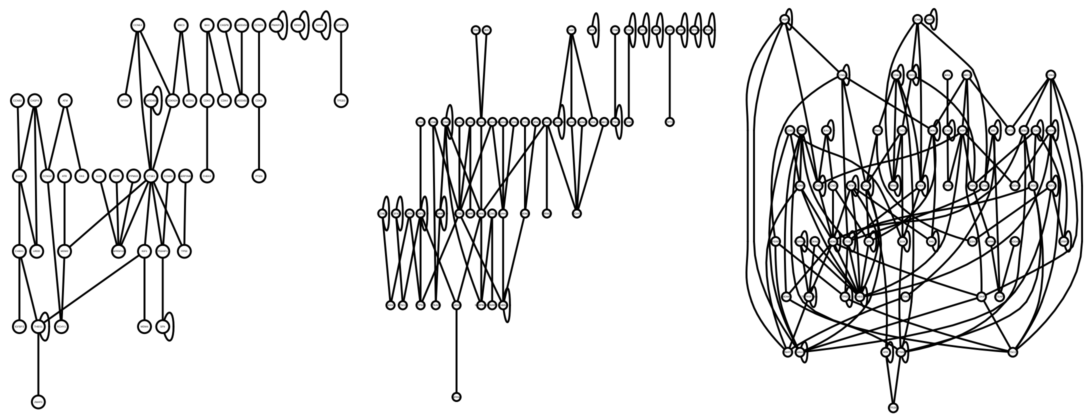
Small differences between add-on normalization and a complete normalization, influencing the results only marginally, are well known. Examples for good agreement are presented in the vignettes of the **vsn** package. The effect on the results depends on the size of the reference data, the size of the add-on arrays, the intensity of the arrays and several other factors. Furthermore, the influence of batch effects on the data was never discussed in detail. A strong batch effect (due to the factors described in Section 7.3.2)



(a) all data RMA preprocessed together



(b) all data RMA preprocessed together and for batch effect corrected



(c) data separated RMA preprocessed

Figure 7.5: Network analysis (PC-Algorithm) with the small cancer data set for genes in the p53 signaling pathway. The last column shows the graph for all data together.

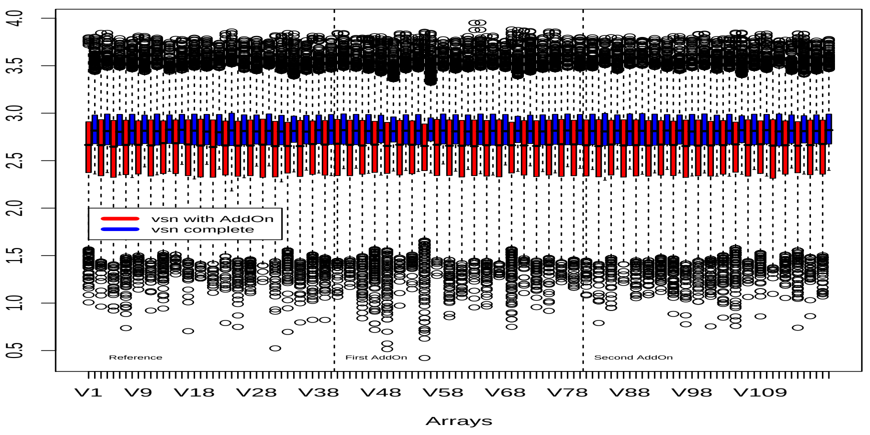


Figure 7.6: Boxplots of intensities for complete vsn normalization and vsn add-on normalization with arrays from the E-GEOD-11121 experiment.

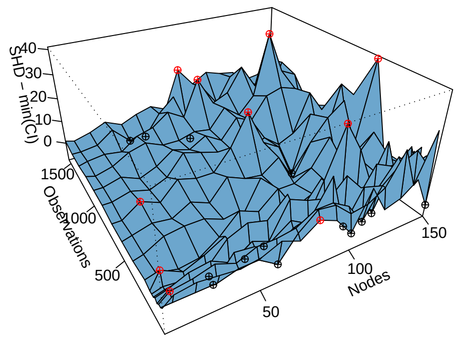
may be expected especially in studies hybridizing arrays at different time points. For example in the large multi-centre study 'Microarray Innovations in LEukaemia' (MILE) [BKH09] problems using the vsn add-on normalization for huge numbers of arrays and a long study time are reported. In the **affyPara** package vignette 'Simulation Study for VSN Add-On Normalization and Subsample Size', some examples for problems with vsn add-on normalization – especially big differences between add-on and complete normalization – are presented. Figure 7.6 shows the boxplots of intensities for complete vsn normalization compared to the vsn add-on normalization with two add-on data sets. The graphic shows the variance stabilization, but there is an obvious difference in the mean values, IQRs and outliers, too. Furthermore, problems with the size of the subsamples (rows) are discussed in the vignette.

The parallel vsn implementation in the **affyPara** package enables a vsn normalization for all data together. Unfortunately, there is only a low improvement in speed (see Chapter 5.2). Due to the mentioned problems a complete vsn normalization should be preferred, if possible.

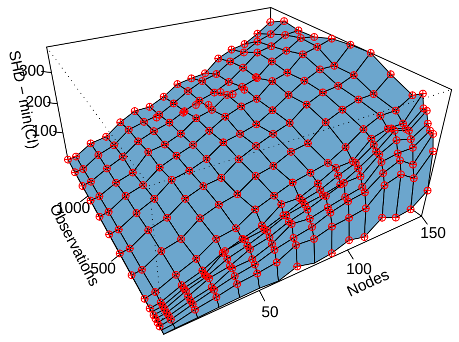
### 7.3.6 Simulation Study for Permutation Test of Array Affiliation

As described in the analysis process, to assess the differences of the graph structures from two data sets a permutation test for the arrays' group affiliation is conducted. Two graphs are estimated with the PC-Algorithm on the two original data sets and compared using the SHD. Then in 500 runs the affiliation of the arrays (observations) is permuted, the two graphs are estimated and the SHD is calculated. If the two groups of data have the same correlation structure, the original SHD should be located in the confidence interval (CI). If the original SHD is outside (bigger) of the CI, then there is a difference in the correlation structure of the two groups.

To interpret and validate the results of the permutation test in the large cancer study, a simulation study was designed. For two graphs with the same and a different graph structure normal distributed sample data were generated. A graph sparseness of 20% was assumed, which is consistent to the sparseness of KEGG graphs. The permutation test was calculated with different numbers of observations and nodes. The results of the simulation study are visualized in the Figures 7.7, which show the difference of the original SHD to the lower bound of the 95% confidence interval of the permutation test. SHDs below the CI are marked with black dots and SHDs above the CI are marked with red dots.



(a) fix graph structure



(b) different graph structure

Figure 7.7: Distance between original SHD and minimum in confidence interval of permutation test with fix and different graph structure.

For the different graph structures the SHD is bigger than the CI, which indicates the differences between the graph structures. For the fix graph structure the original SHD value is in most cases in the CI. As described in Chapter 3.2.2 for more than 50 nodes the PC algorithm overestimates the graph. Therefore, SHD values in the permutation test become bigger and the CI slides upwards. In some cases the original SHD is smaller than the lower bound of the CI (black dot). In all cases the original SHD is very close to the lower bound, whereas for different graph structures there is a clear difference between the original SHD and the upper bound.

## 7.4 Results

The goal of the study was not to identify new biological relations. First of all a working framework to manage, process and analyze the amount of data should be presented. The chapter ends with a result section presenting graphs for different cancer entities and different pathways. In addition basic differential gene expression and some statistics for similarities between the graph structures are shown. Not all graphs for all cancer entities and pathways are presented due to the limited space.

### 7.4.1 Differential Gene Expression

To study differential gene expression between the different entities in a group of genes – associated with a pathway – the GlobalAncova approach was used. Testing whether two cancer entities have different gene expression patterns GlobalAncova asks for differences in mean expression between the two clinical groups.

For hemic cancer entities as well as for solid cancer entities all pairs of entities (6 per hemic and 6 per solid tumor) and for all pathways the p-values are smaller than 0.01. From this result it is concluded that the overall gene expression profile for all genes in one pathway is associated with the clinical outcome. This means, that samples with different entity status tend to have different expression profiles. For example, Table 7.10 shows the F-statistics and the asymptotic p-values for the hemic cancer entities AML-CLL. The test

| | genes | F.value | p.approx |
|---:|---:|---:|---:|
| Cell cycle | 207 | 11.5 | 0.0000000055 |
| p53 signaling pathway | 129 | 16.1 | 0.0000000000 |
| Apoptosis | 159 | 18.7 | 0.0000000000 |
| ECM-receptor interaction | 187 | 26.7 | 0.0000000000 |
| Colorectal cancer | 166 | 13.8 | 0.0000000000 |
| Prostate cancer | 194 | 15.6 | 0.0000000000 |
| Acute myeloid leukemia | 123 | 14.0 | 0.0000000000 |
| Non-small cell lung cancer | 110 | 16.9 | 0.0000000000 |

Table 7.10: Test result matrix created from **GlobalAncova** for the hemic cancer entities AML-CLL.

results clearly indicate that the expression pattern of all pathways is different between all tested pairs of entities. This result is surprising, because the non cancer specific pathways (e.g., Cell Cyle or p53) should have a similar expression profile in all cancer entities.

Using the diagnostic plot for the interpretation of the GlobalAncova results, the genes with the most influence to the test statistic can be extracted. In the Tables 7.11 the five most influencing genes in the solid and hemic cancer entities for each pathway are listed.

Comparing these genes with literature, they are all well known to have any relation to cancer. For example the insulin-like growth factor (IGF-1) appears to play a role in prostate development and carcinogenesis [CSP$^{+}$06] or an over-expression of the CDC2 gene can be observed in leukemia cells [FTS$^{+}$95].

(a) Solid Cancer Entities

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Cell cycle | GADD45B | SFN | CDC2 | CDKN2A | MCM4 |
| p53 signaling pathway | GADD45B | IGFBP3 | CDC2 | RRM2 | SFN |
| Apoptosis | TNFSF10 | CFLAR | FAS | IRAK1 | NFKBIA |
| ECM-receptor interaction | FN1 | CD44 | COL3A1 | COL1A1 | COL4A2 |
| Colorectal cancer | IGF1R | EGFR | FOS | FZD1 | PIK3R1 |
| Prostate cancer | IGF1 | IGF1R | E2F3 | EGFR | FGFR1 |
| Acute myeloid leukemia | PIK3R1 | STAT3 | AKT3 | CCND1 | KRAS |
| Non-small cell lung cancer | FOXO3 | AKT3 | CDKN2A | E2F3 | EGFR |

(b) Hemic Cancer Entities

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Cell cycle | GADD45B | SFN | CDC2 | CDKN2A | MCM4 |
| p53 signaling pathway | GADD45B | IGFBP3 | CDC2 | RRM2 | SFN |
| Apoptosis | TNFSF10 | CFLAR | FAS | IRAK1 | NFKBIA |
| ECM-receptor interaction | FN1 | CD44 | COL3A1 | COL1A1 | COL4A2 |
| Colorectal cancer | IGF1R | EGFR | FOS | FZD1 | PIK3R1 |
| Prostate cancer | IGF1 | IGF1R | E2F3 | EGFR | FGFR1 |
| Acute myeloid leukemia | PIK3R1 | STAT3 | AKT3 | CCND1 | KRAS |
| Non-small cell lung cancer | FOXO3 | AKT3 | CDKN2A | E2F3 | EGFR |

Table 7.11: Genes with the most influence on the test statistic of **GlobalAncova** for the hemic and solid cancer entities.

### 7.4.2 Correlation for Hemic Cancer Entities

Figure 7.8 shows the graphs for the four hemic cancer entities and the cell cycle pathway (hsa04110) calculated with the PC-Algorithm with $\alpha = 0.05$. Due to the high number of

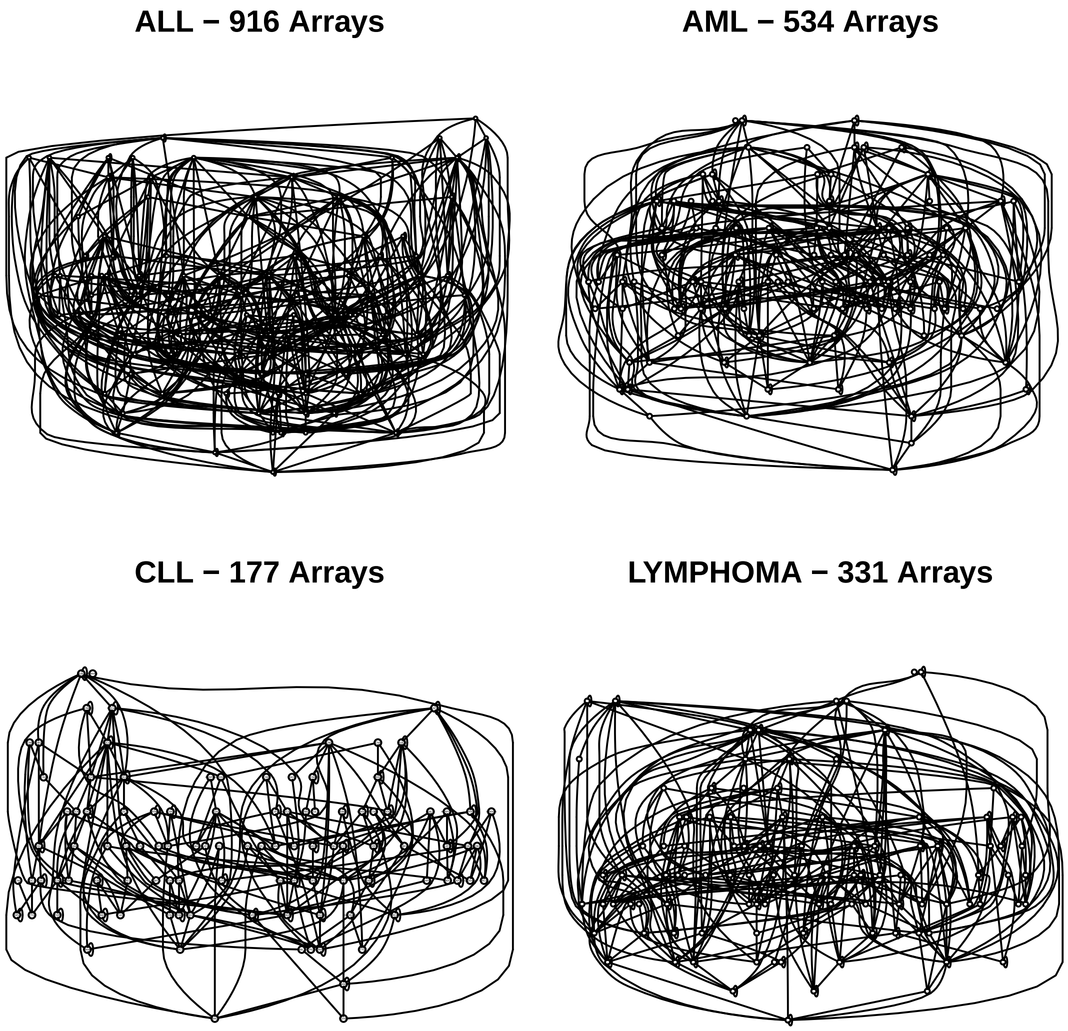


Figure 7.8: Network analysis (PC-Algorithm) for different hemic cancer entities and the cell cycle pathway (same number of nodes (genes) in all four graphs: 104).

edges (267 to 521) and nodes (same number in all four graphs: 104) it is very difficult to discover any similarities. The graphs for the other pathways have more nodes and edges (see Table 7.1). Therefore, these graphs are not plotted and differences are analyzed using the described measures for comparing graphs.

## Comparison to the KEGG graph

First of all the generated graphs are compared to the known graphs of the KEGG pathways. The true positive rates for the hemic cancer graphs are compared in Table 7.12 to the KEGG graphs (row 0). As expected less than 20% of all edges between genes can be found in the generated graphs. Especially for all signaling pathways (e.g., MAPK = hsa04010) the number of true edges is very low. Surprisingly there is no better performance in the pathways for the hemic disease tumors (e.g., Acute myeloid leukemia pathway = hsa05221) than in pathways for solid tumors (e.g., Prostate cancer pathway = hsa05215). Similar numbers of correct correlations between genes can be found. Best results are available for the ALL cancer entity, 15% of the edges in the acute myeloid leukemia pathway can be regenerated.

Additional paths – correlations or edges over one (up to 5) nodes – were analyzed. Looking at paths over three edges more than 72% of the edges in the graphs compared to the KEGG pathway can be recovered. For the CLL cancer entity the lowest number of correct paths can be found. For some entities in a couple of pathways all paths can be found. However, for example for AML in the apoptosis pathway (hsa04210) only 94% of the true edges can be found.

(a) pathway hsa04010

| | ALL | AML | CLL | LYM |
|---|---|---|---|---|
| 0 | 0.06 | 0.05 | 0.02 | 0.03 |
| 1 | 0.42 | 0.22 | 0.12 | 0.21 |
| 2 | 0.94 | 0.79 | 0.39 | 0.65 |
| 3 | 1.00 | 0.98 | 0.79 | 0.95 |
| 4 | 1.00 | 1.00 | 0.96 | 0.99 |
| 5 | 1.00 | 1.00 | 0.98 | 1.00 |

(b) pathway hsa04210

| | ALL | AML | CLL | LYM |
|---|---|---|---|---|
| 0 | 0.13 | 0.11 | 0.04 | 0.06 |
| 1 | 0.54 | 0.48 | 0.12 | 0.40 |
| 2 | 0.97 | 0.87 | 0.42 | 0.85 |
| 3 | 1.00 | 0.94 | 0.72 | 0.99 |
| 4 | 1.00 | 0.94 | 0.91 | 1.00 |
| 5 | 1.00 | 0.94 | 0.96 | 1.00 |

(c) pathway hsa05215

| | ALL | AML | CLL | LYM |
|---|---|---|---|---|
| 0 | 0.15 | 0.12 | 0.05 | 0.09 |
| 1 | 0.68 | 0.52 | 0.27 | 0.47 |
| 2 | 0.99 | 0.92 | 0.61 | 0.91 |
| 3 | 1.00 | 1.00 | 0.88 | 0.98 |
| 4 | 1.00 | 1.00 | 0.94 | 0.98 |
| 5 | 1.00 | 1.00 | 0.96 | 0.98 |

(d) pathway hsa05221

| | ALL | AML | CLL | LYM |
|---|---|---|---|---|
| 0 | 0.18 | 0.12 | 0.09 | 0.09 |
| 1 | 0.78 | 0.57 | 0.22 | 0.44 |
| 2 | 1.00 | 0.97 | 0.59 | 0.92 |
| 3 | 1.00 | 1.00 | 0.93 | 1.00 |
| 4 | 1.00 | 1.00 | 0.98 | 1.00 |
| 5 | 1.00 | 1.00 | 0.99 | 1.00 |

Table 7.12: TPR for hemic cancer entities compared to the KEGG pathways.

The listed edges in Table 7.13 are available in the KEGG pathways and in # of the four hemic cancer graphs. Less than four correlations or edges between two genes of one pathway are available in all graphs.

(a) pathway hsa04210

| Genes | # |
|---|---|
| APAF1~CASP9 | 4 |
| IL1RAP~IRAK3 | 4 |
| NFKB1~NFKBIA | 4 |
| AKT3~IKBKB | 2 |
| CASP3~CASP6 | 2 |

(b) pathway hsa05215

| Genes | # |
|---|---|
| NFKB1~NFKBIA | 4 |
| CCNE2~RB1 | 3 |
| IGF1~PDGFRA | 3 |
| AKT3~CASP9 | 2 |
| AKT3~FRAP1 | 2 |

(c) pathway hsa05221

| Genes | # |
|---|---|
| KRAS~PIK3R2 | 4 |
| RARA~SPI1 | 4 |
| KIT~PIK3CB | 3 |
| KRAS~PIK3R1 | 3 |
| AKT3~FRAP1 | 2 |

Table 7.13: Top 5 edges in the graphs of the hemic cancer entities.

### Rates

It is very difficult to identify similarities between these graphs. Therefore, the Tables 7.14 show the True Positive Rates (left) and the False Positive Rates (right) between two graphs for hemic cancer entities and different pathways. A high TPR and a low FPR are indicators

(a) pathway hsa04115

| | ALL | AML | CLL | LYM | | ALL | AML | CLL | LYM |
|---|---|---|---|---|---|---|---|---|---|
| ALL | 1.00 | 0.38 | 0.37 | 0.46 | ALL | 0.00 | 0.11 | 0.12 | 0.11 |
| AML | 0.29 | 1.00 | 0.29 | 0.33 | AML | 0.07 | 0.00 | 0.09 | 0.08 |
| CLL | 0.17 | 0.17 | 1.00 | 0.23 | CLL | 0.04 | 0.05 | 0.00 | 0.05 |
| LYM | 0.26 | 0.24 | 0.28 | 1.00 | LYM | 0.05 | 0.06 | 0.06 | 0.00 |

(b) pathway hsa04210

| | ALL | AML | CLL | LYM | | ALL | AML | CLL | LYM |
|---|---|---|---|---|---|---|---|---|---|
| ALL | 1.00 | 0.29 | 0.34 | 0.36 | ALL | 0.00 | 0.09 | 0.09 | 0.08 |
| AML | 0.24 | 1.00 | 0.23 | 0.25 | AML | 0.07 | 0.00 | 0.08 | 0.07 |
| CLL | 0.15 | 0.13 | 1.00 | 0.17 | CLL | 0.03 | 0.04 | 0.00 | 0.04 |
| LYM | 0.25 | 0.21 | 0.27 | 1.00 | LYM | 0.05 | 0.06 | 0.06 | 0.00 |

(c) pathway hsa05221

| | ALL | AML | CLL | LYM | | ALL | AML | CLL | LYM |
|---|---|---|---|---|---|---|---|---|---|
| ALL | 1.00 | 0.44 | 0.50 | 0.44 | ALL | 0.00 | 0.12 | 0.13 | 0.13 |
| AML | 0.36 | 1.00 | 0.33 | 0.33 | AML | 0.08 | 0.00 | 0.11 | 0.10 |
| CLL | 0.23 | 0.18 | 1.00 | 0.23 | CLL | 0.04 | 0.06 | 0.00 | 0.05 |
| LYM | 0.28 | 0.26 | 0.32 | 1.00 | LYM | 0.07 | 0.08 | 0.08 | 0.00 |

Table 7.14: TPR (left) and FPR (rigth) for different hemic cancer entities.

for similar edges in the graphs. The FPR is very low for all entities and pathways, which indicates a very low number of wrong edges between the graphs. There are less than 50% of the same edges in different cancer entities for the same pathway shown by the numbers of TPR. Using these two rates a low similarity between the graphs can be found. Most similarity is in the graphs of the acute myeloid leukemia pathway (hsa05221).

### Structural Hamming Distance

Another rate for the similarity of two graphs is the structural hamming distance (SHD). In simple terms, this is the number of edge insertions, deletions or flips in order to transform one graph to another graph. Therefore, a low number shows the similarity between graphs. The SHD is symmetric.

A permutation test with 500 runs was done to test for significant correlation differences between the cancer entities. The affiliation of the arrays between two groups of cancer entities was permuted, the graphs estimated, compared with the SHD metric and the confidence interval calculated.

The Tables 7.15 show the SHDs, the 95%-confidence intervals (quantiles of SHD values), and the number of nodes and edges in the graphs – to evaluate the number of SHD transformations – for hemic cancer entities and different pathways. If there would be no

(a) pathway hsa04110

| | ALL | AML | CLL | LYM | nodes | edges |
|---|---|---|---|---|---|---|
| ALL | 0 | 581 [363,410] | 542 [453,496] | 570 [399,448] | 104 | 521 |
| AML | - | 0 | 447 [345,389] | 453 [313,356] | 104 | 381 |
| CLL | - | - | 0 | 436 [331,374] | 104 | 267 |
| LYM | - | - | - | 0 | 104 | 342 |

(b) pathway hsa04210

| | ALL | AML | CLL | LYM | nodes | edges |
|---|---|---|---|---|---|---|
| ALL | 0 | 438 [274,315] | 373 [340,379] | 393 [307,347] | 80 | 367 |
| AML | - | 0 | 345 [269,305] | 383 [257,296] | 80 | 302 |
| CLL | - | - | 0 | 296 [243,279] | 80 | 182 |
| LYM | - | - | - | 0 | 80 | 265 |

(c) pathway hsa05221

| | ALL | AML | CLL | LYM | nodes | edges |
|---|---|---|---|---|---|---|
| ALL | 0 | 265 [215,251] | 242 [250,284] | 261 [213,250] | 56 | 269 |
| AML | - | 0 | 233 [215,246] | 248 [190,224] | 56 | 219 |
| CLL | - | - | 0 | 195 [173,204] | 56 | 134 |
| LYM | - | - | - | 0 | 56 | 182 |

Table 7.15: SHD, confidence intervals, and numbers of nodes and edges for different hemic cancer entities and pathways.

differences between the cancer entities, the calculated SHD should be located in the confidence interval. For most pathways and pairs of entities the SHD is obviously (very small p-values) outside (bigger) the interval. This is an indicator for the expected different correlation structures in the groups. In several pathways there is no obvious difference between the ALL-CLL, AML-CLL and CLL-LYM pairs. Figure 7.9 plots the histogram of the distribution of SHD for some selected examples. All other histograms look very similar. As described, an original SHD value lower than the CI occurs due to the bias in the graphs estimated with the PC-Algorithm.

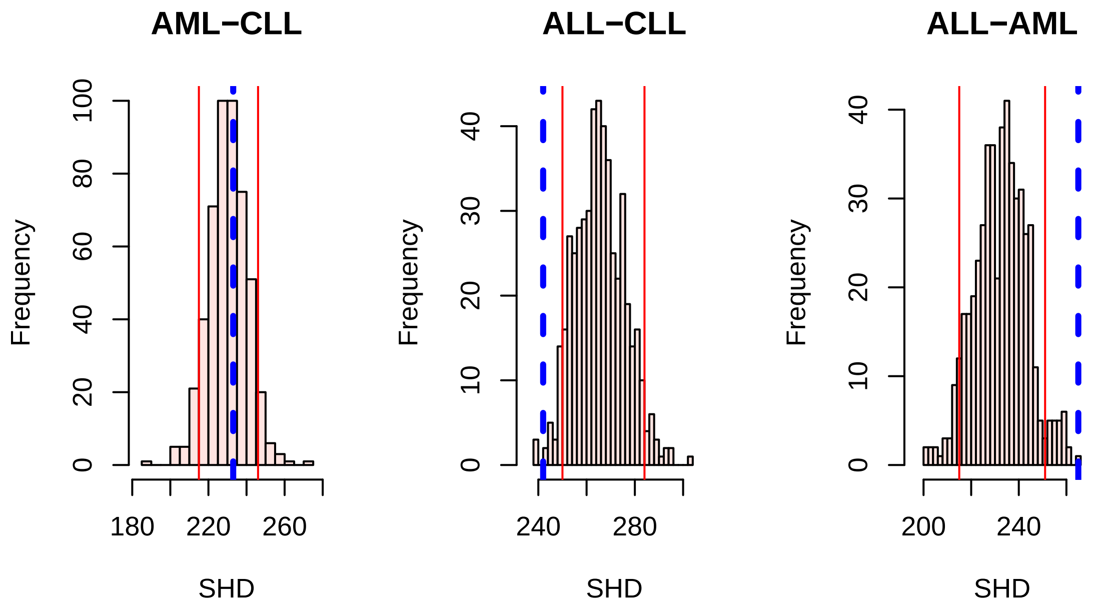


Figure 7.9: Histogram of the distribution of SHD for pathway hsa05221.

Using the SHD rate there is an obvious difference in the graph structure for all data in all pathways. For some pathways no significant differences between the cancer entities could be found in the permutation test. First of all this is an indicator for a bad grouping in the cancer entities during the creation of the large cancer study. Especially different types of ALL were combined to one group. In view of medical scientists this grouping is critical due to big biological differences in the cancer subgroups. Furthermore, gene expression signatures are very similar for some entities e.g., ALL-CLL. Due to the experiment design and the mentioned problems with the annotation files every kind of array was used. There was no differentiation for example for sex or age.

## 7.4.3 Correlation for Solid Cancer Entities

Similar to the section about correlation in the hemic cancer entities, in this section the tables for the solid cancer entities and the different pathways are presented. The graph structures are not plotted due to the complexity of the graphs.

### Comparison to the KEGG graph

First of all the generated graphs are compared to the known graphs of the KEGG pathways. In Table 7.16 the true positive rates for the hemic cancer graphs are compared to the KEGG graphs (row 0). As expected less than 22% (hemic 20%) of all edges between genes can be found in the generated graphs.

Especially for all signaling pathways (e.g., p53 signaling pathway = hsa04115) the number of true edges is very low. Surprisingly there is no better performance in the pathways for the solid tumors (e.g., colorectal cancer pathway = hsa05210) than in pathways for hemic disease tumors (e.g., acute myeloid leukemia = hsa05221). Best results are available for the BREAST cancer entity. In the non-small lung cancer pathway (hsa05223) 22% of the edges can be recovered.

Additional paths – correlations or edges over one (up to 5) nodes – were analyzed. Looking on paths over three edges more than 85% (hemic 72%) of the edges in the graph compared to the KEGG pathways can be recovered. The lowest number of correct paths can be found for the COLON cancer entity.

(a) pathway hsa04115

| | BREAST | COLON | LUNG | PROSTATE |
|---|---|---|---|---|
| 0 | 0.19 | 0.07 | 0.09 | 0.08 |
| 1 | 0.77 | 0.24 | 0.49 | 0.33 |
| 2 | 1.00 | 0.56 | 0.92 | 0.80 |
| 3 | 1.00 | 0.87 | 1.00 | 1.00 |
| 4 | 1.00 | 0.95 | 1.00 | 1.00 |
| 5 | 1.00 | 0.99 | 1.00 | 1.00 |

(b) pathway hsa05210

| | BREAST | COLON | LUNG | PROSTATE |
|---|---|---|---|---|
| 0 | 0.14 | 0.07 | 0.05 | 0.10 |
| 1 | 0.82 | 0.25 | 0.47 | 0.44 |
| 2 | 1.00 | 0.75 | 0.95 | 0.93 |
| 3 | 1.00 | 0.95 | 1.00 | 1.00 |
| 4 | 1.00 | 0.99 | 1.00 | 1.00 |
| 5 | 1.00 | 1.00 | 1.00 | 1.00 |

(c) pathway hsa05221

| | BREAST | COLON | LUNG | PROSTATE |
|---|---|---|---|---|
| 0 | 0.20 | 0.05 | 0.11 | 0.10 |
| 1 | 0.89 | 0.39 | 0.53 | 0.45 |
| 2 | 1.00 | 0.72 | 0.98 | 0.80 |
| 3 | 1.00 | 0.89 | 1.00 | 1.00 |
| 4 | 1.00 | 0.99 | 1.00 | 1.00 |
| 5 | 1.00 | 0.99 | 1.00 | 1.00 |

(d) pathway hsa05223

| | BREAST | COLON | LUNG | PROSTATE |
|---|---|---|---|---|
| 0 | 0.22 | 0.11 | 0.15 | 0.12 |
| 1 | 0.93 | 0.39 | 0.65 | 0.47 |
| 2 | 1.00 | 0.81 | 0.93 | 0.86 |
| 3 | 1.00 | 0.97 | 1.00 | 0.98 |
| 4 | 1.00 | 0.98 | 1.00 | 0.98 |
| 5 | 1.00 | 0.98 | 1.00 | 0.98 |

Table 7.16: TPR for solid cancer entities compared to the KEGG graphs.

The edges listed in Table 7.17 are available in the KEGG pathways and in # of the four solid cancer graphs. Only one correlation is available in all graphs.

(a) pathway hsa04115

| Genes | # |
|---|---|
| APAF1˜CASP9 | 4 |
| ATM˜TP53 | 3 |
| TP53˜TSC2 | 3 |
| BID˜CYCS | 1 |
| CDKN2A˜MDM2 | 1 |

(b) pathway hsa05210

| Genes | # |
|---|---|
| SMAD2˜SMAD4 | 3 |
| AKT3˜GSK3B | 2 |
| KRAS˜PIK3CB | 2 |
| KRAS˜PIK3R1 | 2 |
| AKT1˜BAD | 1 |

(c) pathway hsa05223

| Genes | # |
|---|---|
| AKT3˜FOXO3 | 3 |
| KRAS˜PIK3CB | 3 |
| KRAS˜PIK3R1 | 3 |
| ARAF˜MAP2K2 | 2 |
| EGFR˜PIK3CG | 2 |

Table 7.17: Top 5 edges in the graphs of the solid cancer entities.

### Rates

The Tables 7.18 show the TPR and the FPR between two graphs for solid cancer entities and different pathways. Similar to the hemic disease data, the FPR for all data is very low,

(a) pathway hsa04115

| | BREAST | COLON | LUNG | PROSTATE | | BREAST | COLON | LUNG | PROSTATE |
|---|---|---|---|---|---|---|---|---|---|
| BREAST | 1.00 | 0.55 | 0.38 | 0.44 | BREAST | 0.00 | 0.15 | 0.15 | 0.15 |
| COLON | 0.19 | 1.00 | 0.17 | 0.17 | COLON | 0.03 | 0.00 | 0.05 | 0.05 |
| LUNG | 0.21 | 0.28 | 1.00 | 0.21 | LUNG | 0.07 | 0.09 | 0.00 | 0.09 |
| PROSTATE | 0.22 | 0.24 | 0.19 | 1.00 | PROSTATE | 0.06 | 0.08 | 0.08 | 0.00 |

(b) pathway hsa05215

| | BREAST | COLON | LUNG | PROSTATE | | BREAST | COLON | LUNG | PROSTATE |
|---|---|---|---|---|---|---|---|---|---|
| BREAST | 1.00 | 0.52 | 0.39 | 0.39 | BREAST | 0.00 | 0.12 | 0.12 | 0.13 |
| COLON | 0.21 | 1.00 | 0.18 | 0.18 | COLON | 0.03 | 0.00 | 0.05 | 0.05 |
| LUNG | 0.23 | 0.26 | 1.00 | 0.23 | LUNG | 0.06 | 0.07 | 0.00 | 0.07 |
| PROSTATE | 0.20 | 0.22 | 0.20 | 1.00 | PROSTATE | 0.05 | 0.06 | 0.06 | 0.00 |

(c) pathway hsa05221

| | BREAST | COLON | LUNG | PROSTATE | | BREAST | COLON | LUNG | PROSTATE |
|---|---|---|---|---|---|---|---|---|---|
| BREAST | 1.00 | 0.60 | 0.46 | 0.48 | BREAST | 0.00 | 0.16 | 0.16 | 0.17 |
| COLON | 0.25 | 1.00 | 0.22 | 0.23 | COLON | 0.04 | 0.00 | 0.06 | 0.07 |
| LUNG | 0.28 | 0.33 | 1.00 | 0.25 | LUNG | 0.08 | 0.10 | 0.00 | 0.11 |
| PROSTATE | 0.25 | 0.29 | 0.21 | 1.00 | PROSTATE | 0.07 | 0.09 | 0.09 | 0.00 |

Table 7.18: TPR and FPR for different solid cancer entities.

which indicates a very low number of wrong edges. The numbers of TPR show that there are less than 60% (hemic 50%) of the same edges. A lot of edges in the BREAST data can be found in the COLON and LUNG data, but there are disproportionally many false edges for these data, too. Using these two rates a low similarity between the graphs can be found. Surprisingly, the highest number of true edges between the different entities can be found between the genes in the hemic pathway 'acute myeloid leukemia' (hsa05221).

### Structural Hamming Distance

The Tables 7.19 show the SHDs, the 95%-confidence intervals and the number or nodes and edges for solid cancer entities and different pathways.

(a) pathway hsa04210

| | BREAST | COLON | LUNG | PROSTATE | nodes | edges |
|---|---|---|---|---|---|---|
| BREAST | 0 | 475 [435,476] | 540 [397,441] | 475 [418,458] | 80 | 488 |
| COLON | - | 0 | 335 [275,314] | 298 [255,292] | 80 | 187 |
| LUNG | - | - | 0 | 375 [295,335] | 80 | 284 |
| PROSTATE | - | - | - | 0 | 80 | 245 |

(b) pathway hsa04512

| | BREAST | COLON | LUNG | PROSTATE | nodes | edges |
|---|---|---|---|---|---|---|
| BREAST | 0 | 435 [464,504] | 477 [415,460] | 453 [436,480] | 81 | 467 |
| COLON | - | 0 | 314 [301,343] | 310 [294,335] | 81 | 202 |
| LUNG | - | - | 0 | 354 [345,391] | 81 | 295 |
| PROSTATE | - | - | - | 0 | 81 | 273 |

Table 7.19: SHD and confidence interval for different solid cancer entities and pathways.

Using the SHD rate there is an obvious difference in the graph structure for all data in all pathways. Similar to the hemic data in the permutation test for some pathways and entity groups no significant differences between the cancer entities can be found. Especially for the BREAST-COLON, BREAST-PROSTATE, COLON-LUNG, and COLON-PROSTATE pairs.

### 7.4.4 Comparison of Correlation between Solid and Hemic Cancer Entities

At the end of the analysis the graphs of the solid and hemic cancer entities are compared among each other using the structural hamming distance. As expected, there is more similarity between the pairs of graphs in the four solid cancer entities and in the four hemic cancer entities than between them.

Table 7.20 prints the SHD values of all pairs in the eight cancer entities and the KEGG graph for the apoptosis pathway (hsa04210). Due to the symmetry of the SHD, the matrix is symmetric. The first column and last row is removed (contain no information) to save space.

| | BREAST | COLON | LUNG | PROSTATE | ALL | AML | CLL | LYM |
|---|---|---|---|---|---|---|---|---|
| KEGG | 590 | 310 | 405 | 362 | 466 | 409 | 299 | 379 |
| BREAST | 0 | 475 | 540 | 475 | 591 | 577 | 522 | 530 |
| COLON | 0 | 0 | 335 | 298 | 412 | 366 | 277 | 329 |
| LUNG | 0 | 0 | 0 | 375 | 469 | 447 | 348 | 400 |
| PROSTATE | 0 | 0 | 0 | 0 | 434 | 392 | 313 | 349 |
| ALL | 0 | 0 | 0 | 0 | 0 | 438 | 373 | 393 |
| AML | 0 | 0 | 0 | 0 | 0 | 0 | 345 | 383 |
| CLL | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 296 |

Table 7.20: SHD between pairs of KEGG graph, solid and hemic cancer entities for the Apoptosis pathway (hsa04210).

Figure 7.10 visualizes the distribution of the SHD for the solid cancer, hemic cancer and mixed (solid+hemic) pairs. The data for the hemic and solid cancer entities include the 0, because there is no difference (SHD=0) between the same graphs (e.g., ALL-ALL). In all 11 analysed pathways the mean value of the mixed pairs is higher than in the other two groups. This indicates, that there is a different correlation structure for genes in solid and hemic disease tumors.

## 7.5 Summary

This chapter demonstrates that the use of parallel computing allows the analysis of more than 4000 microarrays and reduces the computation time to less than one day. A R package for data management of microarray experiments was presented and is available at the R-forge repository: `http://ArrayExpressDataManage.R-forge.R-project.org/`

The results of the analysis are very similar for both groups of data:

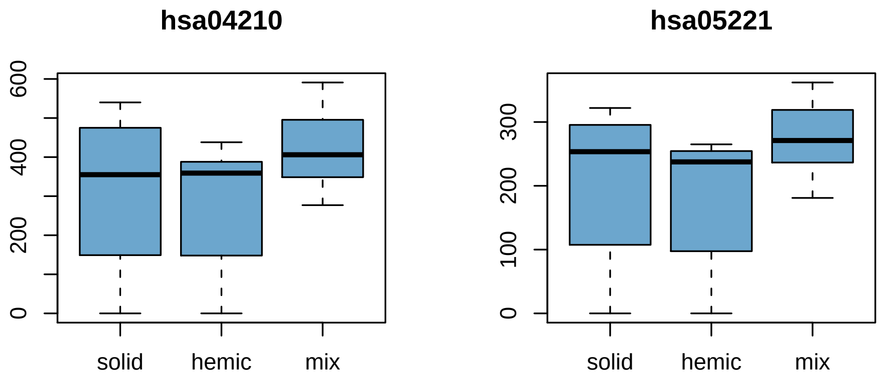


Figure 7.10: Boxplots for SHD distribution between solid cancer and hemic cancer graphs and between solid and hemic graphs (mix) for the pathways hsa04210 hsa05221.

- Testing the used data for differential gene expression shows a very strong differential gene expression for many genes and no consistent differential expression profile for the analysed entities. The strong differential gene expression (small p values) is an indicator for problems in the grouping – only eight entities –. It is well known, that there are huge differences in data from subentities. This coarse grouping was chosen to keep acceptable group sizes (more than 180 arrays) .
- There is only a small similarity between the calculated graphs (PC-Algorithm) and the KEGG pathways. This indicates that the paths in biological pathways can not be indicated by expression microarrays. But by analysing paths (correlations) over three other nodes (genes) about 80% of the paths in the KEGG pathways can be proven.
- Comparing the calculated graphs for each entity based on the pathways and using the discussed rates as measurement, up to 60% of the same correlation between genes can be found. However, these correlations are different to the KEGG pathways.
- Using a permutation test in the arrays of the cancer entities and comparing the calculated graphs with the SHD, for several pairs significant differences in the gene-gene correlation structure between the cohorts of cancer entities can be found.

The same study could be executed on public available data of the chip type 'HG-U133 Plus 2.0'. This chip type is a newer generation and stores more probes and genes. The number of arrays will be less than 7000 but nearly all genes of the pathways should be available on the chip. Furthermore, due to younger data (2003 to today) the data and annotation file quality should be better. Therefore, the data can be analyzed more in detail (e.g., female - male) and can be compared to the presented large data study.

# Chapter 8

# Summary and Outlook

Bioinformatics as an interdisciplinary research area at the interface between computer science and biological science has been confronted with new challenges in the last few years. Especially large data sets and increased computational requirements stemming from more sophisticated methodologies require new computational and statistical solutions, ideas and approaches. This work demonstrates the usefulness of parallel computing as to solving these new challenges as well as its power and limitations with two well-known biological examples. Basic results are published or submitted to relevant journals.

The summary chapter describes the state of development and discusses open topics regarding further parallel applications. In the end, two current trends for the future of parallel computing will be outlined in detail.

## 8.1 State of Development

For the open-source programming language R – a software environment for statistical computing and graphics – research has focused on using parallel computing techniques in the last decade. Existing packages for different parallel computing hardware environments were compared [SME$^+$09]. The **snow** and **Rmpi** package stand out as particularly useful for general use on computer clusters, the **multicore** package for multi-core systems.

For preprocessing of high-density oligonucleotide microarrays, the **affyPara** package based on the **snow** package was developed [SM08, SM09]. Existing statistical algorithms and data structures had to be adjusted and reformulated for parallel computing. Using the parallel infrastructure, the known methods could be enhanced and new methods have became available. Parallelization of existing preprocessing methods produces, in view of machine accuracy, the same results as serialized methods. The partition of data and distribution to several nodes solves the main memory problems and accelerates the method up to factor 15 for 300 arrays or more.

For next-generation sequence data, improvements could be achieved using parallel computing in the R language with the **snow** or **Rmpi** package, but existing data structures and huge amounts of data (network traffic) limit its usefulness. Using the **multicore** package can accelerate the process considerably. However, a huge amount of main memory is required.

The parallelized preprocessing methods were used to analyze more than 7000 microarray data from more then 60 experiments from the public microarray database ArrayExpress. For data management, a new package called **ArrayExpressDataManage** was developed. The study proves the feasibility of data management and of analysing this amount of microarray data. Eight different cancer entities were studied, and the gene-gene interaction of more than ten KEGG pathways was estimated. Adequate similarities between the graphs were found, but it is not possible to rebuild more than 30% of KEGG pathways using gene expression data.

To conclude, microarrays remain a useful technology to address a wide area of biological problems and the optimal analysis tool of these data to extract meaningful results, but still pose many bioinformatic challenges. Sequencing based characterization of transcriptome is appealing because it effectively surmounts the limitations of microarrays. As access disseminates and costs for next-generation sequencing continue to drop, it seems probable that a steadily increasing fraction of the community will begin to use sequencing, rather than microarrays, to interrogate biological phenomena at the genomic scale [She08].

## 8.2 Open Topics

The **affyPara** package provides parallelized preprocessing methods for oligonucleotide microarray data. The most frequently used and most famous methods for background correction, normalization and summarization are provided in the package. Due to the modular software design of the package, the extension to further preprocessing methods is possible. At the moment, the package does not support all Affymetrix chip types. In theory, similar parallel computing techniques can be adapted, as for example exon, SNP or tiling arrays. Furthermore, the package provides methods for quality control of large amounts of microarray data. Especially the graphical visualization of this amount of data is quite difficult and is supposed to be improved. The use of interactive GUIs seems to be a promising solution. Developments in R-GUI interfaces are in progress in the R user community (e.g., **RGG** package, [VDV+09]).

New techniques for analyzing microarray data are constantly being developed. Although the underlying distribution of microarray data is not known, analysis methods are typically assumed to have normally distributed data. This fact can have misleading consequences. [HW09] demonstrates that preprocessed microarray data are not – as a rule – well approximated by the normal distribution. Furthermore, not having normal data can yield misleading results for both standard and novel analysis methods. Further developments in low- and high-level analysis should focus on model-based methods as described in Chapter 2.2.3, and new methods should be tested in simulations with heavier-tailed and/or skewed distributions. In addition, new developments have to deal with new challenges, such as larger data sets and increased computational requirements stemming from more sophisticated methodologies. This work demonstrates the usefulness of parallel computing in solving these challenges. Using the R language and existing parallel computing packages provides an excellent framework for parallel applications. In other research areas, a lot of parallel methods exists and can be adopted to bioinformatics, especially to genomic data. For example, the parallel implementation of a fuzzy c-means cluster analysis for oil and gas exploration – presented in [MEC08] – is an efficient parallel implementation for cluster analysis.

Finally there are new developments in high-performance computing every day. These techniques have to be evaluated regarding their usefulness in biological applications. The flexibility of the R package system allows integration of many different technologies, and small test environments can be implemented very fast. At the moment (August 2009) the development of R-interfaces to the trends of Cloud Computing and GPUs should not be missed (for more details see Section 8.3 below). Furthermore, as described in [SME$^+$09], there is a great development potential in existing R packages for HPC. For example the long communication times in the **snow** package (see Chapter 5.3) could be improved using non-blocking communication mechanisms or broadcast commands. As well, multi-core systems are now very common, and the number of processors per chip is growing. There is a compelling need for integration of R code into multi-core environments (see Chapter 8.3).

## 8.3 The Future is Parallel

The popularized version of *Moore's law* [Moo65], expected to double performance per processing element every few years, has ended in 2006 (see International Technology Roadmap for Semiconductors (ITRS) in Figure 8.1). While Dr. Gordon Moore's original observa-

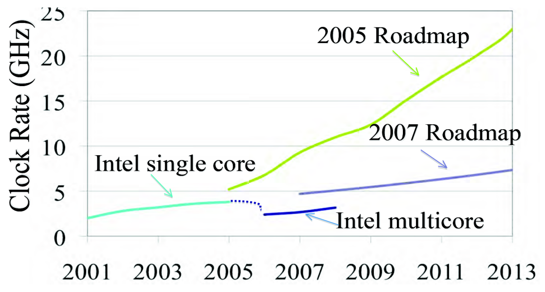


Figure 8.1: ITRS Clock Rate Roadmap 2005 and 2007.

tion of doubling transistor-count every 18 to 24 months is still holding true, by 2006 the popular expectation of doubling performance per processing core could not longer be met. Due to physical and practical problems – transmission speed, limits of miniaturization, end of voltage scaling, economic limitations, temperature (see Chapter 4.2) – it has not been possible to produce faster processors. Therefore, chip producers found it necessary to produce a chip with two or more cores to increase total performance. With the prospect of more systems with four, eight, or more processors in a machine, it is left to software programmers to use these systems efficiently. The need for programming parallel computers

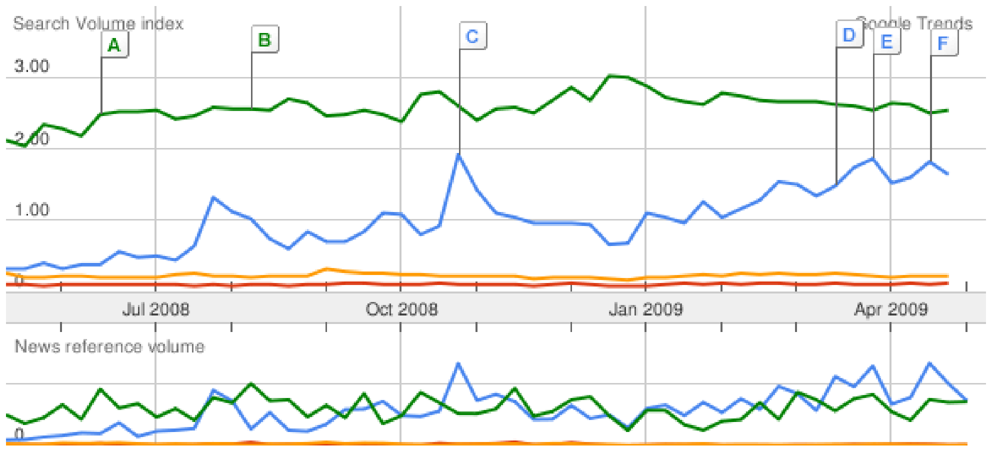


Figure 8.2: Interest in Cloud Computing visualized with Google Trends from May 2008 to 2009. Green: GPU, blue: Cloud Computing, orange: Grid Computing, red: Parallel Computing

has arrived at the users' desktops. Therefore, users have to start thinking in parallel when developing new software or codes. If not, there will be a lot of idle processors in the future.

$$\text{Performance} = \text{Parallelism}$$

Beside the new multi-core chip generation, there are further environments which require parallel computing approaches. John Burdette Gage from Sun Microsystems had a vision 25 years ago: 'Network is the computer' (`http://news.cnet.com/8301-10784_3-9964131-7.html`). Today, Cloud Computing could have the potential for this vision to become more relevant than ever. Cloud Computing is the next generation of Grid Computing and is growing in interest and importance every day. Figure 8.2 visualizes the Cloud Computing trend during the last year using Google Trends Labs[1].

The search keywords and available websites for parallel computing (red) and grid computing (orange) have remained stable during the last year. But for Cloud Computing (blue) and GPUs (green), there has been an obvious and growing interest in the last one to two years. The number of reference websites is growing, and especially for Cloud Computing, there is an increasing number of queries. Even in the field of bioinformatics, there is an increased interest in and demand for Cloud Computing [BW09]. Furthermore, for the near future, it is expected that the hardware architecture will be a combination of specialised CPU and GPU type cores [Meu09].

[1]Google Trends compares the world's interest in favorite topics and visualizes how often certain keywords have been searched via Google over a certain time period. Google Trends also shows how frequently topics have appeared in Google News stories, and in which geographic regions people have searched for them most (`http://www.google.com/trends`).

### 8.3.1 Cloud Computing

Due to the inflationary use of the term *Cloud Computing*, it is very difficult to give an exact definition of the term. Following [Vou08, BK09], Cloud Computing allows users to access scalable, real-time, Internet-based information technology services and resources. It is a type of computing in which resources are provided as a service over the Internet to users who do not need to have knowledge of, expertise in, or control over the technology infrastructure. Cloud Computing is often confused with grid computing, utility computing and autonomic computing. Indeed many cloud computing deployments depend on grids, have autonomic characteristics and bill-like utilities - cloud computing can thus be seen as a natural next step from the grid-utility model. The major part of a cloud computing infrastructure consists of reliable services delivered through data centers and computer servers. The services are accessible anywhere in the world, with 'The Cloud' appearing as a single point of access for all the computing needs of consumers. Open-standards and open-source software are available. As customers generally do not own the infrastructure, they merely access or rent it. Thus they can avoid having to pay large sums, consume resources as a service and only pay what they use. Many cloud-computing offerings have adopted the utility computing model, which is analogous to how traditional utilities like electricity are consumed, while others are billed on a subscription basis. Sharing 'perishable and intangible' computing power among multiple tenants can improve utilization rates. As a result, servers are not left idle, which can reduce costs significantly while increasing the speed of application development.

Developing and operating a R Cloud Computing Infrastructure will provide a cost-effective cluster/grid-based computing to end users. It offers computer resources to all R users and research areas conducted to the R language.

Two basic concepts have to be developed: A business model and a software environment. Both concepts have to meet different demands like the R GUI as a user interface, code execution in parallel or serial, attractive and fundable pricing, etc.. A service-provider-oriented infrastructure has to be developed due to the costs for cloud resources. In terms of software, three different modules or R packages have to be implemented (see Figure 8.3).

### 8.3.2 GPGPU

A serious competitor for the multi-core CPU is represented by *Graphical Processing Units* (GPUs), which are graphical cards used for scientific computing. *General-Purpose computing on Graphics Processing Units* (GPGPU) is the technique of using a GPU, which typically handles computation only for computer graphics, to perform computation in applications traditionally handled by the CPU. GPUs are only suitable for tasks that perform some type of number crunching within a parallel processing environment. Today the fastest GPUs ('Nvidia Tesla') are already in the teraflops range, whereas normal multi-core chips are slowly reaching this milestone. The real problem with GPUs is that they may not be programmed as is usual for CPUs. Therefore, Nvidia GPUs offer the support of the *Com-*

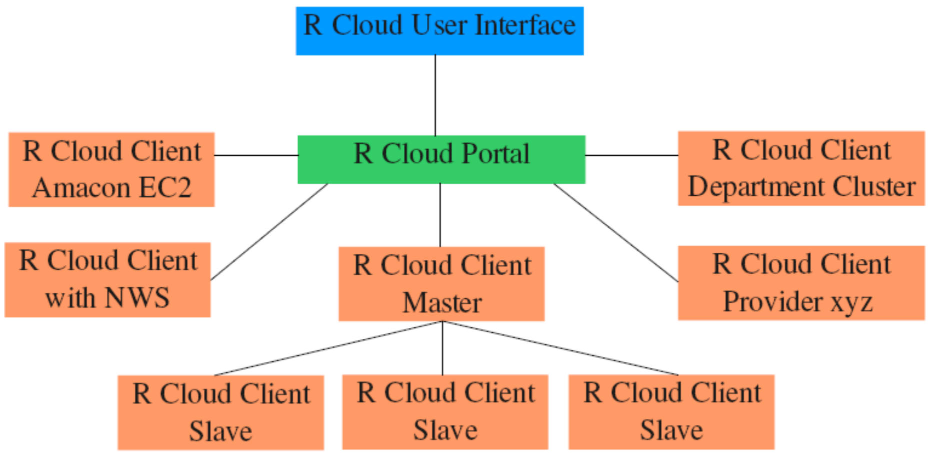


Figure 8.3: Software modules for a R Cloud Computing Infrastructure.

*puter Unified Device Architecture* (CUDA, [Nvi09]) library that provides a set of user-level subroutines and allows the GPU to be programmed with standard C or Fortran. This user interface could be used to integrate the new GPU computing power into the R language. A first example implementation was proposed in June 2009 in the form of the R package **gputools**.

Due to the limited memory at the graphical cards, an optimized data structure is required for efficient programming. One efficient example application in bioinformatics using GPUs is sequence alignment [MV08, LMS09]. Up to now, these are C code implementations and do not have an integration into the R language.

# Appendix A

# Documentation of Appended DVD

To assure the reproducibility of all results in this dissertation a DVD with code and example data is available. If the disk is not attached to this version of the PhD thesis, please contact the author. The DVD contains the following directories and files:

**README.txt:** A README file with important information for the use of the DVD.

**diss_schmidb.pdf:** The PDF version of the PhD thesis.

**largeCancerStudy:** As described in the thesis, it is not required to provide the raw data from all experiments and these data do not fit onto one DVD. The **ArrayExpressDataManage** package can be used to regenerate the directory structure and to download the raw data for the large cancer study.

- **create_large_cancer_study.R:** The R file to create the directory structure and to download all experiment data from the ArrayExpress database.
- **hemic:** A directory with the SDRF (Sample and Data Relationship Format) file for all hemic cancer data, a Rdata file with the complete rma preprocessed `ExpressionSet` object and Rdata files with the estimated graphs.
- **solid:** A directory with the SDRF (Sample and Data Relationship Format) file for all solid cancer data, a Rdata file with the complete rma preprocessed `ExpressionSet` object and Rdata files with the estimated graphs.

**Rcode:** A directory with the most important R code for this PhD thesis. In all files the path variables to the file structure have to be adapted and a computer cluster is required. Most of the code files are configured for the import into the Sun Grid Engine at the IBE computer pool.

- **compareMethods:** A directory with the R code files to reproduce the benchmarks of the **affyPara** package.
- **largeCancerStudy:** A directory with the R code files to reproduce the large cancer study.

**simulationPCalg:** A directory with the R code files to reproduce the simulation study for the permutation test of array affiliation.

**Rpackages:** A directory with the latest versions of the **affyPara** and **ArrayExpressDataManage** package (source code and windows build).

**vignettes:** A directory with the vignettes.

For the R packages it is advised to install the latest package versions from the repositories, because these versions are continuously enhanced and work correctly with the latest Bioconductor base packages. Furthermore, the additional required R packages (Dependencies) are automatically installed, too.

**affyPara:** `http://www.bioconductor.org/packages/release/bioc/html/affyPara.html`

```
R> install.packages("affyPara", repos = "http://www.bioconductor.org")
```

**ArrayExpressDataManage:** `http://r-forge.r-project.org/projects/aedatamanage`

```
R> install.packages("ArrayExpressDataManage",
+                   repos = "http://R-Forge.R-project.org")
```

# Appendix B

# Description of the Large Cancer Data Set

From the public available database AE microarray data were collected. To preprocess all data together without any special data combination only data from one chip type (HG-U133A) were used and only experiments satisfying the different selection criteria were included into the study. Selecting more than 60 public available experiments from the AE database a large cancer data set with more than 7000 microarrays was built. An detailed overview of the selected experiments is available in the following tables. The overview tables list the ArrayExpress ID, the cancer entity, the title and first 400 letters of the description of the experiment, the number of Arrays (cancerous/HGU133A/all) and the Pubmed ID.

The data set was created using the new developed **ArrayExpressDataManage** package. The following commands (and experiments) were used:

```
R> library(ArrayExpressDataManage)
R> path_small <- createDataStruct(path = '/home/cancdat',
+          data= list(
+                  breast=c('E-GEOD-6772', 'E-TABM-43', 'E-MEXP-440',
+                          'E-GEOD-11965', 'E-GEOD-4917')
+                  #E-GEOD-6596 included in E-GEOD-6772
+          ),
+          name='small')
R> # Solid tumors
R> path_solid <- createDataStruct(path = '/home/cancdat',
+  data= list(
+          breast=c('E-GEOD-6532', 'E-GEOD-4922',  'E-GEOD-1456',
+                  'E-GEOD-11121', 'E-GEOD-7390', 'E-GEOD-12093', 'E-GEOD-2603',
+                  'E-GEOD-5462', 'E-GEOD-9936', 'E-GEOD-5847', 'E-MTAB-7',
+                  'E-GEOD-1561', 'E-TABM-43', 'E-MEXP-440', 'E-GEOD-11965',
+                  'E-GEOD-6772', 'E-GEOD-9574', 'E-GEOD-4917', 'E-GEOD-3494',
+                  'E-GEOD-2990' ),
+                   #E-GEOD-6883 to small
+                   #E-TABM-244 included in E-MTAB-7
+          prostate=c('E-GEOD-8218', 'E-TABM-26', 'E-TABM-90', 'E-MEXP-1327',
+                  'E-GEOD-2443'),
+          colon=c('E-MTAB-57', 'E-GEOD-4045', 'E-MEXP-383', 'E-MEXP-101',
+                  'E-GEOD-2742', 'E-MEXP-833'),
+          lung=c('E-GEOD-4824', 'E-GEOD-10072', 'E-GEOD-6253', 'E-GEOD-7670',
+                  'E-MEXP-231', 'E-TABM-15', 'E-GEOD-4127')
+  ),
+  name='solid')
R> #Hemic tumors
R> path_hemic <- createDataStruct(path = '/home/cancdat',
+  data= list(
+          cll=c('E-GEOD-11038', 'E-GEOD-8835', 'E-GEOD-6691'),
+                  #E-GEOD-9992 included in E-GEOD-11038 (super series)
+          aml=c('E-GEOD-12417','E-GEOD-1159','E-GEOD-9476', 'E-GEOD-1729'),
+                  #E-GEOD-8970 defect annotation file
+          all=c('E-GEOD-12995','E-GEOD-635', 'E-GEOD-10255', 'E-GEOD-2351',
+                  'E-GEOD-3912', 'E-MEXP-313', 'E-GEOD-14618', 'E-TABM-125',
+                  'E-GEOD-4698', 'E-GEOD-8879', 'E-MEXP-120', 'E-GEOD-1577'),
+                  #E-GEOD-14613 included in E-GEOD-14618
+                  #E-GEOD-3910 and E-GEOD-3911 included in
+                  #E-GEOD-3912 (super series)
+                  #E-GEOD-643-660 included in E-GEOD-635 (super series)
+          lymphoma=c('E-GEOD-4475','E-TABM-346','E-TABM-117', 'E-GEOD-8388')
+                  #E-GEOD-4176 is to small
+          ),
+  name='hemic')
```

For more details see Chapter 7.2.

| ID | Entity | Title | Description | Arrays | PubMed ID |
|---|---|---|---|---|---|
| E-GEOD-10255 | all | Gene expression in primary acute lymphoblastic leukemia (ALL) associated with methotrexate treatment response | Genome-wide assessment of gene expression in primary acute lymphoblastic leukemia cells was performed to identify genomic determinants of MTX{\~A}{\textcent}{\^A}?{\^A}?s antileukemic effects. Reduction of circulating leukemia cells after in vivo methotrexate treatment served as a measure MTX's antileukemic effects. Experiment Overall Design: Gene expression in diagnostic primary acute lymphoblastic leukemia cells from bo ... | 161/161 | |
| E-GEOD-12995 | all | Expression data for diagnosis acute lymphoblastic leukemia samples | We studied a cohort of 221 high-risk pediatric B-progenitor ALL patients that excluded known high risk ALL subtypes (BCR-ABL1 and infant ALL), using Affymetrix single nucleotide polymorphism microarrays, gene expression profiling and candidate gene resequencing. A CNA poor outcome predictor was identified using a semi-supervised principal components approach, and tested in an independent validatio ... | 175/175 | |
| E-GEOD-14618 | all | Microarray analyses of induction failure in T-ALL | The clinical and cytogenetic features associated with T-cell acute lymphoblastic leukemia (T-ALL) are not predictive of early treatment failure. Based on the hypothesis that microarrays might identify patients who fail therapy, we used the Affymetrix U133 Plus 2.0 chip and prediction analysis of microarrays (PAM) to profile 50 newly diagnosed patients who were treated in the Children's Oncology Gr ... | 42/92 | 17495134 |
| E-GEOD-1577 | all | T-ALL and T-lymphoblastic lymphoma | T-cell acute lymphoblastic leukemia (T-ALL) and T-cell lymphoblastic lymphoma (T-LL) and are often thought to represent a spectrum of a single disease. The malignant cells in T-ALL and T-LL are morphologically indistinguishable, and they share the expression of common cell surface antigens and cytogenetic characteristics. However, despite these similarities, differences in the predominant sites ... | 29/29 | 16358311 |
| E-GEOD-2351 | all | chemotherapy cross-resistance and treatment response in childhood acute lymphoblastic leukemia | Acute lymphoblastic leukemia (ALL) can be cured with combination chemotherapy in over 75% of children, but the cause of treatment failure in the remaining patients is unknown. We determined the sensitivity of ALL cells to individual antileukemic agents in 441 patients, and used a genome-wide approach to identify 45 genes differentially expressed in ALL exhibiting cross-resistance to prednisolone, ... | 129/129 | 15837626 |
| E-GEOD-3912 | all | First bone marrow relapse with or without initial diagnosis | This SuperSeries comprises the following subset Series:; GSE3910: 35 patients at diagnosis and relapse; GSE3911: 60 samples obtained at relapse Experiment Overall Design: Refer to individual Series | 113/113 | 16822902 |
| E-GEOD-4698 | all | Molecular characterization of very early relapsed childhood ALL | Purpose: In childhood acute lymphoblastic leukemia (ALL), approximately 25% of patients suffer from relapse. In recurrent disease, despite intensified therapy, overall cure rates of 40% remain unsatisfactory and survival rates are particularly poor in certain subgroups. The probability of long-term survival after relapse is predicted from well-established prognostic factors, i. e. time and site of ... | 60/60 | 16899601 |

Table B.1: Selected ArrayExpress experiments of cancer entity 'all' - part 1.

| ID | Entity | Title | Description | Arrays | PubMed ID |
|---|---|---|---|---|---|
| E-GEOD-635 | all | Identification of novel genomic determinants of cellular drug resistance in acute lymphoblastic leukemia. | Cellular drug resistance is associated with an unfavorable prognosis in pediatric acute lymphoblastic leukemia (ALL). To identify genes conferring resistance to antileukemic agents, we analyzed the expression of >12,700 genes in sensitive and resistant ALL cells obtained at diagnosis from 174 patients. This revealed 42, 59, 54 and 22 genes (P{\~A}{\textcent}{\^A}?{\^A}{\textcurrency}0.001) that were differentially expressed in B-lineag ... | 173/173 | 15295046 |
| E-GEOD-8879 | all | Gene expression profiling of atypical T-ALL | Despite improved therapy, approximately one-fifth of children with acute T-lymphoblastic leukemia (T-ALL) succumb to the disease, suggesting unrecognized biologic heterogeneity that may contribute to drug resistance. We studied leukemic cells, collected at diagnosis, to identify features that could define this high-risk subgroup. A total of 139 patients with T-ALL were treated consecutively from 1 ... | 55/55 | |
| E-MEXP-120 | all | Transcription profiling of bone marrow samples of 31 children with acute lymphoblastic leukemia to identify changes in gene expression that are associated with the current risk assignment, irrespective of the genetic subtype | We analyzed bone marrow samples of 31 children with acute lymphoblastic leukemia to identify changes in gene expression that are associated with the current risk assignment, irrespective of the genetic subtype | 31/31 | 15257931 |
| E-MEXP-313 | all | CIT-TALL-SIGAUX | 104 samples; Affymetrix U133A micro-arrays.<br> <br> Ninety two patients with T-ALL were diagnosed and treated at Saint-Louis hospital, Paris. Seven patients were studied at diagnosis and relapse (total 99 T-ALL samples). There were 56 children (median age 9 years old; range 1 to 16), and 36 adults (median age 27; range 17 to 66). Informed consent was obtained from the patients and/or relatives. T ... | 104/104 | 15774621 |
| E-TABM-125 | all | Translating microarray data for diagnostic testing in childhood leukaemia | We examined published microarray data from 104 acute lymphoblastic leukaemia patient specimens, that represent six different subgroups defined by cytogenetic features and immunophenotypes. Using the decision-tree based supervised learning algorithm Random Forest (RF), we determined a small set of genes for optimal subgroup distinction and subsequently validated their predictive power in an indepen ... | 68/68 | 17002788 |

Table B.2: Selected ArrayExpress experiments of cancer entity 'all' - part 2.

| ID | Entity | Title | Description | Arrays | PubMed ID |
|---|---|---|---|---|---|
| E-GEOD-11121 | breast | The humoral immune system has a key prognostic impact in node-negative breast cancer | Estrogen receptor (ER) expression and proliferative activity are established prognostic factors in breast cancer. In a search for additional prognostic motives we analyzed the gene expression patterns of 200 tumors of patients who were not treated by systemic therapy after surgery using a discovery approach. After performing hierarchical cluster analysis, we identified co-regulated genes related t ... | 200/200 | 18593943 |
| E-GEOD-11965 | breast | Contribution of HSD17B12 for estradiol biosynthesis in human breast cancer | 17beta-hydroxysteroid dehydrogenase type12 (HSD17B12) has been demonstrated to be involved in regulation of in situ biosynthesis of estradiol (E2). HSD17B12 expression was reported in breast carcinomas but its functions have remained unknown. Therefore, we examined the correlation between mRNA expression profiles determined by microarray analysis and tissue E2 concentrations obtained from 16 postm ... | 16/32 | |
| E-GEOD-12093 | breast | The 76-gene Signature Defines High-Risk Patients that Benefit from Adjuvant Tamoxifen Therapy | Classification of tamixifen-treated breast cancer patients into high and low risk groups using the 76-gene signature Experiment Overall Design: 136 breast cancer samples that were treated with tamoxifen were classified using the 76-gene signature | 136/136 | 18821012 |
| E-GEOD-1456 | breast | Gene expression of breast cancer tissue in a large population-based cohort of Swedish patients | Tissue material was collected from all breast cancer patients receiving surgery at Karolinska Hospital from 1994-1996. Material was frozen immediatley on dry ice or in liquid nitrogen and stored in -70{\~A}?{\^A}{\textdegree}C freezers. This series contains expression data for n=159 tumors from which RNA could be collected in sufficient amounts and quality for analysis. Experiment Overall Design: All tumor specimens we ... | 159/318 | 16280042 |
| E-GEOD-1561 | breast | EORTC 10994 clinical trial | EORTC 10994 is a phase III clinical trial comparing FEC with ET in patients with large operable, locally advanced or inflammatory breast cancer (www.eortc.be). Frozen biopsies were taken at randomisation. RNA was extracted from 100um thickness of 14G core needle biopsies. Adjacent sections were tested by H&E to confirm >20% tumour cell content. 100 ng total RNA per chip were amplified using the Af ... | 49/49 | 15897907 |
| E-GEOD-2603 | breast | Subpopulations of MDA-MB-231 and Primary Breast Cancers | Subpopulations of MDA-MB-231 that exhibit different metastatic tropisms when injected into immuno-deficient mice. Also, a cohort of primary breast cancers surgically resected at the Memorial Sloan-Kettering Cancer Center (MSKCC). | 121/121 | 16049480 |
| E-GEOD-2990 | breast | Gene Expression Profiling in Breast Cancer: Understanding the Molecular Basis of Histologic Grade To Improve Prognosis | Background: Histologic grade in breast cancer provides clinically important prognostic information. However, 30%-60% of tumors are classified as histologic grade 2. This grade is associated with an intermediate risk of recurrence and is thus not informative for clinical decision making. We examined whether histologic grade was associated with gene expression profi les of breast cancers and whether ... | 189/189 | 16478745 |

Table B.3: Selected ArrayExpress experiments of cancer entity 'breast' - part 1.

| ID | Entity | Title | Description | Arrays | PubMed ID |
|---|---|---|---|---|---|
| E-GEOD-3494 | breast | An expression signature for p53 in breast cancer predicts mutation status, transcriptional effects, and patient survival | The biological tumor samples (ie, breast tumor specimens) consisted of freshly frozen breast tumors from a population-based cohort of 315 women representing 65% of all breast cancers resected in Uppsala County, Sweden, from January 1, 1987 to December 31, 1989. Estrogen receptor status was determined by biochemical assay as part of the routine clinical procedure. An experienced pathologist determ ... | 251/502 | 16141321 |
| E-GEOD-4917 | breast | Time course microarray data following GR activation in MCF10A-Myc breast cells | This series contain time course microarray data from MCF10A-Myc cells treated with either ethanol or Dexamethasone for 30 min, 2 hr, 4 hr, and 24 hr. This series contains three biological replicates that were analyzed as independent replicate experiments. | 24/24 | 16690749 |
| E-GEOD-4922 | breast | Transcription profiling of human breast cancer tumor samples from Uppsala and Singapore cohorts | Histological grading of breast cancer defines morphological subtypes informative of metastatic potential, although not without considerable inter-observer disagreement and clinical heterogeneity particularly among the moderately differentiated grade II (G2) tumors. We posited that a gene expression signature capable of discerning tumors of grade I (G1) and grade III (G3) histology might provide a ... | 289/578 | 17079448 |
| E-GEOD-5462 | breast | Letrozole (Femara) early response to treatment | In the present investigation, we have exploited the opportunity provided by neoadjuvant treatment of a group of postmenopausal women with large operable or locally advanced breast cancer (in which therapy is given with the primary tumour remaining within the breast) to take sequential biopsies of the same cancers before and after 10-14 days treatment with letrozole. RNA extracted from the biopsie ... | 116/116 | 17885619 |
| E-GEOD-5847 | breast | Tumor and stroma from breast by LCM | Tumor epithelium and surrounding stromal cells were isolated using laser capture microdissection of human breast cancer to examine differences in gene expression based on tissue types from inflammatory and non-inflammatory breast cancer Experiment Overall Design: We applied LCM to obtain samples enriched in tumor epithelium and stroma from 15 IBC and 35 non-IBC cases to study the relative contribu ... | 95/95 | 17999412 |
| E-GEOD-6532 | breast | Transcription profiling of human breast cancers to define clinically distinct molecular subtypes in estrogen receptor positive breast carcinomas using genomic grade | Purpose: A number of microarray studies have reported distinct molecular profiles of breast cancers (BC): basal-like, ErbB2-like and two to three luminal-like subtypes. These were associated with different clinical outcomes. However, although the basal and the ErbB2 subtypes are repeatedly recognized, identification of estrogen receptor (ER)-positive subtypes has been inconsistent. Refinement of t ... | 327/741 | 17401012 |
| E-GEOD-6772 | breast | Comparison of gene expression data from human and mouse breast cancers | This SuperSeries is composed of the following subset Series:; GSE6581: Expression data from mammary glands of transgenic mice; GSE6596: Comparison of gene expression data from human and mouse breast cancers: Identification of conserved breast tumor genes Experiment Overall Design: Refer to individual Series | 26/38 | 17410534 |

Table B.4: Selected ArrayExpress experiments of cancer entity 'breast' - part 2.

| ID | Entity | Title | Description | Arrays | PubMed ID |
|---|---|---|---|---|---|
| E-GEOD-7390 | breast | Strong Time Dependence of the 76-Gene Prognostic Signature | Background: Recently a 76-gene prognostic signature able to predict distant metastases in lymph node-negative (N-) breast cancer patients was reported. The aims of this study conducted by TRANSBIG were to independently validate these results and to compare the outcome with clinical risk assessment. Materials and Methods: Gene expression profiling of frozen samples from 198 N- systemically untreate ... | 198/198 | 17545524 |
| E-GEOD-9574 | breast | Gene expression abnormalities in histologically normal breast epithelium of breast cancer patients | Normal-appearing epithelium of cancer patients can harbor occult genetic abnormalities. Data comprehensively comparing gene expression between histologically normal breast epithelium of breast cancer patients and cancer-free controls are limited. The present study compares global gene expression between these groups. We performed microarrays using RNA from microdissected histologically normal term ... | 29/29 | 18058819 |
| E-GEOD-9936 | breast | Expression data from human breast cancer cells (MCF-7) co-expressing ERalpha and Erbeta, treated with phytoestrogens | We used microarrays to detail the global transcriptional response mediated by ERalpha or ERbeta to the phytoestrogen genistein in the MCF-7 human breast cancer cell model. Experiment Overall Design: MCF-7 human breast cancer cells expressing endogenouse Estrogen Receptor Alpha (ERalpha) were infected with adenovirus carrying either estrogen receptor beta (AdERb) or no insert (Ad) at multiplicity o ... | 105/105 | |
| E-MEXP-440 | breast | Gene expression changes associated with lapatinib treatment of breast cancer cell lines | Dose and time course response of lapatinib in breast cancer cell lines. | 36/36 | 17513611 |
| E-MTAB-7 | breast | MAGETAB Worked Examples DC 2008 | A cell line comparison experiment for breat cancer 51 cancer cell lines | 51/51 | |
| E-TABM-43 | breast | CIT-HS-BREAST-CANCER-CHEMOTHERAPY-RESPONSE | 37 samples hybridized on Affymetrix HG-U133A arrays. Analysis of advanced breast cancers treated with a dose-intense epirubicin /cyclophosphamide regimen followed by mastectomy; Validation of TP53-related genes in breast and bladder cancers. We found that a complete response to chemotherapy was only observed in TP53 mutant tumours. We further show that, among TP53 mutant tumours, high basalcytoker ... | 37/37 | 17388661 |

Table B.5: Selected ArrayExpress experiments of cancer entity ’breast’ - part 3.

| ID | Entity | Title | Description | Arrays | PubMed ID |
|---|---|---|---|---|---|
| E-GEOD-10072 | lung | Transcription profiling of human lung adenocarcinoma and non-tumors from former, current and never smoking individuals | Tobacco smoking is responsible for over 90% of lung cancer cases, and yet the precise molecular alterations induced by smoking in lung that develop into cancer and impact survival have remained obscure. We performed gene expression analysis using HG-U133A Affymetrix chips on 135 fresh frozen tissue samples of adenocarcinoma and paired noninvolved lung tissue from current, former and never smoker ... | 107/107 | |
| E-GEOD-4127 | lung | Anticancer drug clustering in lung cancer based on gene expression profiles and sensitivity database | Anticancer drug clustering in lung cancer based on gene expression profiles. We performed gene expression analysis in lung cancer cell lines. (used: Affymetrix GeneChip Human Genome U133 Array Set HG-U133A). We also examines the sensitivity of these cell lines to commonly used anti-cancer agents (docetaxel, paclitaxel, gemcitabine, vinorelbine, 5-FU, SN38, cisplatin, and carboplatin) via MTT assay ... | 29/29 | |
| E-GEOD-4824 | lung | Analysis of lung cancer cell lines | These arrays are used for various projects Experiment Overall Design: HG-U133A and HG-U133B data are combined and analyzed together with other U133A & B or with HG-U133plus2 samples. No replicates were performed. Controls are human bronchial epithelial cells (HBECs) | 79/164 | 16843264 |
| E-GEOD-6253 | lung | A Gene Expression Signature Predicts Survival of Patients with Stage I Non-Small Cell Lung Cancer | We applied a meta-analysis of datasets from seven different microarray studies on lung cancer for differentially expressed genes related to survival time (under 2 y and over 5 y). Systematic bias adjustment in the datasets was performed by distance-weighted discrimination (DWD). We identified a gene expression signature consisting of 64 genes that is highly predictive of which stage I lung cancer ... | 18/72 | 17194181 |
| E-GEOD-7670 | lung | Expression data from Lung cancer | Detection, treatment, and prediction of outcome for lung cancer patients increasingly depend on a molecular understanding of tumor development and sensitivity of lung cancer to therapeutic drugs. The application of genomic technologies, such as microarray, is widely used to monitor global gene expression and has built up invaluable information and knowledge, which is essential to the discovery of ... | 66/66 | 17540040 |
| E-MEXP-231 | lung | Normal Lung + Lung adenocarcinoma microarray | Gene transcription in a set of 49 human primary lung adenocarcinomas and 9 normal lung tissue samples was examined using Affymetrix GeneChip technology. We aimed to investigate differential gene expression between the two tissue types. A total of 3,442 genes, called the set MAD, were found to be either up- or down-regulated by at least two fold between the two phenotypes. Genes assigned to a parti ... | 58/58 | 15653641 |
| E-TABM-15 | lung | Transcription profiling of cancerous and non cancerous lung adenocarcinoma tissue. Tumour and normal samples from human lung carcinoma from 18 patients plus tumour only from 5 patients | Comparison of gene expression of cancerous and non cancerous lung adenocarcinoma tissue. Tumour and normal samples from 18 patients plus tumour only from 5 patients. | 41/41 | |

Table B.6: Selected ArrayExpress experiments of cancer entity 'lung'.

| ID | Entity | Title | Description | Arrays | PubMed ID |
|---|---|---|---|---|---|
| E-GEOD-2742 | colon | Genomic Strategies Identify the Antitumor Agent Apratoxin A as a Potent Antagonist of FGF Signaling and STAT3 Activation | Total RNA was extracted from apratoxin A or vehicle treated HT29 cells using the RNeasy Mini Kit (Qiagen). Probe values from CEL files were condensed to probe sets using Rosetta Resolver software. Resolver ANOVA analysis was then performed between groups. Experiment Overall Design: 2 doses were compared to vehicle at 3 time points. Each time point had its own vehicle control | 27/27 | 16474387 |
| E-GEOD-4045 | colon | Classification of serrated colorectal tumors | Serrated adenocarcinomas are morphologically different from conventional adenocarcinomas. The serrated pathway has recently been proposed to represent a novel mechanism of colorectal cancer (CRC) formation. However, whether they are biologically different and truly form a distinct subclass of CRC, is not known. This study shows that the gene expression profile of serrated and conventional CRCs dif ... | 37/37 | 16819509 |
| E-MEXP-101 | colon | Narayanan Lab RKO SIM2s Antisense | RKO Colon Carcinoma Cells were treated with 100nM of either SIM2s Control or Antisense oligos in a timecourse (10, 14, 18, 24hr) dependent manner. | 32/32 | 16129820 |
| E-MEXP-383 | colon | Colorectal cancer: UICC II versus UICC III | Analyze differential expression between stage UICC II and UICC III colorectal cancer | 36/36 | 16721809 |
| E-MEXP-833 | colon | Croner_CRC_GBP-1 | The gene expression profile of 24 human colorectal carcinoma patients either expressing high (n=12) or low (n=12) levels of human guanylate binding protein-1 (GBP-1) were compared in order to identify coregulated genes. | 24/24 |  |
| E-MTAB-57 | colon | Transcription profiling of colon cancer tumor and normal biopsies from a series of patients to identify molecular biomarkers | Expression profiling studies on colon cancer comparing tumoral and normal biopsies from a series of patients in order to identify molecular biomarkers. | 47/47 | 16919171 |

Table B.7: Selected ArrayExpress experiments of cancer entity ’colon’.

| ID | Entity | Title | Description | Arrays | PubMed ID |
|---|---|---|---|---|---|
| E-GEOD-2443 | prostate | Prostate cancer - comparison of androgen-dependent and -independent microdissected primary tumor | Affymetrix U133A comparison of two groups (10 samples each): untreated (androgen-dependent) primary prostate cancer (Gleasons 5-9) and androgen-independent primary prostate cancer. All samples were microdissected for tumor cells only. | 20/20 | 16203770 |
| E-GEOD-8218 | prostate | Transcription profiling of human prostate cancer samples | Prostate cancer gene expression profiles were studied in this project. A total RNA from 148 prostate sample with various amount of different cell types were hybridized to Affymetrix U133A arrays. The percentage of different cell types vary considerably among samples and were determined by pathologist. Cell type specific genes can be determined by linear regression using the methods of Stuart et al ... | 148/148 | |
| E-MEXP-1327 | prostate | Selenium vitaminE trial in Prostate Cancer | 85 radical prostatectomy specimens (where 16 samples are in Placebo group (PL), 15 are in Selenium group (SE), 25 are in Vitamin E group (VE) and 27 are in Vitamin E & Selenium group (VS.Treatment groups: l-selenomethionine, 400 ug + placebo (vitamin E); vitamin E, 400 IU + placebo (l-selenomethionine); l-selenomethionine, 400 ug + vitamin E, 400 IU; placebos) were subjected to laser capture micr ... | 85/85 | |
| E-TABM-26 | prostate | CSM-Prostate-Cancer-Samples | Microarray studies of Prostate tissues obtained from multiple Institutions. Analysis done during Aug. 2002 to June 2004. | 57/114 | 16618720 |
| E-TABM-90 | prostate | Transcription profiling of irradiated human lymphyocytes from prostate carcinoma patients following curative radiotherapy to study late radiation toxicity | For a case-control study, we selected 54 prostate carcinoma patients with no evidence of disease 2 years after curative | 108/108 | |

Table B.8: Selected ArrayExpress experiments of cancer entity ’prostate’.

| ID | Entity | Title | Description | Arrays | PubMed ID |
|---|---|---|---|---|---|
| E-GEOD-1159 | aml | Expression profiles of acute myeloid leukemia patient samples | Expression profiles of acute myeloid leukemia patient samples. Blasts and mononuclear cells were purified from bone marrow or peripheral blood aspirates of acute myeloid patients. Samples contained 80-100 percent blast cells. Total RNA was extracted by lyses with guanidium isothiocyanate followed by cesium chloride gradient purification | 293/293 | 17910043 |
| E-GEOD-12417 | aml | Prognostic gene signature for normal karyotype AML | Patients with cytogenetically normal acute myeloid leukemia (CN-AML) show heterogeneous treatment outcomes. We used gene expression profiling to develop a gene signature that predicts overall survival (OS) in CN-AML. Based on data from 163 patients treated in the German AMLCG 1999 trial and analyzed on oligonucleotide microarrays, we used supervised principal component analysis to identify 86 prob ... | 163/326 | 18716133 |
| E-GEOD-1729 | aml | Gene expression profile of acute myeloid leukemia | Gene expression profile of acute myeloid leukemia. Bone marrow (BM) samples from 43 adult patients with newly de novo diagnosed AML.All samples contained more than 80% blast cells. Total RNA was extracted using Trizol reagent (Life Technologies, Gaithersburg, MD) and purified with RNeasy Mini Kit (Quiagen, Valencia, CA). The RNA integrity was assessed using Agilent 2100 Bioanalyzer (Agilent, Palo ... | 43/43 | 15674361 |
| E-GEOD-9476 | aml | Abnormal Expression Changes in AML | Acute myeloid leukemia (AML) is one of the most common and deadly forms of hematopoietic malignancies. We hypothesized that microarray studies could identify previously unrecognized expression changes that only occur only in AML blasts. We were particularly interested in those genes with increased expression in AML, believing that these genes may be potential therapeutic targets. Experiment Overa ... | 64/64 | 17910043 |

Table B.9: Selected ArrayExpress experiments of cancer entity 'aml'.

| ID | Entity | Title | Description | Arrays | PubMed ID |
|---|---|---|---|---|---|
| E-GEOD-11038 | cll | Molecular and transcriptional characterization of chromosome 17p loss in chronic lymphocytic leukemia | Distinct genetic abnormalities such as TP53 deletion at 17p13.1, have been identified as having an adverse prognostic relevance in B-cell chronic lymphocytic leukemia (B-CLL). Conventional cytogenetic studies have shown that TP53 deletion in B-CLL is associated predominantly with 17p loss resulting from complex chromosomal rearrangements. We performed genome-wide DNA (SNPs arrays), fluorescence in ... | 60/72 | 18521849 |
| E-GEOD-6691 | cll | Gene expression profiling of B lymphocytes and plasma cells from Waldenstrom's macroglobulinemia. | The tumoral clone of Waldenstrom{\^A}?s macroglobulinemia (WM) shows a wide morphological heterogeneity which ranges from B-lymphocytes (BL) to plasma cells (PC). By means of genome-wide expression profiling we have been able to identify genes exclusively deregulated in BL and PC from WM, but with a similar expression pattern in their corresponding cell-counterparts from CLL and MM, as well as normal i ... | 56/56 | 17252022 |
| E-GEOD-8835 | cll | Chronic lymphocytic leukemia cells induce changes in gene expression of CD4 and CD8 T cells. | To examine the impact of tumors on the immune system, we compared global gene expression profiles of peripheral blood T cells from previously untreated patients with B cell chronic lymphocytic leukemia (CLL) with those from age-matched healthy donors. Although the cells analyzed were not part of the malignant clone, analysis revealed differentially expressed genes, mainly involved in cell differen ... | 66/66 | 15965501 |

Table B.10: Selected ArrayExpress experiments of cancer entity 'cll'.

| ID | Entity | Title | Description | Arrays | PubMed ID |
|---|---|---|---|---|---|
| E-GEOD-4475 | lymphoma | A Biologic Definition of Burkitt's Lymphoma from Transcriptional and Genomic Profiling | The distinction between the Burkitt lymphoma and diffuse large B-cell lymphoma is imprecise using current diagnostic criteria. We applied transcriptional and genomic profiling to molecularly define Burkitt lymphoma. Gene expression profiling employing Affymetrix GeneChips (U133A) was performed in 220 mature aggressive B-cell lymphomas, including a core group of eight Burkitt lymphomas, which fulfi ... | 221/221 | 16760442 |
| E-GEOD-8388 | lymphoma | Epigenetic upregulation of B-cell inappropriate genes induces extinction of B-cell program in classical Hodgkin lymphoma | A unique feature of the tumour cells (Hodgkin/Reed-Sternberg (HRS)) of classical Hodgkin lymphoma (cHL) is the loss of their B-cell phenotype despite their B-cell origin. Several lines of evidence suggest that epigenomic events, especially promoter DNA-methylation, are involved in this silencing of many B-cell associated genes. Here we show that DNA-demethylation alone or in conjunction with histo ... | 24/24 | |
| E-TABM-117 | lymphoma | CIT_LYMPHOMA_ALCL_ALK_SUBTYPES | Affymetrix UU133A gene expression data for a series of 32 cases of systemic Anaplastic Large Cell Lymphoma<br> (ALCL) and 5 ALCL cell lines; used to (1) confirm that tumors expressing Anaplastic Lymphoma Kinase (ALK+ ALCL) and ALK- ALCLs are different entities, (2) identify most significantly differentially expressed genes between ALK+ and ALK- samples, (3) generate a molecular signature of ALK- A ... | 37/37 | 17077326 |
| E-TABM-346 | lymphoma | CIT-HS-DLBCL-KLY | 53 samples hybridized on Affymetrix HG-U133A GeneChips arrays, for 53 patients with diffuse large B-cell lymphoma (DLBCL); patients are treated with CHOP (cyclophosphamide, doxorubicin, vincristine, prednisone) or Ritxumab (R)-CHOP in the Groupe d{\^A}?Etude des Lymphomes de l{\^A}?Adulte (GELA) clinical centers. | 53/53 | |

Table B.11: Selected ArrayExpress experiments of cancer entity 'lymphoma'.

# Appendix C

# Vignettes

Each Bioconductor package contains at least one vignette, a document that provides a task-oriented description of package functionality. Vignettes contain executable examples and are intended to be used interactively.

You can access the PDF version of a vignette for any installed package from inside R as follows:

```
R> library("affyPara")
R> openVignette(package = "affyPara")
```

This will present you a menu, where you can select the desired vignette. Selecting a vignette should cause the corresponding PDF file to open on your system.

The latest vignettes are available in the latest package versions at the Bioconductor or R-forge repositories.

# Bibliography

[Aff02] Affymetrix. Statistical Algorithms Description Document. Technical report, Affymetrix, Santa Clara, CA, 2002.

[Aff04] Affymetrix. GeneChip Expression Platform: Comparison, Evolution, and Performance. Technical report, Affymetrix, Santa Clara, CA, 2004.

[Amd67] Gene M. Amdahl. Validity of the Single Processor Approach to achieving large scale Computing Capabilities. In *AFIPS '67 (Spring): Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, New York, NY, USA, 1967. ACM.

[Bar02] David Barkan. A parallel Implementation of the Needleman-Wunsch Algorithm for global gapped pair-wise Alignment. *Journal of Computing Sciences in Colleges*, 17(6):238–239, 2002.

[BCS+04] B. M. Bolstad, F. Collin, K. M. Simpson, R. A. Irizarry, and T. P. Speed. Experimental Design and low-level Analysis of Microarray Data. *International Review of Neurobiology*, 60:25–58, 2004.

[BDG03] Daniel P. Berrar, Werner Dubitzky, and Martin Granzow, editors. *A Practical Approach to Microarray Data Analysis.* Kluwer Academic Publishers Group, 2003.

[Ben04] Henrik Bengtsson. aroma - An R Object-oriented Microarray Analysis Environment. *Preprints in Mathematical Sciences, Mathematical Statistics, Lund University*, 18, 2004.

[BHQ+01] A. Brazma, P. Hingamp, J. Quackenbush, G. Sherlock, P. Spellman, C. Stoeckert, J. Aach, W. Ansorge, C. A. Ball, H. C. Causton, T. Gaasterland, P. Glenisson, F. C. Holstege, I. F. Kim, V. Markowitz, J. C. Matese, H. Parkinson, A. Robinson, U. Sarkans, S. Schulze-Kremer, J. Stewart, R. Taylor, J. Vilo, and M. Vingron. Minimum Information about a Microarray Experiment (MIAME) - toward Standards for Microarray Data. *Nature Genetics*, 29(4):365–371, Dec 2001.

[BIAS03] B. M. Bolstad, R. A. Irizarry, M. Astrand, and T. P. Speed. A Comparison of Normalization Methods for high density oligonucleotide Array Data based on Variance and Bias. *Bioinformatics*, 19(2):185–193, Jan 2003.

[Bin08] Andrew Binstock. Threading Models for High-Performance Computing: Pthreads or OpenMP? Intel software network, Pacific Data Works LLC, 2008.

[BK09] Christian Baun and Marcel Kunze. Cloud Computing - Infrastrukturen für Clouds mit Eucalyptus selbst aufbauen. In *iX Magazin für professionelle Informationstechnik*, page S. 128, 2009.

[BKH09] Ulrike Bacher, Alexander Kohlmann, and Torsten Haferlach. Current Status of Gene Expression Profiling in the Diagnosis and Management of Acute Leukaemia. *British Journal of Haematology*, 145(5):555–568, June 2009.

[BL00] Clay P. Breshears and Phu Luong. Comparison of openMP and Pthreads within a Coastal Ocean Circulation Model Code. In *Workshop on OpenMP Applications and Tools*, 2000.

[Bol04] BM Bolstad. *Low Level Analysis of High-density Oligonucleotide Array Data: Background, Normalization and Summarization.* PhD thesis, University of California, Berkeley, 2004.

[BTW$^+$09] Tanya Barrett, Dennis B Troup, Stephen E Wilhite, Pierre Ledoux, Dmitry Rudnev, Carlos Evangelista, Irene F Kim, Alexandra Soboleva, Maxim Tomashevsky, Kimberly A Marshall, Katherine H Phillippy, Patti M Sherman, Rolf N Muertter, and Ron Edgar. NCBI GEO: Archive for high-throughput functional Genomic Data. *Nucleic Acids Research*, 37(Database issue):D885–D890, Jan 2009.

[But97] David R. Butenhof. *Programming with POSIX threads*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[BW09] Alex Bateman and Matt Wood. Cloud computing. *Bioinformatics (Oxford, England)*, May 2009.

[BWB09] Marty C Brandon, Douglas C Wallace, and Pierre Baldi. Data Structures and Compression Algorithms for Genomic Sequence Data. *Bioinformatics*, 25(14):1731–1738, Jul 2009.

[CSP$^+$06] Iona Cheng, Daniel O Stram, Kathryn L Penney, Malcolm Pike, Loic Le Marchand, Laurence N Kolonel, Joel Hirschhorn, David Altshuler, Brian E Henderson, and Matthew L Freedman. Common genetic Variation in IGF1 and Prostate Cancer Risk in the Multiethnic Cohort. *Journal of the National Cancer Institute*, 98(2):123–134, Jan 2006.

[DFF$^+$03] Jack Dongarra, Ian Foster, Geoffrey Fox, William Gropp, Ken Kennedy, Linda Torczon, and Andy White, editors. *Sourcebook of Parallel Computing.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[DM98] L. Dagum and R. Menon. OpenMP: an industry Standard API for shared-memory Programming. *IEEE Computational Science and Engineering*, 5(1):46–55, Jan.–March 1998.

[FBC$^+$07] Francesco Ferrari, Stefania Bortoluzzi, Alessandro Coppe, Alexandra Sirota, Marilyn Safran, Michael Shmoish, Sergio Ferrari, Doron Lancet, Gian Antonio Danieli, and Silvio Bicciato. Novel definition files for human GeneChips based on GeneAnnot. *BMC Bioinformatics*, 8:446, 2007.

[FHT08] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance Estimation with the Graphical Lasso. *Biostatistics*, 9(3):432–441, Jul 2008.

[For98] Message Passing Interface Forum. MPI2: A Message Passing Interface Standard. In *International Journal of High Performance Computing Applications*, volume 12, pages 1–299, 1998.

[FTS$^+$95] Y. Furukawa, Y. Terui, K. Sakoe, M. Ohta, S. Kitagawa, Y. Miura, and M. Saito. Over-expression and Amplification of the CDC2 Gene in Leukaemia Cells. *British Journal of Haematology*, 90(1):94–99, May 1995.

[GBD$^+$94] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, B. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Network Parallel Computing.* MIT Press, Cambridge, 1994.

[GCB$^+$04] Robert C Gentleman, Vincent J Carey, Douglas M Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, Kurt Hornik, Torsten Hothorn, Wolfgang Huber, Stefano Iacus, Rafael Irizarry, Friedrich Leisch, Cheng Li, Martin Maechler, Anthony J Rossini, Gunther Sawitzki, Colin Smith, Gordon Smyth, Luke Tierney, Jean YH Yang, and Jianhua Zhang. Bioconductor: Open Software Development for computational Biology and Bioinformatics. *Genome Biology*, 5, 2004.

[GCBI04] Laurent Gautier, Leslie Cope, Benjamin M Bolstad, and Rafael A Irizarry. **affy** – Analysis of Affymetrix GeneChip Data at the Probe Level. *Bioinformatics*, 20(3):307–315, Feb 2004.

[GCH$^+$05] Robert Gentleman, Vincent Carey, Wolfgang Huber, Rafael Irizarry, and Sandrine Dudoit. *Bioinformatics and Computational Biology Solutions Using R and Bioconductor*. Springer, 1 edition, August 2005.

[Gen08] Robert Gentleman. *R Programming for Bioinformatics*, volume 12 of *Chapman & Hall/CRC Computer Science & Data Analysis*. Chapman & Hall/CRC, 2008.

[GGL01] M. Gardiner-Garden and T. G. Littlejohn. A Comparison of Microarray Databases. *Brief Bioinform*, 2(2):143–158, May 2001.

[GH04] J.E. Gentle and Wolfgang HSrdle. *Handbook of Computational Statistics*. Springer, 2004.

[GKKG03] Ananth Grama, George Karypis, Vipin Kumar, and Anshul Gupta. *Introduction to Parallel Computing (2nd Edition)*. Addison Wesley, January 2003.

[GL02] William Gropp and Ewing Lusk. Goals Guiding Design: PVM and MPI. In *CLUSTER '02: Proceedings of the IEEE International Conference on Cluster Computing*, page 257, Washington, DC, USA, 2002. IEEE Computer Society.

[HCO06] Sepp Hochreiter, Djork-Arne Clevert, and Klaus Obermayer. A new Summarization Method for Affymetrix Probe Level Data. *Bioinformatics*, 22(8):943–949, Apr 2006.

[HGJY01] A. Hartemink, D. Gifford, T. Jaakkola, and R. Young. Maximum Likelihood Estimation of optimal Scaling Factors for Expression Array Normalization. *SPIE BIOS*, 2001.

[HHGF08] Florian Hahne, Wolfgang Huber, Robert Gentleman, and Seth Falcon. *Bioconductor Case Studies*. Springer Publishing Company, Incorporated, 2008.

[HJ08] Robert A Holt and Steven J M Jones. The new Paradigm of Flow Cell Sequencing. *Genome Research*, 18(6):839–846, Jun 2008.

[HMM08] Manuela Hummel, Reinhard Meister, and Ulrich Mansmann. **GlobalANCOVA**: Exploration and Assessment of Gene Group Effects. *Bioinformatics*, 24(1):78–85, Jan 2008.

[HvHS$^+$02] Wolfgang Huber, Anja von Heydebreck, Holger S̈ltmann, Annemarie Poustka, and Martin Vingron. Variance Stabilization applied to Microarray Data Calibration and to the Quantification of Differential Expression. *Bioinformatics*, 18 Suppl 1:S96–104, 2002.

[HW09] Johanna Hardin and Jason Wilson. A Note on oligonucleotide Expression Values not being Normally Distributed. *Biostatistics*, 10(3):446–450, Jul 2009.

[IAB$^+$09] John P A Ioannidis, David B Allison, Catherine A Ball, Issa Coulibaly, Xiangqin Cui, Aedin C Culhane, Mario Falchi, Cesare Furlanello, Laurence Game, Giuseppe Jurman, Jon Mangion, Tapan Mehta, Michael Nitzberg, Grier P Page, Enrico Petretto, and Vera van Noort. Repeatability of published Microarray Gene Expression Analyses. *Nature Genetics*, 41(2):149–155, Feb 2009.

[IBC$^+$03] Rafael A Irizarry, Benjamin M Bolstad, Francois Collin, Leslie M Cope, Bridget Hobbs, and Terence P Speed. Summaries of Affymetrix GeneChip Probe Level Data. *Nucleic Acids Research*, 31(4):e15, Feb 2003.

[IHC$^+$03] Rafael A Irizarry, Bridget Hobbs, Francois Collin, Yasmin D Beazer-Barclay, Kristen J Antonellis, Uwe Scherf, and Terence P Speed. Exploration, Normalization, and Summaries of high density oligonucleotide Array Probe Level Data. *Biostatistics*, 4(2):249–264, Apr 2003.

[IWS$^+$05] Rafael A Irizarry, Daniel Warren, Forrest Spencer, Irene F Kim, Shyam Biswal, Bryan C Frank, Edward Gabrielson, Joe G N Garcia, Joel Geoghegan, Gregory Germino, Constance Griffin, Sara C Hilmer, Eric Hoffman, Anne E Jedlicka, Ernest Kawasaki, Francisco Martinez-Murillo, Laura Morsberger, Hannah Lee, David Petersen, John Quackenbush, Alan Scott, Michael Wilson, Yanqin Yang, Shui Qing Ye, and Wayne Yu. Multiple-laboratory Comparison of Microarray Platforms. *Nature Methods*, 2(5):345–350, May 2005.

[JKSW99] G. S. Jimenez, S. H. Khan, J. M. Stommel, and G. M. Wahl. p53 Regulation by post-translational Modification and Nuclear Retention in Response to diverse Stresses. *Oncogene*, 18(53):7656–7665, Dec 1999.

[JLR07] W. Evan Johnson, Cheng Li, and Ariel Rabinovic. Adjusting Batch Effects in Microarray Expression Data using empirical Bayes Methods. *Biostatistics*, 8(1):118–127, Jan 2007.

[KB07] Markus Kalisch and Peter Buhlmann. Estimating High-Dimensional Directed Acyclic Graphs with the PC-Algorithm. *Journal of Machine Learning Research*, 8:613–636, 2007.

[KF90] Alan H. Karp and Horace P. Flatt. Measuring parallel Processor Performance. *Communications of the ACM*, 33(5):539–543, 1990.

[KGH09] Audrey Kauffmann, Robert Gentleman, and Wolfgang Huber. **arrayQualityMetrics** – a Bioconductor Package for Quality Assessment of Microarray Data. *Bioinformatics*, 25(3):415–416, Feb 2009.

[KS08] Dennis Kostka and Rainer Spang. Microarray based Diagnosis Profits from better Documentation of Gene Expression Signatures. *PLoS Computational Biology*, 4(2):e22, Feb 2008.

[LGG01] N. M. Luscombe, D. Greenbaum, and M. Gerstein. What is Bioinformatics? A proposed Definition and Overview of the Field. *Methods of Information in Medicine*, 40(4):346–358, 2001.

[LMS09] Yongchao Liu, Douglas L Maskell, and Bertil Schmidt. CUDASW++: optimizing Smith-Waterman Sequence Database Searches for CUDA-enabled Graphics Processing Units. *BMC Research Notes*, 2:73, 2009.

[LNZ+94] Peihuang Lu, Jorge Nocedal, Ciyou Zhu, Richard H. Byrd, and Richard H. Byrd. A Limited-Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific Computing*, 16:1190–1208, 1994.

[LR01] Michael Na Li and A.J. Rossini. **RPVM**: Cluster Statistical Computing in R. *R News*, 1(3):4–7, September 2001.

[LW01] C. Li and W. H. Wong. Model-based Analysis of Oligonucleotide Arrays: Expression Index Computation and Outlier Detection. *Proceedings of the National Academy of Sciences*, 98(1):31–36, Jan 2001.

[Mar08] Elaine R Mardis. The impact of next-generation Sequencing Technology on Genetics. *Trends in Genetics*, 24(3):133–141, Mar 2008.

[MEC08] Marta V. Modenesi, Alexandre G. Evsukoff, and Myrian C. Costa. A Load Balancing Knapsack Algorithm for Parallel Fuzzy c-Means Cluster Analysis. pages 269–279, 2008.

[Meu09] Hans Meuer. *Scientific Computing World*, chapter The Future for HPC, page 62. www.scientific-computing.com, June/July 2009.

[MM05] U. Mansmann and R. Meister. Testing Differential Gene Expression in functional Groups. Goeman's Global Test versus an ANCOVA Approach. *Methods of Information in Medicine*, 44(3):449–453, 2005.

[Moo65] G. E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8):114–117, April 1965.

[Mut00] S. Muthukrishnan. Simple Optimal Parallel Multiple Pattern Matching. *Journal of Algorithms*, 34(1):1 – 13, 2000.

[MV08] Svetlin A Manavski and Giorgio Valle. CUDA compatible GPU cards as efficient Hardware Accelerators for Smith-Waterman Sequence Alignment. *BMC Bioinformatics*, 9 Suppl 2:S10, 2008.

[Nvi09] Nvidia. *CUDA 2.2 Programming Guide*, 2009.

[Pea06] Helen Pearson. Genetics: what is a Gene? *Nature*, 441(7092):398–401, May 2006.

[PKK$^+$09] Helen Parkinson, Misha Kapushesky, Nikolay Kolesnikov, Gabriella Rustici, Mohammad Shojatalab, Niran Abeygunawardena, Hugo Berube, Miroslaw Dylag, Ibrahim Emam, Anna Farne, Ele Holloway, Margus Lukk, James Malone, Roby Mani, Ekaterina Pilicheva, Tim F Rayner, Faisal Rezwan, Anjan Sharma, Eleanor Williams, Xiangqun Zheng Bradley, Tomasz Adamusiak, Marco Brandizi, Tony Burdett, Richard Coulson, Maria Krestyaninova, Pavel Kurnosov, Eamonn Maguire, Sudeshna Guha Neogi, Philippe Rocca-Serra, Susanna-Assunta Sansone, Nataliya Sklyar, Mengyao Zhao, Ugis Sarkans, and Alvis Brazma. ArrayExpress Update – from an Archive of functional genomics Experiments to the Atlas of Gene Expression. *Nucleic Acids Research*, 37(Database issue):D868–D872, Jan 2009.

[PKS$^+$07] H. Parkinson, M. Kapushesky, M. Shojatalab, N. Abeygunawardena, R. Coulson, A. Farne, E. Holloway, N. Kolesnykov, P. Lilja, M. Lukk, R. Mani, T. Rayner, A. Sharma, E. William, U. Sarkans, and A. Brazma. ArrayExpress – a public Database of Microarray Experiments and Gene Expression Profiles. *Nucleic Acids Research*, 35(Database issue):D747–D750, Jan 2007.

[R D08a] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[R D08b] R Development Core Team. Writing r extensions. Technical report, R Foundation for Statistical Computing, Vienna, Vienna, Austria, October 2008. ISBN 3-900051-11-9.

[RD01] D. M. Rocke and B. Durbin. A Model for Measurement Error for Gene Expression Arrays. *Journal of Computational Biology*, 8(6):557–569, 2001.

[RHPM04] Markus Ruschhaupt, Wolfgang Huber, Anne-Marie Poustka, and Ulrich Mansmann. A Compendium to ensure computational Reproducibility in high-dimensional Classification Tasks. *SAGMB*, 3:Article37, 2004.

[RMHA08] Adaikalavan Ramasamy, Adrian Mondry, Chris C Holmes, and Douglas G Altman. Key Issues in Conducting a Meta-Analysis of Gene Expression Microarray Datasets. *PLoS Medicine*, 5(9):e184, Sep 2008.

[RS00] T. Rognes and E. Seeberg. Six-fold Speed-up of Smith-Waterman Sequence Database Searches using Parallel Processing on Common Microprocessors. *Bioinformatics*, 16(8):699–706, Aug 2000.

[RTL03] Anthony Rossini, Luke Tierney, and Na (Michael) Li. Simple Parallel Statistical Computing in R. *UW Biostatistics Working Paper Series*, 193, 2003.

[RTL07] Anthony J Rossini, Luke Tierney, and Na (Michael) Li. Simple Parallel Statistical Computing in R. *Journal of Computational and Graphical Statistics*, 16(2):399–420, 2007.

[Rus08] Markus Ruschhaupt. Erzeugung von positiv definiten Matrizen mit Nebenbedingungen zur Validierung von Netzwerkalgorithmen für Microarray-Daten. *LMU Munich*, 2008.

[Sev03] Hana Sevcikova. Statistical Simulations on Parallel Computers. *Journal of Computational and Graphical Statistics*, 13(4):886–906, 2003.

[SGS01] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*, volume 2nd Edition. MIT Press, 2001.

[She08] Jay Shendure. The Beginning of the End for Microarrays? *Nature Methods*, 5(7):585–587, Jul 2008.

[SJ08] Jay Shendure and Hanlee Ji. Next-generation DNA Sequencing. *Nature Biotechnology*, 26(10):1135–1145, Oct 2008.

[Slo04] Joseph Sloan. *High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI (Nutshell Handbooks)*. O'Reilly Media, Inc., November 2004.

[SM08] Markus Schmidberger and Ulrich Mansmann. Parallelized Preprocessing Algorithms for high-density oligonucleotide Arrays. In *Proc. IEEE International Symposium on Parallel and Distributed Processing IPDPS 2008*, pages 1–7, 14–18 April 2008.

[SM09] Markus Schmidberger and Ulrich Mansmann. **affyPara** - a Bioconductor Package for Parallelized Preprocessing Algorithms of Affymetrix Microarray Data. *Bioinformatics and Biology Insights*, 3, 2009.

[SME$^+$09] Markus Schmidberger, Martin Morgan, Dirk Eddelbuettel, Hao Yu, Luke Tierney, and Ulrich Mansmann. State of the Art in Parallel Computing with R. *Journal of Statistical Software*, 31(1), 2009.

[Smy04] Gordon K Smyth. Linear Models and empirical Bayes Methods for Assessing Differential Expression in Microarray Experiments. *Statistical Applications in Genetics and Molecular Biology*, 3:Article3, 2004.

[SNC77] F. Sanger, S. Nicklen, and A. R. Coulson. DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*, 74(12):5463–5467, Dec 1977.

[SS05] Juliane Schäfer and Korbinian Strimmer. An empirical Bayes Approach to inferring large-scale Gene Association Networks. *Bioinformatics*, 21(6):754–764, Mar 2005.

[STDV07] Michael C Schatz, Cole Trapnell, Arthur L Delcher, and Amitabh Varshney. High-throughput Sequence Alignment using Graphics Processing Units. *BMC Bioinformatics*, 8:474, 2007.

[Ste03] Dov Stekel. *Microarray Bioinformatics*. Cambridge University Press, 2003.

[Tie07] Luke Tierney. ***proftools**: Profile Output Processing Tools for R*. R package, 2007.

[Urb09] Simon Urbanek. ***multicore**: Parallel processing of R code on machines with multiple cores or CPUs*, 2009. R package version 0.1-3.

[VDD09] Karl V Voelkerding, Shale A Dames, and Jacob D Durtschi. Next-generation Sequencing: from basic Research to Diagnostics. *Clinical Chemistry*, 55(4):641–658, Apr 2009.

[VDV$^+$09] Ilhami Visne, Erkan Dilaveroglu, Klemens Vierlinger, Martin Lauss, Ahmet Yildiz, Andreas Weinhaeusel, Christa Noehammer, Friedrich Leisch, and Albert Kriegner. **RGG**: A general GUI Framework for R scripts. *BMC Bioinformatics*, 10(1):74, 2009.

[Ven01] W. Venables. *Programmer's Niche. R News, 1(1):27-30*, 2001.

[Vic09] Esmeralda Vicedo. Quality Assessment of Huge Numbers of Affymetrix Microarray Data. Master's thesis, University of Munich, 2009.

[Vou08] M. A. Vouk. Cloud Computing: Issues, Research and Implementations. In *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*, pages 31–40. 2008.

[VSBH08] Fanny Villers, Brigitte Schaeffer, Caroline Bertin, and Sylvie Huet. Assessing the validity Domains of Ggraphical Gaussian Models in order to infer Relationships among Components of Complex Biological Systems. *Statistical Applications in Genetics and Molecular Biology*, 7(1):Article 14, 2008.

[Wic08] H. Wickham. ***profr**: An alternative Display for Profiling Information*. R package, 2008.

[WIG$^+$06] Zhijin Wu, Rafael A. Irizarry, Robert Gentleman, Francisco Martinez-Murillo, and Forrest Spencer. A Model-Based Background Adjustment for Oligonucleotide Expression Arrays. *Journal of the American Statistical Association*, 99(468):909+, 2006.

[WM05] Claire L Wilson and Crispin J Miller. **Simpleaffy**: a BioConductor Package for Affymetrix Quality Control and Data Analysis. *Bioinformatics*, 21(18):3683–3685, Sep 2005.

[Xio06] Jin Xiong. *Essential Bioinformatics.* Cambridge University Press, 2006.

[YDL$^+$02] Yee Hwa Yang, Sandrine Dudoit, Percy Luu, David M Lin, Vivian Peng, John Ngai, and Terence P Speed. Normalization for cDNA Microarray Data: a Robust Composite Method addressing single and multiple slide systematic Variation. *Nucleic Acids Research*, 30(4):e15, Feb 2002.

[Yu02] Hao Yu. **Rmpi**: Parallel Statistical Computing in R. *R News*, 2(2):10–14, June 2002.

[ZB08] Daniel R Zerbino and Ewan Birney. Velvet: Algorithms for de novo short read Assembly using de Bruijn Graphs. *Genome Research*, 18(5):821–829, May 2008.

# Acknowledgment

I owe thanks to many people who supported and encouraged me during the last three years while I was working on my PhD thesis. First of all, I would like to thank my supervisor, Prof. Dr. Ulrich Mansmann, for giving me the opportunity to write this thesis, for allowing me the freedom to pursue my own ideas, for helpful input and discussions, and especially for supporting my three months as a visiting scientist at Robert Gentlemans' group in the Fred Hutchinson Cancer Research Center (Seattle, WA, USA). I also thank Robert Gentleman, all members of Robert Gentlemans' group, and all my colleagues for their readiness to help and cooperate.

Furthermore, I like to thank Prof. Dr. Friedrich Leisch and Prof. Dr. Hao Yu for reviewing this thesis and to Prof. Volker Heun and Dr. Achim Tresch for being part of my dissertation committee. Many thanks go to Florian Hahne and Dirk Eddelbuettel for several interesting discussions and ideas about parallel computing.

I am indebted to my parents for always supporting me and being proud of me and I am very grateful to my wife for all her love, support and reminding me that there is a life beyond the thesis.

# Curriculum Vitae

**Markus Schmidberger** was born April 26th, 1981 in Weilheim, Germany. He received his first degree (Abitur) at the "Gymnasium Weilheim" in 2000. Since 2001 he lives in Munich, Germany.

From 2001 to 2006 he studied Technomathematics with a minor to numerical mathematics, computer sciences, and machine engineering at the Technical University of Munich. His diploma thesis was about 'Partitioned methods for solving fluid-structure-interaction'.

Since August 2006, Markus is working as a PhD student at the Institute of Medical Informatics, Biometrics and Epidemiology (IBE) headed by Prof. Dr. Ulrich Mansmann at the Ludwig-Maximilians-Universität Munich. From October to December 2008 he was a visiting scientist at the Computational Biology Program headed by PhD Robert Gentleman at the Fred Hutchinson Cancer Research Center in Seattle, WA, USA.

His research interest includes high-performance computing and statistics for high-dimensional (biological) problems.

Markus Schmidberger is married. Besides his professional interests, he prefers to go in sports, especially snowboarding and windsurfing.