Universiteit
Leiden
The Netherlands

Carbon starvation in the filamentous fungus Aspergillus niger

Nitsche, B.M.

Citation

Nitsche, B. M. (2012, October 23). *Carbon starvation in the filamentous fungus Aspergillus niger*. Retrieved from https://hdl.handle.net/1887/20011

Version:	Not Applicable (or Unknown)
License:	Leiden University Non-exclusive license
Downloaded from:	https://hdl.handle.net/1887/20011

Note: To cite this publication please use the final published version (if applicable).

Cover Page

Universiteit Leiden

The handle http://hdl.handle.net/1887/20011 holds various files of this Leiden University dissertation.

Author: Nitsche, Benjamin Manuel
Title: Carbon starvation in the filamentous fungus Aspergillus niger
Date: 2012-10-23

Chapter 2

The use of open source bioinformatics tools to dissect transcriptomic data

Benjamin M. Nitsche[1], Arthur F.J. Ram[1,2], Vera Meyer[1,2,3]

[1]Institute of Biology, Leiden University, Sylviusweg 72, 2333 BE Leiden, The Netherlands
[2]Kluyver Centre for Genomics of Industrial Fermentation, PO Box 5057, 2600 GA, Delft, The Netherlands
[3]Institute of Biotechnology, Berlin University of Technology, Gustav-Meyer-Allee 25, 13355 Berlin, Germany

Abstract

Microarrays are a valuable technology to study fungal physiology on a transcriptomic level. Various microarray platforms are available comprising both single and two channel arrays. Despite different technologies, preprocessing of microarray data generally includes quality control, background correction, normalization and summarization of probe level data. Subsequently, depending on the experimental design, diverse statistical analysis can be performed, including the identification of differentially expressed genes and the construction of gene co-expression networks. We describe how Bioconductor, a collection of open source and open development packages for the statistical programming language R, can be used for dissecting microarray data. We provide fundamental details that facilitate the process of getting started with R and Bioconductor. Using two publicly available microarray datasets from *Aspergillus niger*, we give detailed protocols on how to identify differentially expressed genes and how to construct gene co-expression networks.

Methods Mol Biol. 2012;835:311-31

Introduction

The open source and open development project Bioconductor (Gentleman *et al.*, 2004) (http://bioconductor.org/) is actively developed and maintained by members of the academic community, thus providing scientists with leading edge computational biology tools. Bioconductor is well suited for the academic environment where it can be used for research and education. Rather than providing a graphical user interface, most Bioconductor packages depend on command line input. Therefore, getting started with Bioconductor requires some effort, especially for those having limited computational background.

In the following, we give a step-by-step tutorial on how Bioconductor can be used for transcriptomic data analysis and provide the reader with the most important theoretical background on the statistics involved. We use two Affymetrix microarray datasets that were recently published for *Aspergillus niger* (Jørgensen *et al.*, 2010; Martens-Uzunova *et al.*, 2006). The datasets as well as all Bioconductor packages are publicly available, allowing the reader to repeat each step of the analysis. We start with a brief description on how the statistical programming language R (R-Team, 2008) (http://www.r-project.org/), Bioconductor core packages and additional Bioconductor packages can be installed. For the identification of differentially expressed genes, we demonstrate how to import CEL files and associate them with phenotypic labels, how to preprocess microarray data and asses its quality, how to perform multi-way comparisons and finally, how to export the data. For the construction of gene co-expression networks, we subsequently import CEL files, assess the data quality and preprocess the CEL files, perform non-specific filtering and construct gene co-expression networks.

Materials

In the examples described thereafter, the open source and open development packages from the Bioconductor project (Gentleman *et al.*, 2004) (http://bioconductor.org/), which use the statistical programming language R (R-Team, 2008) (http://www.r-project.org/), are used for the analysis of transcriptomic data. The following packages available from the Bioconductor homepage are required: affy (Gautier *et al.*, 2004), affycoretools (MacDonald, 2008), affyPLM (Bolstad *et al.*, 2005), limma (Smyth, 2004), genefilter (Gentleman *et al.*, 2006) and makecdfenv (Irizarry *et al.*, 2006).

Both transcriptomic datasets used for demonstration purposes were recently published for *A. niger* and have been deposited at public databases. The first dataset (Jørgensen *et al.*, 2010) used for the identification of differentially expressed genes comprises nine Affymetrix microarrays corresponding to three different time points during maltose-limited retentostat cultivations and is available via its accession number GSE21752 at the NCBI Gene Expression

Omnibus (GEO) database (http://www.ncbi.nlm.nih.gov/geo/) (Barrett *et al.*, 2011). The second dataset (Martens-Uzunova *et al.*, 2006) used for the construction of gene co-expression networks comprises 29 Affymetrix microarrays from multiple time points after transfer of mycelial biomass from fructose to a variety of carbon sources: rhamnose, xylose, sorbitol, galacturonic acid, polygalacturonic acid and sugar beet pectin. The dataset is available at the EMBL ArrayExpress database (http://www.ebi.ac.uk/arrayexpress/) via the accession number E-MEXP-1626.

Both transcriptomic datasets are based on the Affymetrix chip dsmM_ANIGERa_coll 511030F. The corresponding chip description file (CDF) can be downloaded from the NCBI GEO database via the GEO accession number GPL6758.

Methods

Installation

Before installing Bioconductor packages, the statistical programming language R has to be installed. R installation packages and platform specific help files are available for Linux, Mac OS and Windows at the Comprehensive R Archive Network (CRAN: http://www.r-project.org/). After successful installation, the R command window can be accessed via its application icon or by typing R at the command prompt (see **Note 1**). To install the Bioconductor core packages, make sure to have an Internet connection and type at the R command window (see **Note 2**):

```
> source("http://bioconductor.org/biocLite.R")
> biocLite()
```

Further packages have to be installed:

```
> biocLite("affycoretools")
> biocLite("makecdfenv")
> biocLite("limma")
```

For Affymetrix chips, analysis of raw data usually starts with CEL files. They contain the information from intensity calculations of the pixel values from the raw image data (DAT files) obtained with the chip scanner. To be able to import the Affymetrix raw data from CEL files, Bioconductor requires information about the corresponding Affymetrix chip which has to be provided as a chip-specific package. While those packages can be downloaded for many popular Affymetrix chips, they first have to be built for custom-made arrays such as the *A. niger*

chip. The Affymetrix chip description file (CDF) provides all information required for building the package.

To build the CDF package, first download the dsmM_ANIGERa_coll511030F CDF archive file via its GEO accession number GPL6758 from the GEO database (http://www.ncbi.nlm.nih.gov/geo/) and decompress it (see **Note 3**). Create a new directory (from now on referred to as working directory, WD), copy the CDF file to the WD and rename it into "dsmM_ANIGERa_anColl.CDF". At the R command line, define the WD with the function setwd (see **Note 4** and **Note 5**), load the required package makecdfenv with the library command and build the chip specific package with the make.cdf.package function:

```
> setwd("absolute path to WD/")
> library("makecdfenv")
> make.cdf.package("dsmM_ANIGERa_anColl.CDF", species = "Aspergillus niger
  ")
```

The newly built package is saved in a new folder in the WD (WD/dsmmanigeraancollcdf/). Next, open a new command prompt window, change to the WD and install the package (see **Note 1**):

```
> R CMD INSTALL dsmmanigeraancollcdf
```

Computation of differentially expressed genes

The identification of differentially expressed genes will exemplarily be shown on a transcriptomic dataset recently published for *A. niger* (Jørgensen *et al.*, 2010). The data has been deposited at the NCBI GEO database and can be downloaded via its accession number: GSE21752. *A. niger* was cultivated under controlled conditions in carbon-limited retentostat cultures, which is a specific form of continuous cultivation. The biomass was retained in the bioreactor, while old medium was removed and fresh defined medium was fed at a constant rate. With increasing cultivation time, the biomass specific availability of the sole limiting carbon source maltose decreased and the fungus underwent carbon starvation. The RNA used for microarray hybridizations derived from three different time points: Batch phase referred to as day 0 (d0), day 2 and day 8 of retentostat cultivation (d2 and d8, respectively). With three replicate cultures, the number of microarrays adds up to a total of nine. Below, we describe how to identify sets of genes that are differentially expressed comparing d2 versus d0, d8 versus d0 and d8 versus d2.

Importing CEL files into R

Create a new WD and download the raw data (CEL files) from the NCBI GEO database (http://www.ncbi.nlm.nih.gov/geo/) via the dataset accession number GSE21752. Next, decompress (see **Note 3**) the data and copy the CEL files into the WD. Open the R command line, load the packages that will be used for data analysis with the library function and define the WD using the function setwd (see **Note 4** and **Note 5**):

```
> library("affy")
> library("affycoretools")
> library("affyPLM")
> library("limma")
> setwd("absolute path to WD/")
```

Before importing the CEL files into R, generate a file that allocates phenotypic labels to them. It consists of tab-separated columns and should be saved in the WD as a plain text file named "phenotypic_labels.txt". The required sample information can be obtained from the NCBI GEO database via the corresponding dataset accession number (GSE21752). The content of the phenotypic label file should look like:

```
sample    FileName          Target
d0.1      GSM542228.CEL     d0
d0.2      GSM542335.CEL     d0
d0.3      GSM542336.CEL     d0
d2.1      GSM542337.CEL     d2
d2.2      GSM542338.CEL     d2
d2.3      GSM542339.CEL     d2
d8.1      GSM542340.CEL     d8
d8.2      GSM542341.CEL     d8
d8.3      GSM542342.CEL     d8
```

Use the function read.table to import the text file and assign (see **Note 6**) the phenotypic labels to the variable pData, which represents a data frame with row names equal to the unique file names. Subsequently, call the function ReadAffy to import the raw data from all annotated CEL files:

```
> pData <- read.table("phenotypic_labels.txt", row.names = 2, header =
  TRUE)
> rawData <- ReadAffy(filenames = rownames(pData), verbose = TRUE)
```

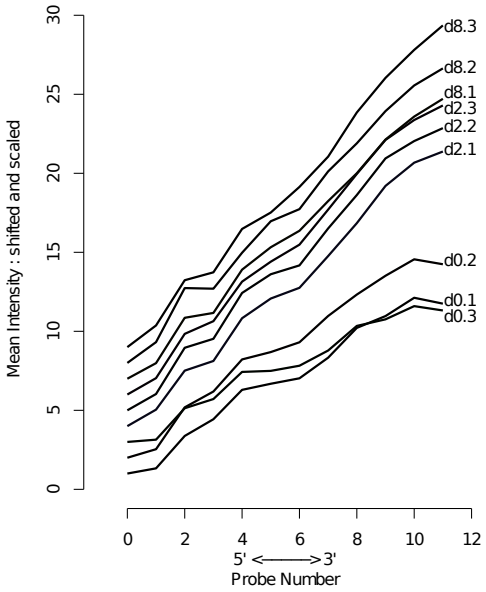

Figure 2.1 — RNA degradation plot
For each array of the retentostat dataset (GSE21752), mean probe intensities are plotted ordered from 5' to 3' end. For assessment of RNA quality, it is important that the 5'/3' intensity ratios are comparable between samples rather than the actual values. Typical 5'/3' ratios depend on the Affymetrix chip used (Bolstad *et al.*, 2005; MacDonald, 2008).

Quality assessment and preprocessing

We recommend checking the average integrity of RNA transcripts, because RNA is very sensitive to degradation by ribonucleases. For this purpose, make a degradation plot showing mean intensities of the probes ordered from 5' to 3' end of their target transcripts. The 5'/3' ratio of mean probe intensities should be comparable between the samples, as shown in Figure 2.1 (see **Note 7**). The following code will create a degradation plot (see **Note 8**):

```
> plotDeg(rawData, filenames = pData$sample)
```

Next, use the Robust Multi-array Average (RMA) package (Irizarry *et al.*, 2003) for preprocessing probe level data. The function rma performs background correction, quantile normalization and robust median polish to summarize intensities of probe sets to expression values. Background correction aims to differentiate specific from non-specific hybridization signals. For this purpose, Affymetrix chips are designed to provide pairs of nearly identical probes for hybridization, so called perfect match (PM) and mismatch (MM) probes. The MM probes are identical to the PM probes, except for their central nucleotide at position 13, which is

changed to its complement. The Affymetrix MAS5.0 algorithm for background correction uses the intensities of both probes to correct for non-specific hybridization signals. However, it has been shown that MM probes in many cases give stronger signals than their corresponding PM probes indicating that besides non-specific also specific hybridization occurs (Naef *et al.*, 2002). Therefore, RMA does not use MM probe intensities but applies a global empirical model for the distribution of PM probe intensities for background correction. Especially for lowly expressed genes, the RMA algorithm for background correction has been shown to be superior to MAS5.0 (Irizarry *et al.*, 2003). Before normalization, differences in probe intensities between arrays are due to biological and technical differences. The later is mainly introduced during labeling and hybridization, which normalization aims to compensate for. After normalization, differences in probe intensities between arrays should in theory exclusively be due to biological differences. The quantile normalization performed by RMA makes probe intensity distributions comparable between the set of arrays under investigation. Finally, RMA fits a robust multi-array model to summarize the probe intensities of each probe set into an expression value. Use the rma function to preprocess the probe level data and compute expression values. The results are organized as an object of the class ExpressionSet. To allocate phenotypic labels to the ExpressionSet object eset, call the function new:

```
> eset <- rma(rawData)
> phenoData(eset) <- new("AnnotatedDataFrame", data = pData)
```

To show the effect of quantile normalization, generate two box plot diagrams for the raw and RMA processed data. First, use the function par to define a new window in which two plots can be placed next to each other (see **Note 9**). Subsequently, generate two box plot diagrams with the function boxplot. The two box plots nicely illustrate the effect of RMA preprocessing. As a result of quantile normalization, the arrays have nearly identical median intensities and comparable intensity distributions after RMA preprocessing compared to the raw data (see Figure 2.2).

```
> par(mfrow = c(1,2))
> boxplot(rawData, las = 2, cex.axis = 0.5, main = "raw data")
> boxplot(eset, las = 2, cex.axis = 0.5, main = "RMA preprocessed data")
```

Next, convert the ExpressionSet object eset, to a matrix with RMA expression values (see **Note 10**) using the function exprs and assign the resulting matrix to the variable e. The matrix has nine columns (one for each CEL file) and 14,554 rows, each containing probe specific RMA expression values. The probes still include Affymetrix control probes beginning with

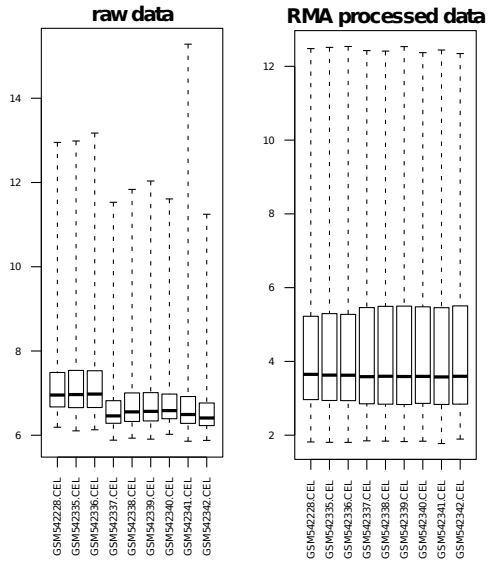

Figure 2.2 – Boxplots of raw and RMA processed data
For each array of the retentostat dataset (GSE21752), boxplots show the log intensity distributions of raw and RMA processed data. As a result of quantile normalization, the log intensity distributions are comparable after RMA processing (Irizarry *et al.*, 2003).

"AFFX". To remove them, use the function grep (see **Note 11**) and obtain an index referring to all probes that are starting with "AFFX". Subsequently create the matrix e.anig containing exclusively *A. niger* specific probes (see **Note 12**):

```
> e <- exprs(eset)
> subIndex <- grep("AFFX", featureNames(eset))
> e.anig <- e[-subIndex,]
```

Before starting the comparative analysis, do a simple test to exclude errors during the allocation of phenotypic labels to the CEL files. Using the function plotPCA, perform a principal component analysis (PCA) and plot the first two principal components against each other. Array data deriving from replicate time points is expected to cluster together. The PCA plot clearly shows that replicate array data from d0, d2 and d8 clusters together and that the three clusters are well separated (see Figure 2.3). Errors during the allocation of phenotypic labels to the CEL files can therefore be excluded.

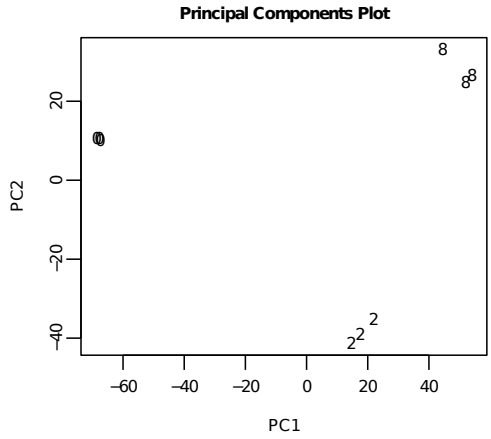

Figure 2.3 – Principle component analysis (PCA)
For each array of the retentostat dataset (GSE21752), the first two principle components are plotted against each other. Ideally, replicate arrays should cluster together and clusters of replicate arrays should be well separated. PCA plots can thus be applied to exclude errors during the allocation of phenotypic labels and to assess the reproducibility.

```
> par(mfrow = c(1,1))
> plotPCA(e.anig, groupnames = labels, pch = c
  ("0","0","0","2","2","2","8","8","8"), col = rep("black",9), legend =
  FALSE)
```

Multi-way comparisons

When analyzing microarray data, multiple-hypothesis testing has to be taken into account (see **Note 13**). Diverse methods have been suggested to correct p-values for multiple testing among which the Benjamini and Hochberg (BH) false discovery rate (FDR) (Benjamini *et al.*, 1995) is commonly used. However, multiple testing correction decreases the statistical power (see **Note 14**), which in many cases is anyway low due to small sample sizes. Different methods have been suggested to compensate for that. A reduction of the number of hypothesis tests to be performed by non-specific filtering of probes with low information content is one possibility. If the non-specific filtering is independent from the following hypothesis tests, it has been shown to increase the statistical power (Bourgon *et al.*, 2010). The standard deviations or mean values of probe intensities over all arrays (neglecting phenotypic labels) could be used for non-specific filtering. An alternative approach could be the computation of moderated (Baron *et al.*, 1986) instead of normal statistics. One such approach is implemented in the eBayes method from the Limma package (Smyth, 2004), where a global variance estimator

borrows information from all genes to infer probe-specific variances. A combination of non-specific filtering and moderated statistics is obviously an interesting approach. However, it has been shown to potentially decrease the statistical power and therefore either, non-specific filtering with normal t-statistics or moderated t-statistics with unfiltered data are recommended (Bourgon *et al.*, 2010).

It is very convenient to use the limma package for multi-way comparisons of microarray data. It can even be used for multi-factorial experimental designs (see **Note 15**). Compute moderated t-statistics on the unfiltered data by calling multiple limma functions. Indicate which experimental conditions should be compared to each other by defining a contrasts matrix with the function makeContrasts:

```
> f <- factor(pData$Target, levels = levels(pData$Target))
> design <- model.matrix(~0 + f)
> colnames(design) <- levels(pData[,2])
> fit <- lmFit(e.anig, design)
> contrasts <- makeContrasts(d2 - d0, d8 - d2, d8 - d0, levels = design)
> fit2 <- contrasts.fit(fit, contrasts)
> fit2 <- eBayes(fit2)
```

To better understand the concept of the contrasts matrix, have a look at how R represents it. Type at the R command line:

```
> contrasts
```

Next, make Venn diagrams to get an overview of the sets of genes that are differentially expressed comparing d2 to d0, d8 to d2 and d8 to d0. First, use the function decideTests to decide whether or not genes are differentially expressed controlling the FDR at 0.005:

```
> results <- decideTests(fit2, adjust.method = "fdr", p.value = 0.005)
```

The results matrix consists of three columns, one for each comparison defined in the contrasts matrix, and rows for each probe. The numbers indicate if a gene is not differentially expressed (0), upregulated (+1) or downregulated (-1). To get an impression of the results matrix, take a look at rows 120 to 130:

```
> results[120:130,]
```

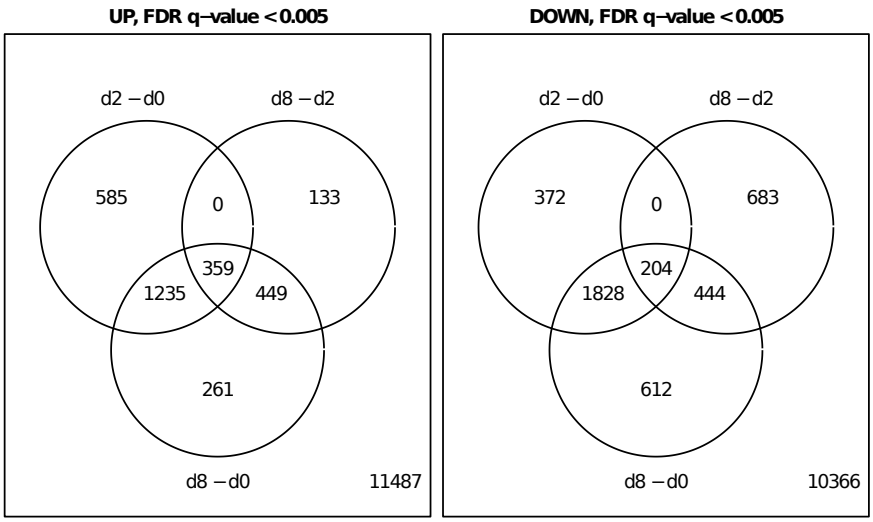

Figure 2.4 – Venn diagrams
Venn diagrams showing the relations between sets of up- and downregulated genes identified comparing day 0 (d0), day 2 (d2) and day 8 (d8) of retentostat cultivation.

Finally, use the information of the results matrix to generate two Venn diagrams (see Figure 2.4) for the up- and downregulated genes:

```
> par(mfrow = c(1,2))
> vennDiagram(results, include = "up", main = "UP, FDR q−value < 0.005",
  cex = 0.8)
> vennDiagram(results, include = "down", main = "DOWN, FDR q−value <
  0.005", cex = 0.8)
```

Next, obtain some of the results from the moderated t-statistics that were computed with the limma package. Extract the p-values and assign them to the matrix p.values (see **Note 16**).

```
> p.values <− fit2$p.value
```

In order to have distinct column names when combining different data later, change the column names of the p.value matrix. Use a for loop (see **Note 17**) to access column by column of the p.value matrix. In each cycle of the for loop, the function paste (see **Note 18**) is used to generate a new column name by preceding the corresponding column name from the contrasts matrix with “pValue”.

```
> for(i in 1:ncol(contrasts)){
>  colnames(p.values)[i] <- paste("pValue.", colnames(contrasts)[i], sep =
   " ")
> }
```

To obtain FDR q-values, use the function p.adjust and correct the p-values for multiple hypothesis testing. For each column of the p.value matrix, apply the Benjamini and Hochberg method to compute FDR q-values, combine them to a new matrix fdr.values using the function cbind (see **Note 19**) and change their column names:

```
> fdr.values <- NULL
> for(i in 1:ncol(p.values)){
>  fdr.values <- cbind(fdr.values, p.adjust(p.values[,i], method = "BH"))
>  colnames(fdr.values)[i] <- paste("qValue.", colnames(contrasts)[i], sep
   = " ")
> }
```

Next, obtain the fold changes applying the limma function topTable (see **Note 20**). Loop through all comparisons defined in the contrasts matrix and call the function topTable to extract the log2 fold changes for the corresponding comparison. Subsequently transform the log2 fold changes to normal scale and appended them to the matrix FCs. Define the column names of the matrix FCs analogously to the examples given above.

```
> FCs <- NULL
> for(i in 1:ncol(contrasts)){
>  toptable <- topTable(fit2, coef = i, number = 15000, adjust.method = "
  BH", sort.by = "none", p.value = 1, lfc = 0)
>  FCs <- cbind(FCs,2^toptable$logFC)
>  colnames(FCs)[i] <- paste("FC.", colnames(contrasts)[i], sep = " ")
> }
```

Besides the RMA expression data for the nine arrays, it is helpful to provide mean expression data for each of the three time points. Make use of the pData matrix, which allocates phenotypic labels to the CEL files to loop through the three time points. Because the dataset consists of triplicates, use the function unique (see **Note 21**) to access each time point only once. In each cycle of the loop, call the function which (see **Note 22**) to obtain an index pointing to the corresponding columns of the expression data matrix. Subsequently, use the function apply (see **Note 23**) to calculate mean expression values and convert the log2 expression data to normal scale. At the end of each loop, extend the matrix mean by the computed mean expression data and define the column names accordingly:

```
> means <- NULL
> for(i in 1:length(unique(pData$Target))){
>  Index <- which(eset$Target == unique(pData$Target)[i])
>  log2.a <- apply(e.anig[,Index],1,mean)
>  a <- 2^(log2.a)
>  means <- cbind(means, a)
>  colnames(means)[i] <- paste("mean.", unique(pData$Target)[i], sep = "
  ")
> }
```

Exporting data

For sharing data, it is wise to export it in such a way that it can be imported into any spreadsheet program. Use the function cbind (see **Note 19**) to combine the RMA and mean expression data as well as fold changes, p-vlaues and FDR q-values into the new matrix export:

```
> export <- cbind(e.anig, means, FCs, p.values, fdr.values)
```

Finally, use the function write.table to save the matrix export as a tab-delimited plain text file "results.txt" in the current WD:

```
> write.table(export, quote = FALSE, row.names = TRUE, col.names = NA, sep
  = "\t", file = "results.txt")
```

Construction of gene co-expression networks

In the following, we give a short demonstration on how gene co-expression networks can be built when starting with raw CEL file data. The transcriptomic dataset used was published in 2006 for *A. niger* (Martens-Uzunova *et al.*) and comprises 29 microarrays, which can be downloaded from the EMBL ArrayExpress database via its accession number: E-MEXP-1626. Briefly, the data derives from shake flask experiments in which fungal biomass from 18 hours pre-grown cultures was transferred to seven carbon sources: rhamnose, xylose, sorbitol, fructose, galacturonic acid, polygalacturonic acid and sugar beet pectin. In total, one reference sample was taken from the pre-culture and subsequently four samples were taken within 24 hours after transfer for each of the seven carbon sources. Thus, no biological replicates are available for the transcriptomic data. The carbon sources cover repressing (fructose) and non-repressing (sorbitol) carbon sources as well as complex carbon sources (polygalacturonic acid

and sugar beet pectin) and common monomeric constituents (rhamnose, xylose and galacturonic acid).

Importing CEL files

Download the expression dataset from the ArrayExpress database (http://www.ebi.ac.uk/arrayexpress/), decompress (see **Note 3**) and copy the CEL files into a new WD. On the R command line, load the required libraries and define the WD:

```
> library("affy")
> library("affyPLM")
> library("affycoretools")
> library("genefilter")
> setwd("absolute path to WD/")
```

Next, use the function ReadAffy to import all CEL files present in the WD. No phenotypic labels need to be associated with the CEL file names, because we are not interested in differential expression between specific conditions.

```
> rawData <- ReadAffy(verbose = TRUE)
```

Quality assessment and preprocessing

The affyPLM package has been suggested for quality assessment of transcriptomic data and detection of outlier arrays (Bolstad *et al.*, 2005). First, fit a linear model to the Affymetrix probe level data using the function fitPLM. Based on the linear model, generate two plots that are particularly helpful to detect outlier arrays. The normalized unscaled standard error (NUSE) and the relative log expression (RLE) plots.

```
> rawDataPLM <- fitPLM(rawData)
> par(mfrow = c(1,2))
> boxplot(rawDataPLM, main = "NUSE", ylim = c(0.95, 1.25), las = 2,
  whisklty = 0, staplelty = 0, cex.axis = 0.5)
> abline(h = 1)
> Mbox(rawDataPLM, main = "RLE", whisklty = 0, staplelty = 0, ylim = c
  (-0.4, 0.4), las = 2, cex.axis = 0.5)
> abline(h = 0)
```

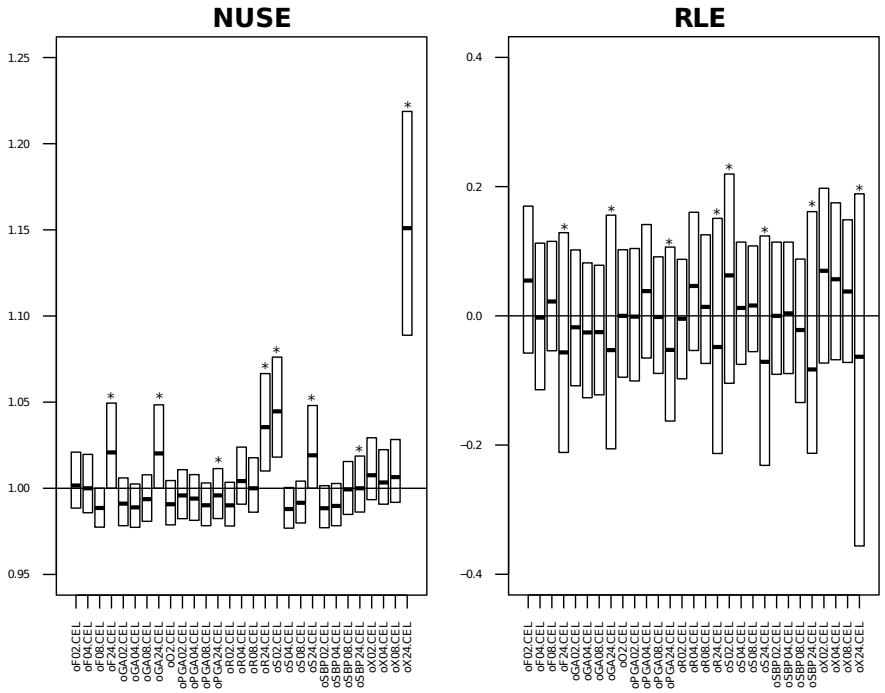

Figure 2.5 — Quality assessment using probe level models (PLM)
Normalized Unscaled Standard Error (NUSE) and Relative Log Expression (RLE) plots for each array of the carbon source transfer experiments (E-MEXP-1626). Outlier arrays (marked with asterisks) can be recognized by their deviation from the remaining arrays (Bolstad *et al.*, 2005).

Problematic arrays can be recognized from the NUSE plot as having elevated and more spread intensities. In the RLE plot, problematic arrays tend to have a larger spread and a median different from zero (Bolstad *et al.*, 2005). Inspecting the NUSE and RLE plots, all seven arrays from the 24 h samples can be considered as outlier arrays (see Figure 2.5). This could be due to secondary effects such as carbon and oxygen limitations during shake flask cultivation that are independent from the carbon sources. Furthermore, the first sorbitol time point appears to be problematic, probably because the fungus requires more time to induce the enzymatic machinery to metabolize it and therefore, during the first time points suffers from carbon limitation. We recommend removing these eight arrays before continuing with further analysis (see **Note 11** and **Note 12**).

```
> problematicArrays <- grep("24|oS04", sampleNames(rawData))
> rawData.sub <- rawData[,-problematicArrays]
```

Using a spike-in and dilution benchmark dataset, it was shown that the RMA algorithm is superior to the Affymetrix MAS5.0 algorithm for the detection of differentially expressed genes (Irizarry *et al.*, 2003). However, as recently reviewed (Usadel *et al.*, 2009), the preprocessing method of choice for gene correlation studies is less clear. With two different approaches (Harr *et al.*, 2006; Lim *et al.*, 2007), it was shown that RMA preprocessing is not suited for gene co-expression studies because of gene correlations introduced as artifacts by the RMA algorithm. For *Escherichia coli*, it was reported that a combination of the MAS5.0 background correction procedure with non-linear normalization (invariantset method) and probe summarization according to Li-Wong (Li *et al.*, 2001) gave the best correlations between co-transcribed operon genes (Harr *et al.*, 2006).

To combine MAS5.0 background correction with non-linear normalization and Li-Wong probe summarization, use the expresso function from the affy package. Assign the results, which are organized as an object of the class ExpressionSet to the variable eset. Subsequently, convert the ExpressionSet object to the expression matrix e and remove all Affymetrix control probes to finally obtain the matrix e.anig:

```
> eset <- expresso(rawData.sub, bgcorrect.method = "mas", pmcorrect.method
   = "mas", normalize.method = "invariantset", summary.method = "liwong")
> e <- exprs(eset)
> subIndex <- grep("AFFX", featureNames(eset))
> e.anig <- e[-subIndex,]
```

Non-specific filtering

For the construction of gene co-expression networks, the expression patterns of every pairwise combination of genes are checked for significant correlation. The number of probe pairs that can be drawn from n probes can be calculated with the binomial coefficient $\binom{n}{2}$ and is equal to $\frac{n!}{n(n-2)!}$. With the function choose, calculate the number of comparisons required for the e.anig matrix:

```
> choose(nrow(e.anig), 2)
```

The number of comparisons is 105,248,286, which will require a lot of computing time and memory. The number of comparisons can be reduced by not taking those genes into consideration that have a low overall variability across all arrays. Removal of all genes with standard deviation (SD) lower than the mode of the distribution of SDs has been suggested (Hahne *et al.*, 2008) as a good threshold for non-specific filtering based on variability (see **Note 24**). First,

calculate row-wise SDs with the rowSds function, determine the mode of the distribution of SDs with the function shorth and select all probes with SDs larger than the mode:

```
> sds <- rowSds(e.anig)
> sds.mode <- shorth(sds)
> e.anig.sub <- e.anig[sds > sds.mode,]
```

After this non-specific filtering step, the pairwise comparisons are reduced by approximately 50%. 10,647 probes are left for which a total of 56,673,981 pairwise comparisons have to be computed.

Building gene co-expression networks

To compute pairwise correlation coefficients, use the function cor. For each pairwise combination of columns from a given matrix, cor computes correlation coefficients as specified by the method parameter. Before the function can be applied, the matrix e.anig.sub has to be transposed using the function t. In the resulting matrix e.anig.sub.t, columns (CEL files) and rows (probes) are interchanged.

```
> e.anig.sub.t <- t(e.anig.sub)
```

The application of different correlation coefficients for the construction of gene co-expression networks has been discussed in a recent review (Usadel *et al.*, 2009). The Pearson correlation coefficient is often used as a measure for gene co-expression. However, it is sensitive to outliers and high correlation coefficients can only be obtained for linear relationships. The Spearman rank correlation coefficient has been supposed as a good alternative. The Spearman correlation coefficient is less sensitive to outliers because it is not directly calculated from the expression values but from ranks of expression values. Whereby, it can also detect correlations for non-linear relationships such as the Michaelis-Menten kinetic. To calculate pairwise Spearman correlation coefficients for the transposed matrix e.anig.sub.t, use the function cor (see **Note 25**):

```
> e.anig.sub.t.cor <- cor(e.anig.sub.t, method = "spearman")
```

The correlation matrix is symmetric with respect to its main diagonal. Row-column combinations define pairs of genes. Probes correlated to themselves are lying on the main diagonal and both halves (above and below the main diagonal) of the matrix contain equal information.

Therefore, only correlation coefficients from one half of the correlation matrix excluding its main diagonal have to be checked and compared to a threshold.

Before checking all pairwise comparisons, set a range of critical correlation coefficients by defining the minimal and maximal correlation coefficients cutoff.min and cutoff.max, as well as the increment cutoff.increment. The code below exemplarily defines a range of critical correlation coefficients, for which five different co-expression networks (with absolute correlation coefficients smaller than 0.70, 0.75, 0.80, 0.85, 0.90 and 0.95) will be constructed within a for loop (see **Note 17**).

```
> cutoff.min <- 0.7
> cutoff.max <- 0.95
> cutoff.increment <- 0.05
```

In order to check one half of the correlation matrix excluding its main diagonal, basically two for-loops have to be initiated. The outer for-loop increments its counter i, used as row index, by 1 starting with the first row until it reaches the last but one row. The inner for-loop increments its counter j, used as column index, by 1 starting at i plus 1 until it reaches the last column. In the inner for-loop, the row and column indices i and j are used to access the corresponding correlation coefficient and compare it to the range of thresholds defined above. For this, a third for-loop is initiated where probe pairs are linked to each other by positive or negative correlations if the correlation coefficient is larger or smaller in case of positive or negative values, respectively. The results are exported to files in the simple interaction format (SIF) format and saved in the WD (see **Note 24**).

```
> for(i in seq(1, nrow(e.anig.sub.t.cor) - 1, 1)){
>  print(i)
>  for(j in seq(i + 1, ncol(e.anig.sub.t.cor), 1)){
>   for(cutoff in seq(cutoff.min, cutoff.max, cutoff.increment)){
>    file.name <- paste("network.", cutoff, ".sif")
>    if(e.anig.sub.t.cor[i,j] > 0 && e.anig.sub.t.cor[i,j] > cutoff){
>     write(paste(row.names(e.anig.sub)[i], "POScorrelation", row.names(e.
  anig.sub)[j], sep = " "), file = file.name, append = TRUE)
>    }
>    if(e.anig.sub.t.cor[i,j] < 0 && e.anig.sub.t.cor[i,j] < (-1) * cutoff
  ){
>     write(paste(row.names(e.anig.sub)[i], "NEGcorrelation", row.names(e.
  anig.sub)[j], sep = " "), file = file.name, append = TRUE)
>    }
>   }
>  }
> }
```

The resulting SIF files can be imported into Cytoscape (Shannon *et al.*, 2003), an open source platform for network analysis and visualization. Cytoscape provides a graphical user interface and various plug-ins for network analysis exist, such as MCODE (Bader *et al.*, 2003) for the identification of clusters of highly connected genes or BinGO (Maere *et al.*, 2005) for enrichment analysis of Gene Ontology terms.

Notes

1. If R cannot be called from every directory at the command line, the directory containing its executable has to be added to the PATH environment variable. Details are depending on the operating system.

2. For R code and command line commands, each new line begins with a "larger-than" symbol (>). For copy and pasting the code to the corresponding command prompt, the "larger-than" symbols have to be removed.

3. For Windows systems, the open source program 7-zip (http://www.7-zip.org/) can be used for decompression.

4. In general, most R and Bioconductor functions/packages are well documented. Typing a question mark directly in front of any function at the R command line should open a short documentation (for example `?mean` opens the documentation for the function mean). Furthermore, many packages provide additional documentations (so-called vignettes) containing executable examples. When putting two question marks directly in front of any word, a list can be obtained with functions matching that word (for example `??mean` lists all functions matching the word mean).

5. The function `setwd` can be used to define the WD. The forward slash (/) is used as path separator. It is most robust, to define the absolute path. While on Mac OS and Linux, `setwd("/Users/someUser/Documents/someWD/")` could for example be used, it could look something like `setwd("C:/Documents and Settings/someWD/")` on a Windows system.

6. In R, values can be assigned to variables using <- or ->, which are assigning a value on their right to a variable on their left, or vice versa.

7. There is no critical slope indicating RNA degradation since different chip architectures result in different chip specific slopes. Rather than the actual values of the slopes, it is important that the slopes are comparable for different samples (Bolstad *et al.*, 2005; MacDonald, 2008).

8. Plots can be exported to PDF files by simply preceding the plot command(s) with pdf("desired_filename.pdf"). With dev.off(), the PDF file is finalized and saved in the current WD. If multiple plots are generated after initializing the PDF document, they will be appended to each other resulting in a single PDF file.

9. The function par can be used to set diverse graphical parameters among which mfrow can be used to define the number of rows and columns. Defining par(mfrow = c(i, j)) before plotting graphs will define i rows and j columns allowing to place i times j plots on one page or window.

10. Unlike most other preprocessing procedures, RMA expression values are in log(2) scale.

11. The function grep can be applied on a vector and returns an index of elements that match a given pattern.

12. To obtain parts of vectors or matrixes, subscripts can be used in R. Subscripts are given in squared brackets directly behind the corresponding variable. With v[3], the 3rd element from the one dimensional vector v can be obtained. For matrixes, the squared brackets contain two subscripts separated by a comma. The first subscript refers to the row and the second to the column. For the matrix m, the value of the 5th row and the 2nd column can be obtained by m[5,2]. Furthermore, the values of the 5th row and columns 2 to 5 can be obtained by m[5,2:5]. With m[5,c(2,4,5)], the values of the 5th row and columns 2, 4 and 5 can be obtained. By using negative subscripts, elements can be dropped from a vector or matrix. In addition, subscripts can consist of logical expressions (for example > or <) and subscripts can also be used for sorting. Many more details can be found in basic R documentations.

13. If one performs 10,000 independent t-tests on a set of 10,000 genes to identify differentially expressed genes applying a critical p-value of 0.05, the number of false positives (Type I error α) accounts to 500 (5% of 10,000). Without an adjustment for multiple testing, the Type I error rate is not controlled at the level indicated by the p-value, especially because the majority of genes is not expected to be differentially expressed in microarray studies.

14. The Type II error β accounts for false negatives and the statistical power is defined as $1 - \beta$.

15. The code can easily be adapted for similar analysis. The only lines that have to be changed are line 4 and line 5. Please consult the documentation of the limma package to understand the different commands in more detail.

16. The $ operator can be used in R to obtain a component from an object.

17. For loops can by applied to generalize or automate repeating blocks of code. For loops consist of two parts. The first part, which is enclosed by brackets initiates the loop by setting the loop counter, comparing it to a defined limit and incrementing it in each cycle. The second part of the loop, the body, is enclosed by curly brackets and contains the code to be executed in each cycle until the loop counter reaches the defined limit. By initializing a loop with (i in 1:ncol(contrasts)), the loop counter i is defined to start at 1, being incremented in each cycle by 1 until it is equal to the number of columns of the contrasts matrix. Alternatively, the function seq can be used to define the range of the loop counter i as follows (seq(start value, end value, increment)).

18. The function paste can be used to concatenate strings. The argument sep defines how the strings are separated. Calling paste("one", "two", sep = "-") results in "one-two".

19. The function cbind is used here to append matrixes or columns from matrixes together. The number and the order of rows have to be identical, because cbind does not take row names into consideration. If that his not the case, the function merge can be used.

20. The limma function topTable can be used to list the top-ranked genes from the limma analysis based on fold change, FDR or other criteria. Besides other information, the function topTable returns the logarithmic fold changes for the corresponding comparisons. In this example, the argument sort.by = "none" represses any sorting, because the data will be combined with other data based on the original row order using the function cbind later. The argument coef specifies which comparison of the contrasts matrix is of interest.

21. The function unique can be used to remove redundant elements from a vector.

22. To check within a logical expression if two arguments are equal, a double equal sign (==) has to be used. If only a single equal sign (=) is used, the argument on the right site gets assigned to the variable on the left.

23. The function apply can be used to apply a function to rows or columns of a matrix. While apply(x, 1, mean) computes column-wise mean values for all rows, apply(x, 2, mean) computes row-wise mean values for all columns of the matrix x.

24. Depending on the available computational resources, one might consider not to perform any non-specific filtering or to apply a more severe threshold. We recommend applying a range of filter settings and comparing the resulting co-expression networks.

25. With the method parameter for the cor function, different correlation coefficients can easily be calculated. Should R give an error message related to problems with memory allocation, apply a more stringent threshold for non-specific filtering to reduce the number of pairwise combinations.

Acknowledgments

This work was supported by a grant of SenterNovem IOP Genomics project IGE07008. Part of this work was carried out within the research programme of the Kluyver Centre for Genomics of Industrial Fermentation, which is part of the Netherlands Genomics Initiative/Netherlands Organization for Scientific Research. We thank T.G. Homan for discussions and proof reading of the manuscript.