Basic infrastructure for using oligonucleotide microarray reporter sequence information for preprocessing and quality assessment

Wolfgang Huber and Robert Gentleman

December 19, 2008

Contents

1 Overview

This package provides some basic and simple tools for dealing with the oligonucleotide microarray reporter sequence information in the Bioconductor *probe* packages. This information is used, for example, in the *gcrma* package.

For general work with genomic sequences, this package is inadequate. Please refer to the *Biostrings* package for that.

As an example, the package *hgu95av2* provides microarray reporter sequences for Affymetrix' *HgU95a version 2* genechip, and for almost all major Affymetrix genechips, the corresponding packages can be downloaded from the Bioconductor website. If you have the reporter sequence information of a particular chip, you can also create such a package yourself. This is described in section 5.

First, let us load the *matchprobes* package and some other packages we will use.

```
> library("matchprobes")
> library("affy")
> library("hgu95av2cdf")
> library("hgu95av2probe")
```

2 Using probe packages

Help for the probe sequence data packages can be accessed through

```
> ? hgu95av2probe
```

matchprobes is an infrastructure package. Using it requires basic familiarity with R and with the design of the *AffyBatch* class in the *affy* package, Bioconductor's basic container for Affymetrix genechip data. One of the issues that you have to deal with is that the *probe* packages do not provide the reporter sequences of all the features present in an *AffyBatch*. Some sequences are missing, some are implied; in particular, the data structure in the *probe* packages does not contain explicitely contain the sequences of the mismatch probes, since they are implied by the perfect match probes. Also, some other features, typically harbouring control probes or empty, do not have sequences. This is the choice that Affymetrix made when they made files with probe sequences available, and we followed it.

Practically, this means that the vector of probe sequences in a *probe* package does not align 1:1 with the rows of the corresponding *AffyBatch*; you need to keep track of this mapping, and some tools for this are provided and explained below (see Section 2.2). It also means that some functions from the *affy* package, such as `pm`, cannot be used when the sequences of the probes corresponding to their result are needed; since `pm` reports the intensities, but not the identity of the probes it has selected, yet the latter would be needed to retrieve their sequences.

2.1 Basic functions

Let us look at some basic operations on the sequences.

2.1.1 Complementary sequence

```
> example(complementSeq)

cmplmS>  seq <- c("AAACT", "GGGTT")

cmplmS>  complementSeq(seq)
[1] "TTTGA" "CCCAA"

cmplmS>  seq <- c("CGACTGAGACCAAGACCTACAACAG", "CCCGCATCATCTTTCCTGTGCTCTT")

cmplmS>  complementSeq(seq, start=13, stop=13)
[1] "CGACTGAGACCATGACCTACAACAG" "CCCGCATCATCTATCCTGTGCTCTT"
```

2.1.2 Reverse sequence

```
> example(reverseSeq)

rvrsSq>  w <-  c("hey there", "you silly fool")

rvrsSq>  reverseSeq(w)
[1] "ereht yeh"      "loof yllis uoy"

rvrsSq>  w <- "able was I ere I saw Elba"

rvrsSq>  reverseSeq(w)
[1] "ablE was I ere I saw elba"
```

```
rvrsSq> rna1 <- "UGCA"

rvrsSq> revcompRNA(rna1)
[1] "UGCA"

rvrsSq> dna1 <- "TGCA"

rvrsSq> revcompDNA(dna1)
[1] "TGCA"
```

2.1.3 Matching sets of probes against each other

```
> pm    <- hgu95av2probe$sequence
> pmdup <- c(pm, pm)
> query <- hgu95av2probe$sequence[21:23]
> m = matchprobes(query, pmdup)
> unlist(m)

match1 match2 match3 match4 match5 match6
    21 201821     22 201822     23 201823
```

2.1.4 Base content

The base content (number of occurence of each character) of the sequences can be computed with the function `basecontent`.

```
> bcpm <- basecontent(hgu95av2probe$sequence)
> head(bcpm)

 A  T  C G
 1  8 10 6
 5 10  5 5
 6 12  4 3
 4  6  7 8
 4  8  5 8
 2 10  7 6
```

2.2 Relating to the features of an *AffyBatch*

```
> nr = hgu95av2dim$NROW

[1] 640

> nc = hgu95av2dim$NCOL

[1] 640
```

Each column of an *AffyBatch* corresponds to an array, each row to a certain probe on the arrays. The probes are stored in a way that is related to their geometrical position on the array. For example, the *hgu95av2* array is geometrically arranged into 640 rows and 640 columns; we also call them the x- and y-coordinates. This results in $640 \times 640 = 409600$ rows of the *AffyBatch*; we also call them indices. To

convert between x- and y-coordinates and indices, you can use the functions `xy2indices` and `indices2xy` from the *affy* package.

The sequence data in the *probe* packages is addressed by their x and y–coordinates. Let us construct a vector `abseq` that aligns with the rows of an *hgu95av2 AffyBatch* and fill in the sequences:

```
> abseq = rep(as.character(NA), nr*nc)
> ipm = with(hgu95av2probe, xy2indices(x, y, nr=nr))
> abseq[ipm] = pm
> table(is.na(abseq))

 FALSE   TRUE
201800 207800
```

The mismatch sequences are not explicitely stored in the probe packages. They are implied by the match sequences, by flipping the middle base. This can be done with the function `complementSeq`. For Affymetrix GeneChips the length of the probe sequences is 25, and since we start counting at 1, the middle position is 13.

We compute `imm`, the indices of the mismatch probes, by noting that each mismatch has the same x-coordinate as its associated perfect match, while its y-coordinate is increased by 1.

```
> mm <- complementSeq(pm, start=13, stop=13)
> cat(pm[1], mm[1], sep="\n")

TGGCTCCTGCTGAGGTCCCCTTTCC
TGGCTCCTGCTGTGGTCCCCTTTCC

> imm = with(hgu95av2probe, xy2indices(x, y+1, nr=nr))
> abseq[imm] = mm
> table(is.na(abseq))

 FALSE   TRUE
403600   6000
```

See Figures 3–5 for some applications of the probe sequence information to preprocessing and data quality related plots.

3 CombineAffyBatch

In this section, we show how to combine the data from two different array types whose reporter sequences are partially the same.

```
> library("hu6800cdf")
> library("hu6800probe")
```

Load the data.

```
> f1 <- system.file("extdata", "118T1.cel", package="matchprobes")
> f2 <- system.file("extdata", "CL2001032020AA.cel", package="matchprobes")
> pd1 <- new("AnnotatedDataFrame", data=data.frame(fromFile=I(f1), row.names="f1"))
> pd2 <- new("AnnotatedDataFrame", data=data.frame(fromFile=I(f2), row.names="f2"))
> x1 <- read.affybatch(filenames=f1, compress=TRUE, phenoData=pd1)
> x2 <- read.affybatch(filenames=f2, compress=TRUE, phenoData=pd2)
```

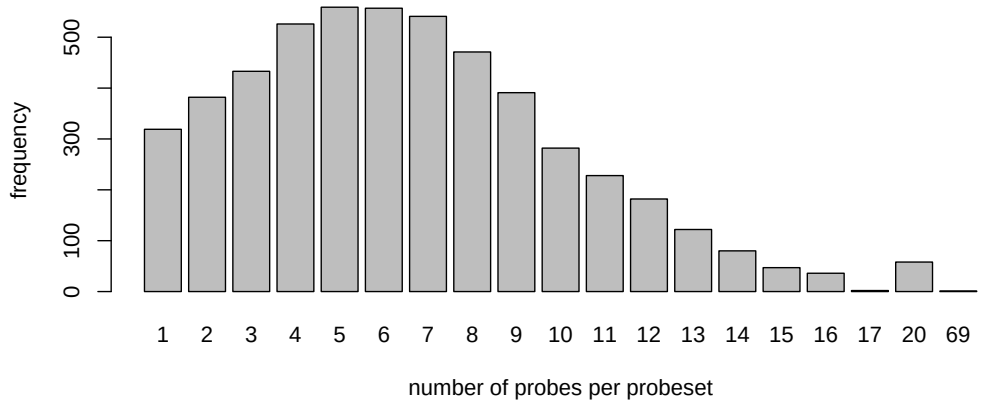

Figure 1: The distribution of the number of common probes per probeset.

`f1` and `f2` contain the filenames of the CEL files. In practice, they will be vectors with many filenames per array type, here, for demonstration, we only use one single CEL file per array type, which are provided in the subdirectory `extdata` of the package.

Combine the data. For this to work it is required that the packages *hu6800probe* and *hgu95av2probe* are installed.

```
> res <- combineAffyBatch(list(x1, x2), c("hu6800probe","hgu95av2probe"), newcdf="comb")
```

```
package:hu6800probe        hu6800probe
package:hgu95av2probe        hgu95av2probe
34431 unique probes in common
```

The function returns a list `res` with two elements: an `AffyBatch data` and a `CDF enviroment newcdf`.

```
> comb <- res$cdf
> z    <- rma(res$dat)
```

First, let us look at the distribution of the number of common probes per probeset (see Figure 1).

```
> prs <- mget(ls(comb), comb, ifnotfound=NA)
> nrprobes <- sapply(prs, function(x) nrow(x))
> barplot(table(nrprobes),
+    xlab="number of probes per probeset",
+    ylab="frequency")
```

Let us also make scatterplots of the two combined arrays, first on the probe and then at the probe set level (see Figure 2).

```
> png("matchprobes-scp.png", width=900, height=480)
> par(mfrow=c(1,2))
> plot(exprs(res$dat), main="after combine",
+   pch=".", asp=1, xlab="f1", ylab="f2", log="xy")
> plot(exprs(z), main="after RMA",
+   pch=".", asp=1, xlab="f1", ylab="f2")
> dev.off()
```

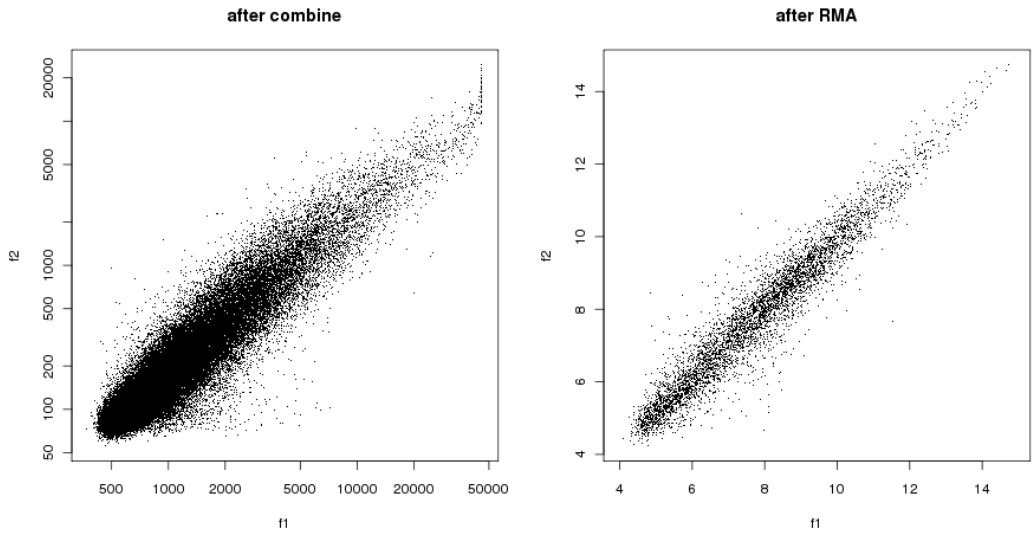

Figure 2: Comparison of the intensity data from common probes (left panel) and probesets (right panel) of the two array platforms.

Here, we explicitely used the `png` device to write this plot to; this circuments the default choice of `Sweave`, PDF, which can become cumbersomely large in size for feature-rich plots such as this one.

4 Some sequence related "preprocessing and quality" plots

The function `basecontent` counts the number of occurences of each of the four bases A, C, G, T in each probe sequence.

```
> bc = basecontent(abseq)
> head(na.omit(bc))
```

```
     A  T C G
[1,] 6  2 9 8
[2,] 5  3 9 8
[3,] 6  3 9 7
[4,] 6  5 8 6
[5,] 4  6 8 7
[6,] 5 11 5 4
```

Let us define an ordered factor variable for GC content:

```
> GC = ordered(bc[,"G"] + bc[,"C"])
> colores = rainbow(nlevels(GC))
```

Figure 3 shows a barplot of the frequencies of counts in `GC`:

```
> barplot(table(GC), col=colores, xlab="GC", ylab="number")
```

Figure 4:

```
> boxplot(log2(exprs(x2)[,1]) ~ GC, outline=FALSE,
+   col=colores, , xlab="GC", ylab=expression(log[2]~intensity))
```

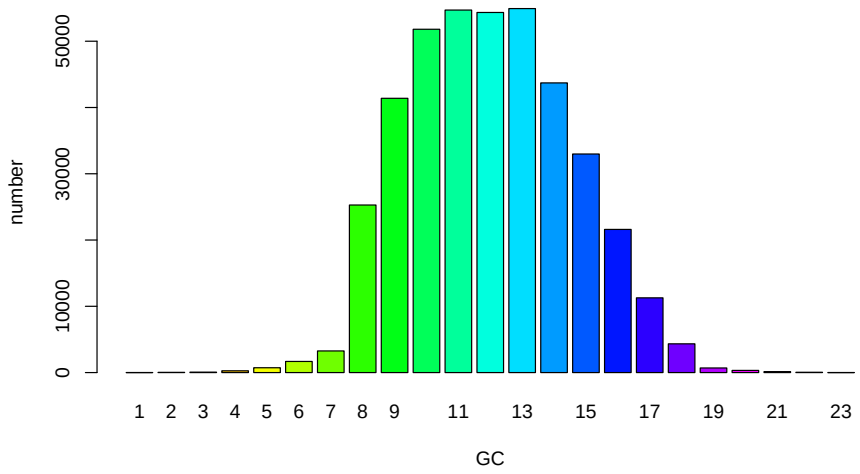

Figure 3: Distribution of probe GC content. The height of each bar corresponds to the number of probes with the corresponing GC content.

Figure 5:

```
> png("matchprobes-p2p.png", width=900, height=480)
> plot(exprs(x2)[ipm,1], exprs(x2)[imm,1], asp=1, pch=".", log="xy",
+      xlab="PM", ylab="MM", col=colores[GC[ipm]])
> abline(a=0, b=1, col="#404040", lty=3)
> dev.off()
```

5 Creating probe packages

Probe packages are a convenient way for distributing and storing the probe sequences and related information.

5.1 For Affymetrix genechips

In this section we see how a probe package can be created for Affymetrix genechips from the tabulator-separated sequence files that can be obtained from the vendor (at http://www.affymetrix.com/support). As an example, the file `HG-U95Av2_probe_tab.gz` is provided in the `extdata` subdirectory of the *matchprobes* package.

```
> filename <- system.file("extdata", "HG-U95Av2_probe_tab.gz",
+                         package="matchprobes")
> outdir   <- tempdir()
> me       <- "Wolfgang Huber <w.huber@dkfz.de>"
> species  <- "Homo_sapiens"
> makeProbePackage("HG-U95Av2",
+                  datafile   = gzfile(filename, open="r"),
+                  outdir     = outdir,
+                  maintainer = me,
```

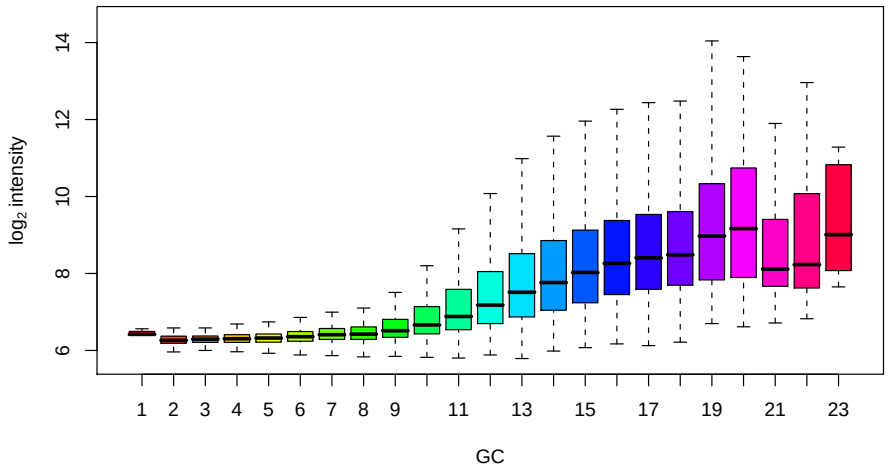

Figure 4: Boxplots of $\log_2$ intensity stratified by probe GC content.

```
+                   species    = species,
+                   version    = "0.0.1",
+                   force      = TRUE)

Importing the data.
Creating package in /tmp/RtmpLZ5gaq/hgu95av2probe
Writing the data.
Checking the package.
Building the package.
[1] "hgu95av2probe"

> dir(outdir)

[1] "hgu95av2probe"                "hgu95av2probe_0.0.1.tar.gz"
```

5.2 For other chiptypes

To deal with different file formats and additional types of probe annotation data from public or in-house databases, the function `makeProbePackage` offers a great deal of flexibility. The user can specify her own import function through the argument `importfun`. By default, its value is `getProbeDataAffy`, a function that reads tabular Affymetrix genechip sequence files. Import functions for other types of arrays can be adapted from this prototype.

The help pages and R code contained in the produced packages are derived from a template directory that obeys the usual R package conventions [1]. A prototype for such a directory is provided within the package *matchprobes*. To facilitate the automated production of large numbers of similar packages, we provide a text substitution mechanism similar to the one used in the GNU `configure` system.

The input parameters of an import function are

- a character string naming the array type
- the input data files

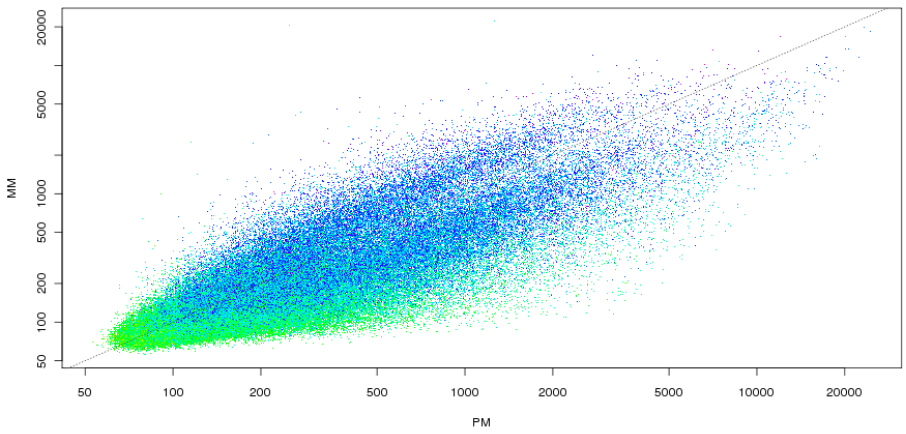

Figure 5: Scatterplot of PM vs MM intensities, colored by probe GC content.

- a character string with the directory name of a package template directory, containing at least a file `DESCRIPTION`, a directory `man` with a file @PKGNAME@.Rd, a directory `data`, and possible other directories and files, conforming to the usual R package conventions.
- ... any sort of further parameters, as necessary.

The output of an import function is a named list with elements

- `pkgname`: a character string with the package name.
- `dataEnv`: an environment containing an arbitrary number of data objects; these make the core of the package. Among these, there should be a data frame whose name is the value of `pkgname` with a column `sequence`. This data frame may have other columns such as the x- and y-position of the probes on the array, identifiers linking a probe to genomic databases, or its length and relative position within the gene it represents. The objects in the environment will be saved as individual `.rda` files of the same name into the data directory of the produced package. Documentation needs to be provided for the columns of the data frame, as well as for the other objects.
- `symVals`: a named list, containing the symbol-value substitutions that are used in the text processing. It must at least contain the elements
  - `ARRAYTYPE`: name of the array
  - `DATASOURCE`: a textual description of how the data were obtained. It should contain the URL, or the name of the company / person.

For more details, please refer to the help files for the functions `makeProbePackage` and `getProbeDataAffy`. For an example, refer to the source code of `getProbeDataAffy`.

References

[1] R Foundation (1999). *Writing R Extensions*. http://www.r-project.org.

8