Using lumi, a package processing Illumina Microarray

Pan Du[‡][*], Warren A. Kibbe[‡][†], Simon Lin[‡][‡]

October 3, 2007

[‡]Robert H. Lurie Comprehensive Cancer Center
Northwestern University, Chicago, IL, 60611, USA

Contents

1 Overview of lumi **2**

2 Installation of lumi package **2**

3 Object models of major classes **2**

4 Data preprocessing **3**
4.1 Intelligently read the BeadStudio output file 3
4.2 Quality control of the raw data 6
4.3 Background correction 14
4.4 Variance stabilizing transform 15
4.5 Data normalization 15
4.6 Quality control after normalization 18
4.7 Encapsulate the processing steps 19
4.8 Inverse VST transform to the raw scale 25

5 Handling large data sets **27**

6 Performance comparison **28**

7 Gene annotation **28**
7.1 Examples of nuID 29
7.2 Illumina microarray annotation packages 30
7.3 Transfer Illumina identifier annotated data into nuID annotated 31

8 A use case: from raw data to functional analysis **31**
8.1 Preprocess the Illumina data 32
8.2 Identify differentially expressed genes 32
8.3 Gene Ontology analysis 34

[*]dupan@northwestern.edu
[†]wakibbe@northwestern.edu
[‡]s-lin2@northwestern.edu

9 Session Info **34**

10 Reference **35**

1 Overview of lumi

Illumina microarray is becoming a popular microarray platform. The BeadArray technology from Illumina makes its preprocessing and quality control different from other microarray technologies. Unfortunately, until now, most analyses have not taken advantage of the unique properties of the BeadArray system. The *lumi* Bioconductor package especially designed to process the Illumina microarray data. The *lumi* package provides an integrated solution for the bead-level Illumina microarray data analysis. The package covers data input, quality control, variance stabilization, normalization and gene annotation.

The *lumi* package includes a new variance-stabilizing transformation (VST) algorithm that takes advantage of the technical replicates available on every Illumina microarray. A new robust spline normalization (RSN) algorithm, which combines the features of the quantile and loess normalization, is also implemented in this package. Options available in other popular normalization methods are also provided. Multiple quality control plots are provided in the package. To better annotate the Illumina data, a new, vendor independent nucleotide universal identifier (nuID) was devised to identify the probes of Illumina microarray. The nuID indexed Illumina annotation packages is compatible with other Bioconductor annotation packages. Mappings from Illumina Target Id or Probe Id to nuID are also included in the annotation packages. The output of lumi processed results can be easily integrated with other microarray data analysis, like differentially expressed gene identification, gene ontology analysis or clustering analysis.

2 Installation of lumi package

In order to install the lumi package, the user needs to first install R, some Bioconductor packages (*Biobase*, *affy*, *annotate*) and R packages (*mgcv*, *methods*). If the user is also interested in using vsn method, then the Bioconductor package *vsn* needs to be installed.

For the users want to install the latest developing version of lumi, which can be downloaded from the developing section of Bioconductor website. Some additional packages may be required to be installed because of the update the Bioconductor. These packages can also be found from the developing section of Bioconductor website.

An Illumina benchmark data package *lumiBarnes* can be downloaded from Bioconductor Experiment data website.

3 Object models of major classes

The *lumi* package has one major class: **LumiBatch**. **LumiBatch** is inherited from **ExpressionSet** class in Bioconductor for better compatibility. Their relations are shown in Figure 1. **LumiBatch** class includes *se.exprs*, *beadNum* and *detection* in **assayData** slot for additional informations unique to Illumina microarrays. A controlData slot is used to keep the control probe information, and a QC slot is added for keeping the quality control information. The S4 function `plot` supports different kinds of plots by specifying the specific plot type of **LumiBatch** object. See help of `plot-methods` function for details. The *history* slot records all the operations made on the **LumiBatch** object. This provides data provenance. Function `getHistory` is to retrieve the *history* slot. Please see the help files of **LumiBatch** class for more details. A series of functions: `lumiR`, `lumiB`, `lumiT`, `lumiN` and `lumiQ` were designed for data input, preprocessing and quality control. Function `lumiExpresso` encapsulates the preprocessing methods for easier usability.

4 Data preprocessing

The first thing is to load the *lumi* package.

```
> library(lumi)
```

```
This is mgcv 1.3-27
```

4.1 Intelligently read the BeadStudio output file

The `lumiR` function supports directly reading the Illumina raw data output of the Illumina Bead Studio toolkit from version 1 to version 3. It can automatically detect the BeadStudio output version and format and create a new **LumiBatch** object for it. An example of the input data format is shown in in Figure 2. For simplicity, only part of the data of first sample is shown. The data in the highlighted columns are kept in the corresponding slots of **LumiBatch** object, as shown in Figure 2. The `lumiR` function will automatically determine the starting line of the data. The columns with header including `AVG_Signal` and `BEAD_STD` are required for the **LumiBatch** object. By default, the sample IDs and sample labels are extracted from the column names of the data file. For example, based on the column name: `AVG_Signal-1304401001_A`, we will extract `"1304401001"` as the sample ID and `"A"` as the sample label (The function assumes the separation of the sample ID and the sample label is `"_"` if it exists in the column name.). The function will check the uniqueness of sample IDs. If the sample ID is not unique, the entire portion after removing `"AVG_Signal"` will be used as a sample ID. The user can suppress this parsing by setting the parameter ”parseColumnName” as FALSE.

The `lumiR` will automatically initialize the QC slot of the **LumiBatch** object by calling `lumiQ`. If BeadStudio outputted the control probe data, their information will be kept in the controlData slot of the **LumiBatch** object. If BeadStudio outputted the sample summary information, which is called [Samples Table] in the output text file, the information will be kept in BeadStudio-Summay within the QC slot of the **LumiBatch** object.

The BeadStudio can output the gene profile or the probe profile. As the probe profile provides unique mapping from the probe Id to the expression profile, outputting probe profile is preferred. When the probe profile is outputted, as show in Figure 2(B), the ProbeId column will be used as the identifier of **LumiBatch** object.

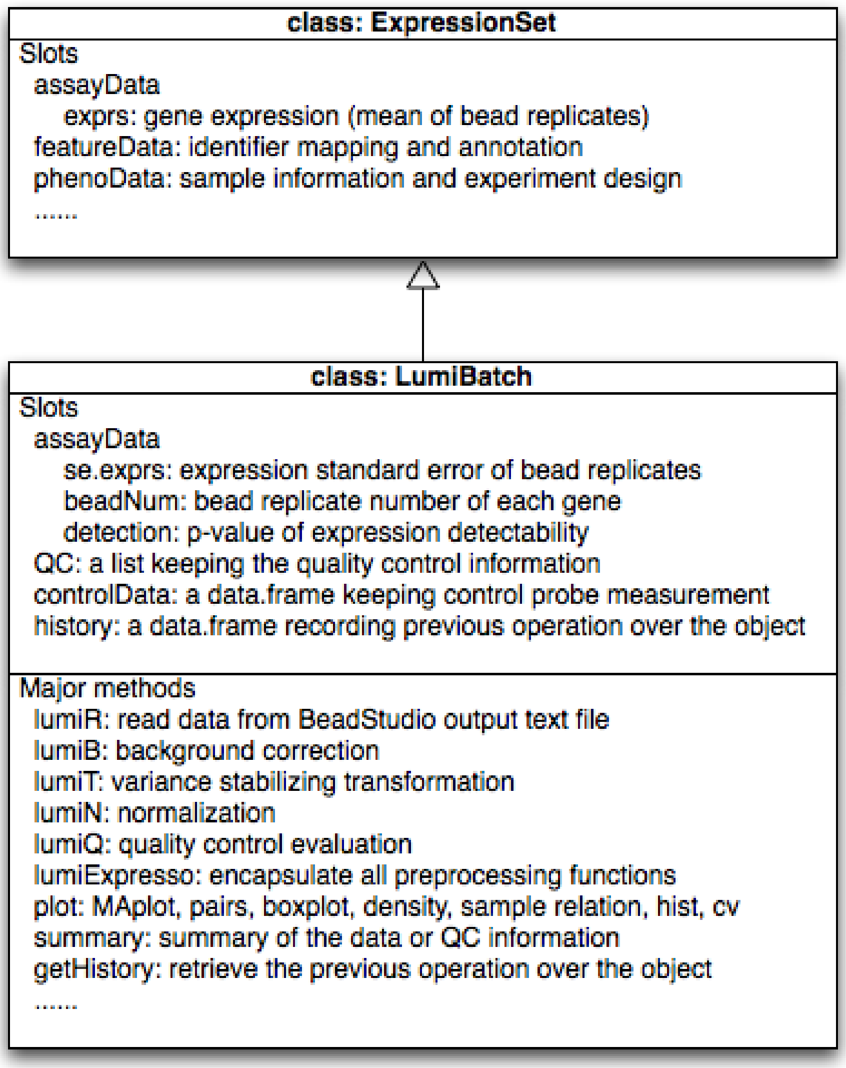
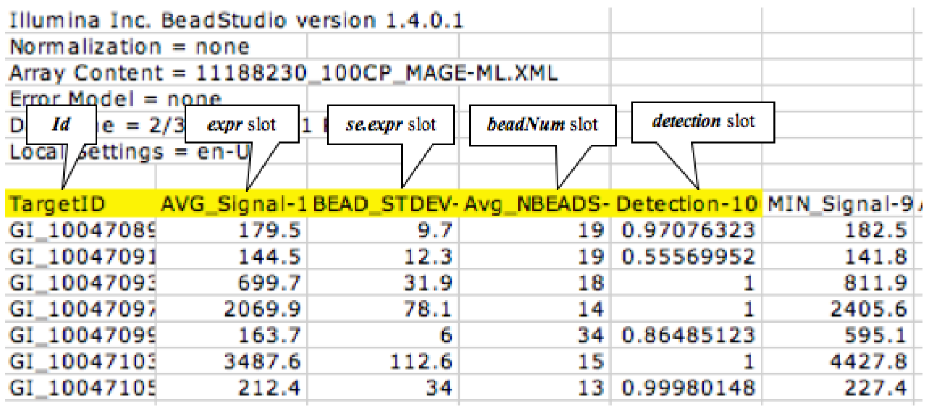

class: ExpressionSet
Slots
assayData
exprs: gene expression (mean of bead replicates)
featureData: identifier mapping and annotation
phenoData: sample information and experiment design
......
class: LumiBatch
Slots
assayData
se.exprs: expression standard error of bead replicates
beadNum: bead replicate number of each gene
detection: p-value of expression detectability
QC: a list keeping the quality control information
controlData: a data.frame keeping control probe measurement
history: a data.frame recording previous operation over the object
Major methods
lumiR: read data from BeadStudio output text file
lumiB: background correction
lumiT: variance stabilizing transformation
lumiN: normalization
lumiQ: quality control evaluation
lumiExpresso: encapsulate all preprocessing functions
plot: MAplot, pairs, boxplot, density, sample relation, hist, cv
summary: summary of the data or QC information
getHistory: retrieve the previous operation over the object
......

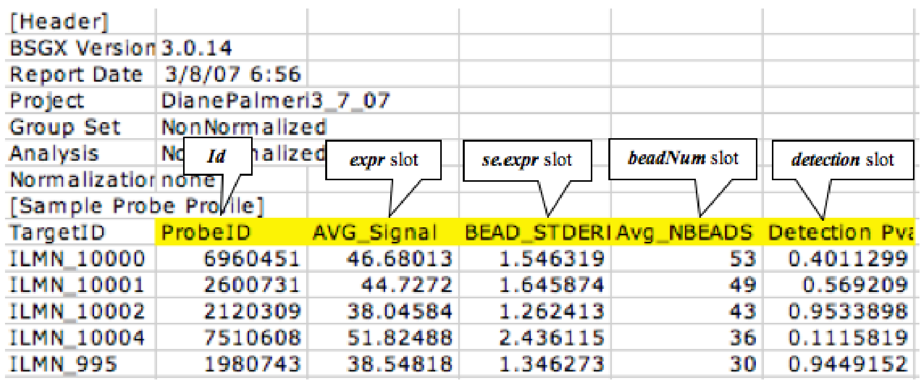
Figure 1: Object models in lumi package



Illumina Inc. BeadStudio version 1.4.0.1
Normalization = none
Array Content = 11188230_100CP_MAGE-ML.XML
Error Model = none
D e = 2/3 1
Local settings = en-U

TargetID	AVG_Signal-1	BEAD_STDEV-	Avg_NBEADS-	Detection-10	MIN_Signal-9
GI_1004708	179.5	9.7	19	0.97076323	182.5
GI_1004709	144.5	12.3	19	0.55569952	141.8
GI_1004709	699.7	31.9	18	1	811.9
GI_1004709	2069.9	78.1	14	1	2405.6
GI_1004709	163.7	6	34	0.86485123	595.1
GI_1004710	3487.6	112.6	15	1	4427.8
GI_1004710	212.4	34	13	0.99980148	227.4

(A) BeadStudio version 1



[Header]
BSGX Version 3.0.14
Report Date 3/8/07 6:56
Project DianePalmeri3_7_07
Group Set NonNormalized
Analysis No alized
Normalization none
[Sample Probe Profile]

TargetID	ProbeID	AVG_Signal	BEAD_STDERI	Avg_NBEADS	Detection Pva
ILMN_10000	6960451	46.68013	1.546319	53	0.4011299
ILMN_10001	2600731	44.7272	1.645874	49	0.569209
ILMN_10002	2120309	38.04584	1.262413	43	0.9533898
ILMN_10004	7510608	51.82488	2.436115	36	0.1115819
ILMN_995	1980743	38.54818	1.346273	30	0.9449152

(B) BeadStudio version 3

Figure 2: An example of the input data format

We strongly suggest outputting the header information when using BeadStudio, as shown in Figure 2.

If a lumi annotation library is provided, the `lumiR` function will automatically mapping the ProbeId or TargetID as nuID (see annotation section for more details), and keep the mapping information in the **featureData** of the **LumiBatch** object.

For convenience, another function `lumiR.batch` is designed to input files in batch. Basically it combines the output of each file. See the help of `lumiR.batch` for details.

```
> ## specify the file name
> # fileName <- 'Barnes_gene_profile.txt'          # Not Run
> ## load the data
> # x.lumi <- lumiR(fileName, lib='lumiHuamnV1')                # Not Run
```

Here, we just load the pre-saved example data, example.lumi, which is a subset of the experiment data package *lumiBarnes* in the Bioconductor. The example data includes four samples ”A01”, ”A02”, ”B01” and ”B02”. ”A” and ”B” represent different Illumina slides (8 microarrays on each slide), and ”01” and

"02" represent different samples. That means "A01" and "B01" are technique replicates at different slides, the same for "A02" and "B02".

```
> ## load example data (a LumiBatch object)
> data(example.lumi)
> ## summary of the example data
> example.lumi
```

```
Summary of BeadStudio output:
        Illumina Inc. BeadStudio version 1.4.0.1
        Normalization = none
        Array Content = 11188230_100CP_MAGE-ML.XML
        Error Model = none
        DateTime = 2/3/2005 3:21 PM
        Local Settings = en-US

Major Operation History:
            submitted            finished
1 2007-04-22 00:08:36 2007-04-22 00:10:36
2 2007-04-22 00:10:36 2007-04-22 00:10:38
3 2007-04-22 00:13:06 2007-04-22 00:13:10
4 2007-04-22 00:59:20 2007-04-22 00:59:36
                                             command lumiVersion
1           lumiR("../data/Barnes_gene_profile.txt")       1.1.6
2                             lumiQ(x.lumi = x.lumi)       1.1.6
3 addNuId2lumi(x.lumi = x.lumi, lib = "lumiHumanV1")       1.1.6
4            Subsetting 8000 features and 4 samples.       1.1.6

Object Information:
LumiBatch (storageMode: lockedEnvironment)
assayData: 8000 features, 4 samples
  element names: beadNum, detection, exprs, se.exprs
phenoData
  rowNames: A01, A02, B01, B02
  varLabels and varMetadata description:
    sampleID: The unique Illumina microarray Id
    label: The label of the sample
featureData
  featureNames: oZsQEQXp9ccVIlwoQo, 9qedFRd_5Cul.ueZeQ, ..., 33KnLHy.RFaieogAF4  (8000 tot
  fvarLabels and fvarMetadata description:
    TargetID: The Illumina microarray identifier
experimentData: use 'experimentData(object)'
Annotation: lumiHumanV1
```

4.2 Quality control of the raw data

The quality control of a **LumiBatch** object includes a data summary (the mean and standard deviation, sample correlation, detectable probe ratio of each sample (microarray)) and different quality control plots.

For BeadStudio version 3 output file, if it includes the control probe (gene) information. The controlData slot in LumiBatch class was added to keep the control probe (gene) information, and a QC slot to keep the quality control information.

`LumiQ` function will produce the data summary of a **LumiBatch** object and organize the results in a QC slot of **LumiBatch** object. When creating the **LumiBatch** object, the `LumiQ` function will be called to initialize the QC slot of the **LumiBatch** object.

Summary of the quality control information of example.lumi data. If the QC slot of the **LumiBatch** object is empty, function `lumiQ` will be automatically called to estimate the quality control information.

```
> ## summary of the quality control
> summary(example.lumi, 'QC')
```

```
Data dimension:  8000 genes x 4 samples

Summary of Samples:
                            A01    A02     B01     B02
mean                     8.3240  8.568  8.2580  8.3470
standard deviation       1.5580  1.686  1.7230  1.6690
detection rate(0.01)     0.5432  0.564  0.5774  0.5758
distance to sample mean 76.9500 65.280 88.3200 49.1100

Major Operation History:
            submitted            finished
1 2007-04-22 00:08:36 2007-04-22 00:10:36
2 2007-04-22 00:10:36 2007-04-22 00:10:38
                                  command lumiVersion
1 lumiR("../data/Barnes_gene_profile.txt")       1.1.6
2                  lumiQ(x.lumi = x.lumi)       1.1.6
```

The S4 method `plot` can produce the quality control plots of **LumiBatch** object. The quality control plots includes: the density plot (Figure 3), box plot (Figure 4), pairwise correlation between microarrays (Figure 5), pairwise MAplot between microarrays (Figure 6), density plot of coefficient of varience, (Figure 7), and the sample relations (Figure 8). More details are in the help of `plot,LumiBatch-method` function. Most of these plots can also be plotted by the extended general functions: `density` (for density plot), `boxplot`, `MAplot`, `pairs` and `plotSampleRelation`.

Figure 3 shows the density plot of the **LumiBatch** object by using `plot` or `density` functions.

```
> ## plot the density
> plot(example.lumi, what='density')
> ## or
> density(example.lumi)
```

Figure 4 shows the box plot of the **LumiBatch** object by using `plot` or `boxplot` functions.

```
> ## plot the box plot
> plot(example.lumi, what='boxplot')
> ## or
> boxplot(example.lumi)
```

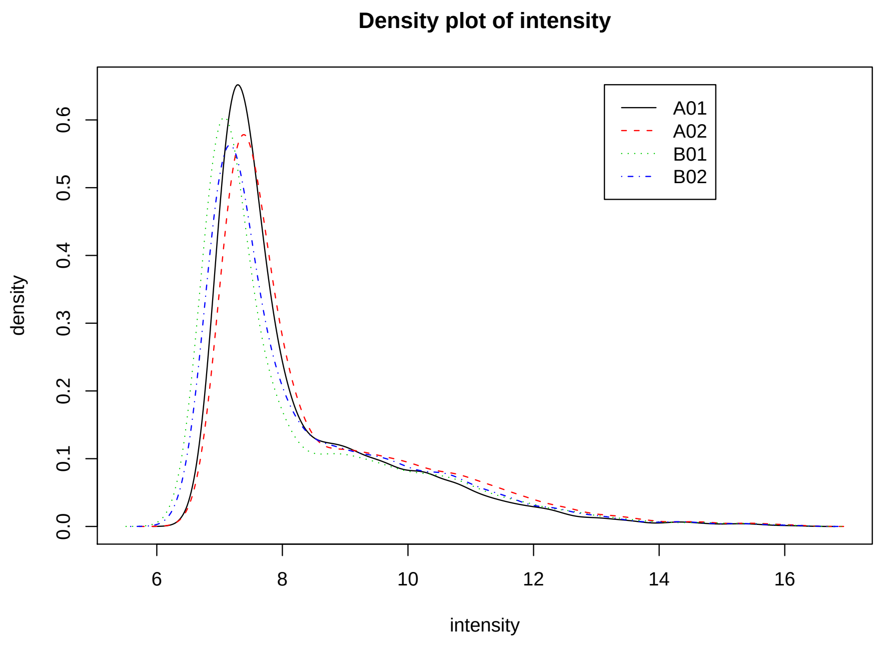


Figure 3: Density plot of Illumina microarrays before normalization

Figure 5 shows the pairwise sample correlation of the **LumiBatch** object by using `plot` or `pairs` functions.

```
> ## plot the pair plot
> plot(example.lumi, what='pair')
> ## or
> pairs(example.lumi)
```

Figure 6 shows the MA plot of the **LumiBatch** object by using `plot` or `MAplot` functions.

```
> ## plot the MAplot
> plot(example.lumi, what='MAplot')
> ## or
> MAplot(example.lumi)
```

The density plot of the coefficient of variance of the **LumiBatch** object. See Figure 7. Figure 7 shows the density plot of the coefficient of variance of the **LumiBatch** object by using `plot` function.

Figure 8 shows the sample relations using hierarchical clustering.

Figure 9 shows the sampleRelation using MDS. The color of the sample is based on the sample type, which is `"01", "02", "01", "02"` for the sample data. Please see the help of `plotSampleRelation` and `plot-methods` for more details.

```
> ## plot the sample relations
> plot(example.lumi, what='sampleRelation', method='mds', color=c("01", "02", "01", "02"))
```

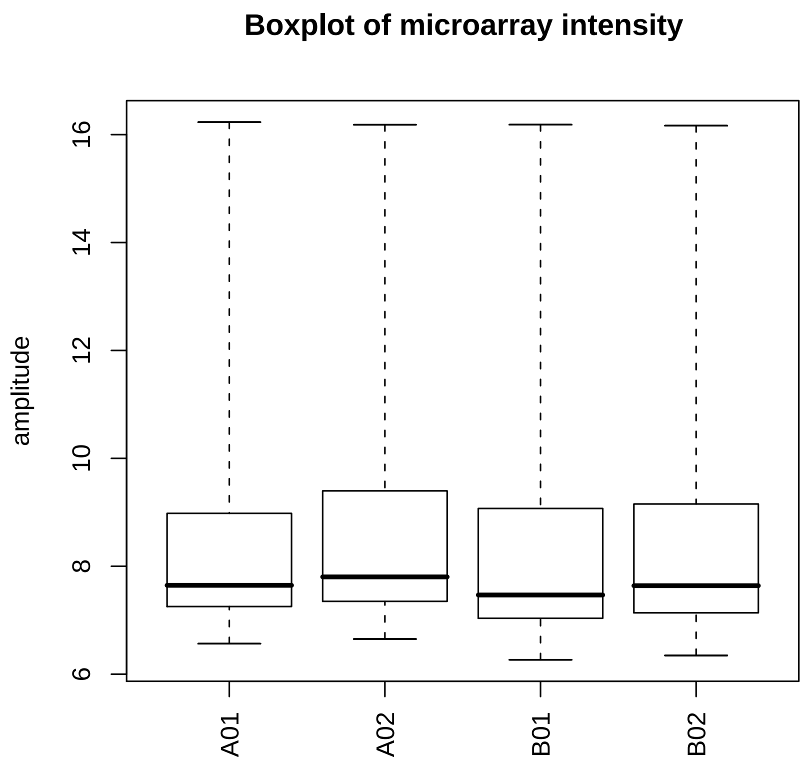


Figure 4: Density plot of Illumina microarrays before normalization

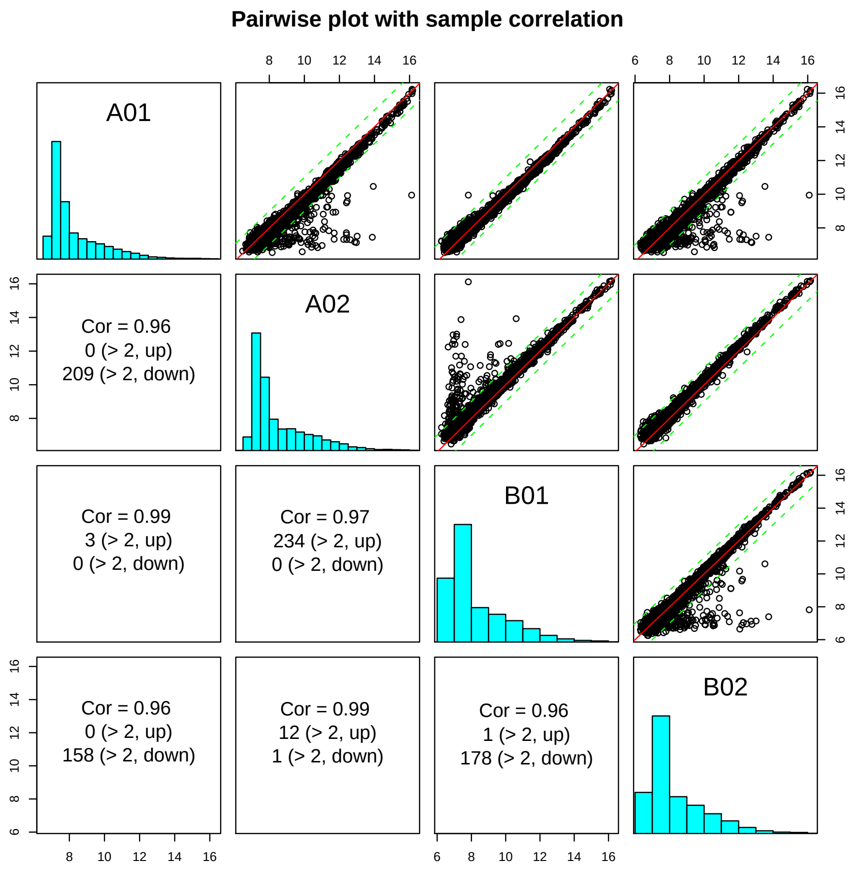


Figure 5: Pairwise plot with microarray correlation before normalization

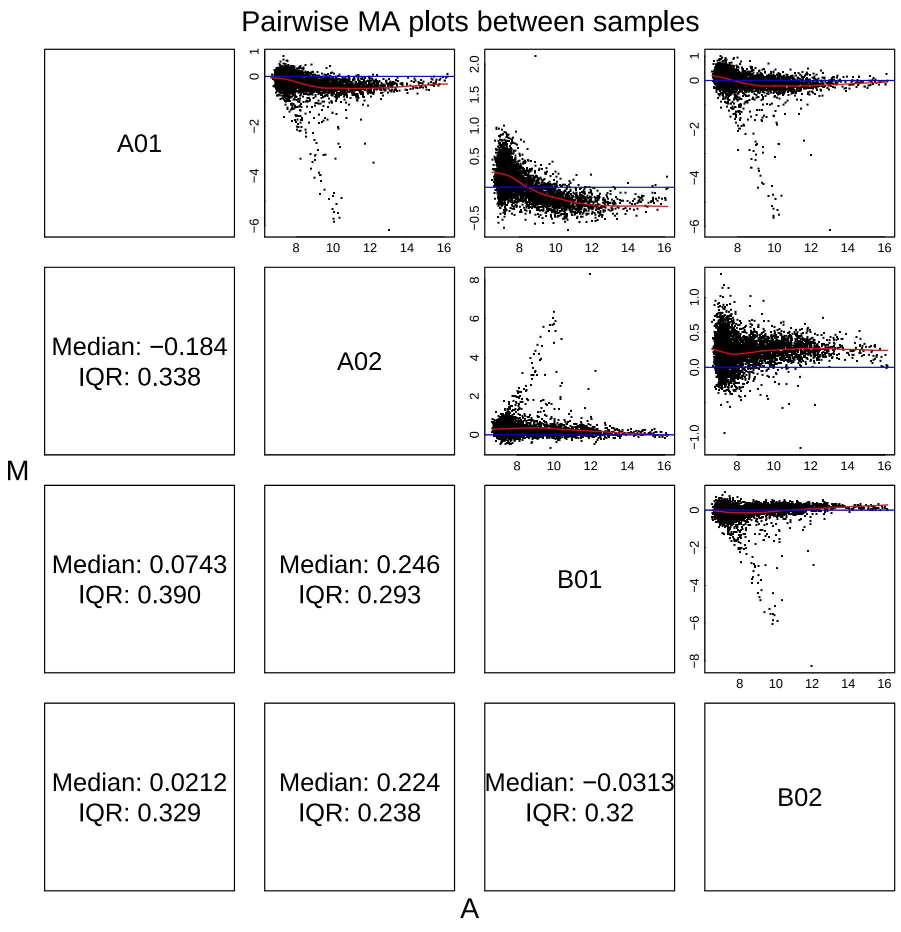


Figure 6: Pairwise MAplot before normalization

```
> ## density plot of coefficient of varience
> plot(example.lumi, what='cv')
```

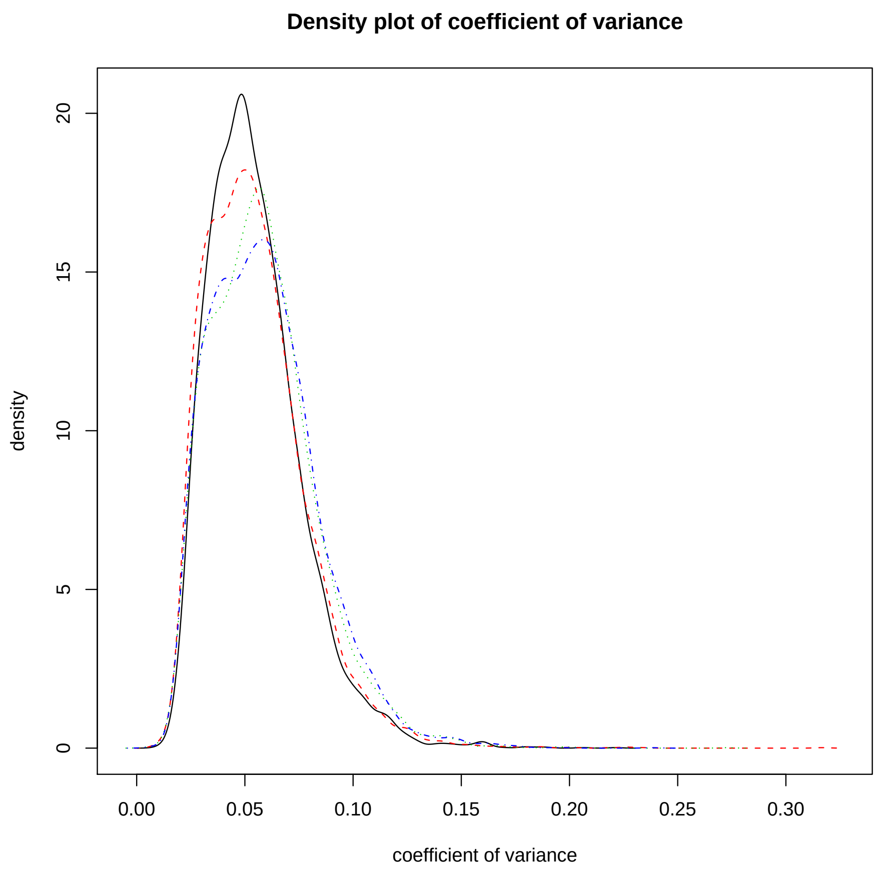


Figure 7: Density Plot of Coefficient of Varience

```
> plot(example.lumi, what='sampleRelation')
```

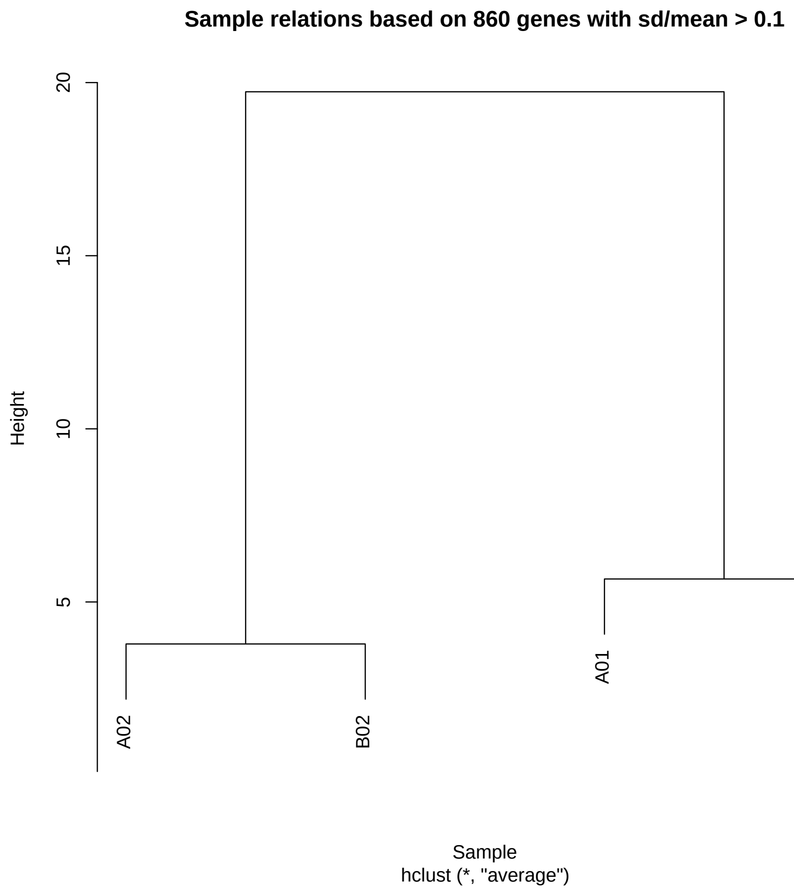


Figure 8: Sample relations before normalization

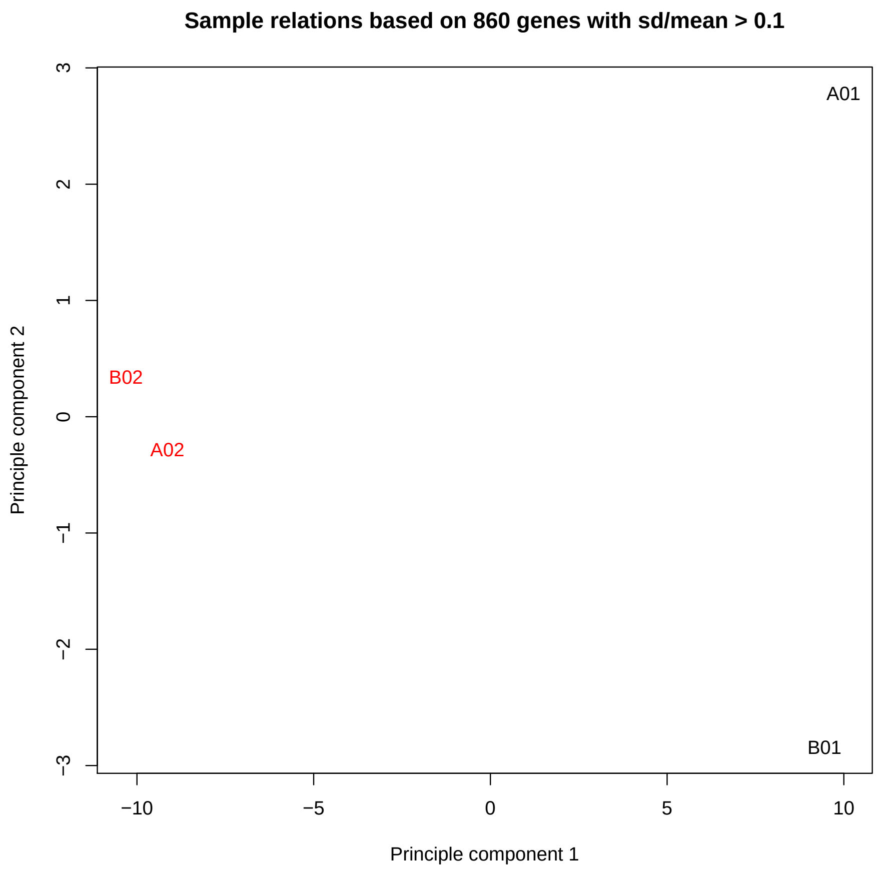


Figure 9: Sample relations before normalization

```
> ## or
> plotSampleRelation(example.lumi, method='mds', color=c("01", "02", "01", "02"))
```

4.3 Background correction

The *lumi* package provides `lumiB` function for background correction. We suppose the BeadStudio output data has been background corrected. As a result, no sophisticated background corrected needed. As both `vst` and `log2` transforms require the expression value to be positive. The default background correction method ('forcePositive') just adds an offset (minus minimum value plus one) if there is any negative values to force all expression values to be positive. It does nothing if all expression values are positive. If users are more interested in the low level background correction, please refer to the package *beadarray* for more details. Users can also provide their own background correction function with a LumiBatch Object as the first argument and return a LumiBatch Object with background corrected. See `lumiB` help document for more details.

4.4 Variance stabilizing transform

Variance stabilization is critical for subsequent statistical inference to identify differential genes from microarray data. We devised a variance-stabilizing transformation (VST) by taking advantages of larger number of technical replicates available on the Illumina microarray. Please see [1] for details of the algorithm.

Because the STDEV (or STDERR) columns of the BeadStudio output file is the standard error of the mean of the bead intensities corresponding to the same probe. (Thanks Gordon Smyth kindly provided this information!). As the variance stabilization (see help of `vst` function) requires the information of the standard deviation instead of the standard error of the mean, the value correction is required. The corrected value will be x * sqrt(N), where x is the old value (standard error of the mean), N is the number of beads corresponding to the probe. The parameter 'stdCorrection' of `lumiT` determines whether to do this conversion and is effective only when the 'vst' method is selected. By default, the parameter 'stdCorrection' is TRUE.

Function `lumiT` performs variance stabilizing transform with both input and output being **LumiBatch** object.

Do default VST variance stabilizing transform

```
> ## Do default VST variance stabilizing transform
> lumi.T <- lumiT(example.lumi)
```

```
2007-10-03 14:44:43 , processing array  1
2007-10-03 14:44:43 , processing array  2
2007-10-03 14:44:43 , processing array  3
2007-10-03 14:44:43 , processing array  4
```

The `plotVST` can plot the transformation function of VST, see Figure 10, which is close to log2 at high expression values, see Figure 11. Function `lumiT` also provides options to do `"log2"` or `"cubicRoot"` transform. See help of `lumiT` for details.

```
> ## plot VST transformation
> trans <- plotVST(lumi.T)
> ## compare the log2 and VST transform
> matplot(log2(trans$untransformed), trans$transformed, main='compare VST and log2 transfo
```

4.5 Data normalization

We proposed a robust spline normalization (RSN) algorithm, which combines the features of quanitle and loess nor-malization. The advantages of quantile normalization include computational efficiency and preserving the rank order of genes. However, the intensity transformation of a quantile normalization is discontinuous because the normalization forces the intensity values for different samples (microarrrays) having exactly the same distribution. This can cause small differences among intensity values to be lost. In contrast, the loess or spline normalization provides a continuous transformation. However, these methods cannot ensure that the rank of the probes remain unchanged across samples. Moreover, the loess normalization assumes the majority of the genes measured

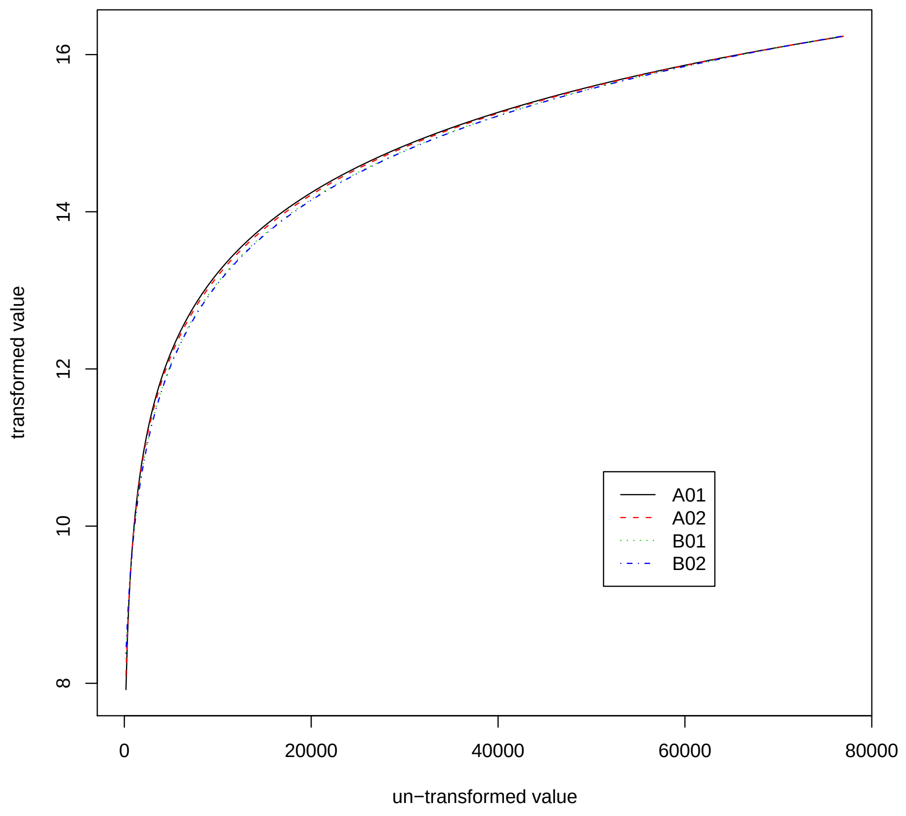


Figure 10: VST transformation

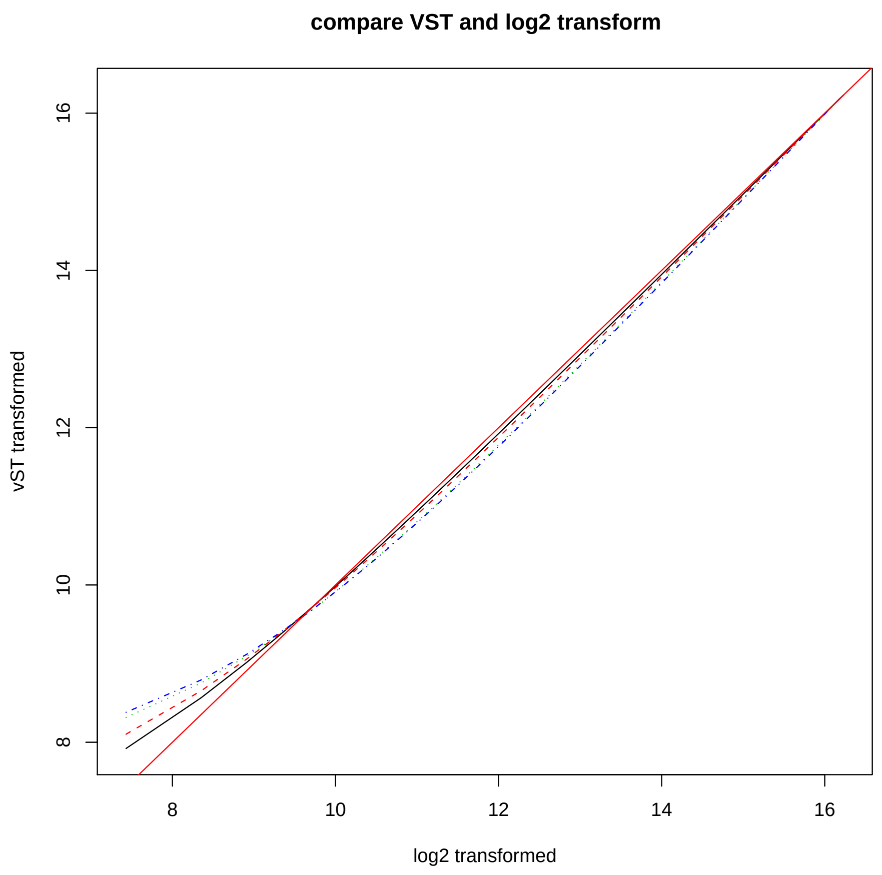


Figure 11: Compare VST and log2 transform

by the probes are non-differentially expressed and their distribution is approximately symmetric, which may not be a good assumption. To address some of these concerns, we developed a Robust Spline Normalization (RSN) method, which combines features from loess and quantile normalization methods. We use a monotonic spline to calibrate one microarray to the reference microarray. To increase the robustness of the spline method, we down-weight the contributions of probes of putatively differentially expressed genes. The probe intensities that are from potentially differentially expressed genes are heuristically determined as follows: First, we run a quantile normalization. Next, we estimate the fold-change of a gene measured by a probe based on the quantile-normalized data. The weighting factor for a probe is calculated based on a Gaussian window function. More details will be shown in a separate manuscript.

By default, function `lumiN` performs robust spline normalization (RSN) algorithm. `lumiN` also provides options to do `"quantile", "loess", "vsn"` normalization. See help of `lumiN` for details.

Do default RSN between microarray normaliazation

```
> ## Do RSN between microarray normaliazation
> lumi.N <- lumiN(lumi.T)
```

```
2007-10-03 14:44:44 , processing array  1
2007-10-03 14:44:44 , processing array  2
2007-10-03 14:44:44 , processing array  3
2007-10-03 14:44:44 , processing array  4
```

Users can also easily select other normalization method. For example, the following command will run quantile between microarray normaliazation.

```
> ## Do quantile between microarray normaliazation
> lumi.N <- lumiN(lumi.T, method='quantile')        ## Not Run
```

4.6 Quality control after normalization

To make sure the data quality meets our requirement, we do a second round of quality control of normalized data with different QC plots. Compare the plots before and after normalization, we can clearly see the improvements.

```
> ## Do quality control estimation after normalization
> lumi.N.Q <- lumiQ(lumi.N)
> ## summary of the quality control
> summary(lumi.N.Q, 'QC')                 ## summary of QC
```

```
Data dimension:  8000 genes x 4 samples

Summary of Samples:
                            A01    A02     B01     B02
mean                     8.9290  8.930  8.9280  8.9280
standard deviation       1.2610  1.262  1.2620  1.2610
detection rate(0.01)     0.5432  0.564  0.5774  0.5758
distance to sample mean 14.1400 13.890 14.1600 14.3500

Major Operation History:
```

```
> plot(lumi.N.Q, what='density')                 ## plot the density
```

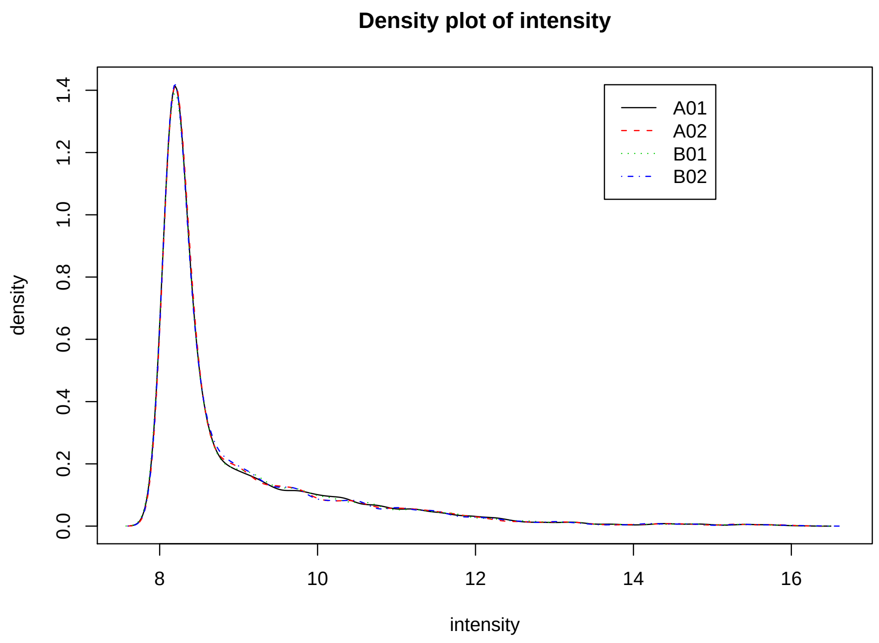


Figure 12: Density plot of Illumina microarrays after normalization

```
            submitted            finished
1 2007-04-22 00:08:36 2007-04-22 00:10:36
2 2007-04-22 00:10:36 2007-04-22 00:10:38
3 2007-04-22 00:13:06 2007-04-22 00:13:10
4 2007-04-22 00:59:20 2007-04-22 00:59:36
5 2007-10-03 14:44:43 2007-10-03 14:44:43
6 2007-10-03 14:44:43 2007-10-03 14:44:44
7 2007-10-03 14:44:44 2007-10-03 14:44:44
                                              command lumiVersion
1           lumiR("../data/Barnes_gene_profile.txt")       1.1.6
2                             lumiQ(x.lumi = x.lumi)       1.1.6
3 addNuId2lumi(x.lumi = x.lumi, lib = "lumiHumanV1")       1.1.6
4            Subsetting 8000 features and 4 samples.       1.1.6
5                       lumiT(x.lumi = example.lumi)       1.4.0
6                             lumiN(x.lumi = lumi.T)       1.4.0
7                             lumiQ(x.lumi = lumi.N)       1.4.0
```

4.7 Encapsulate the processing steps

The `lumiExpresso` function is to encapsulate the major functions of Illumina preprocessing. It is organized in a similar way as the `expresso` function in *affy* package. The following code basically did the same processing as the previous

```
> plot(lumi.N.Q, what='boxplot')               ## box plot
```

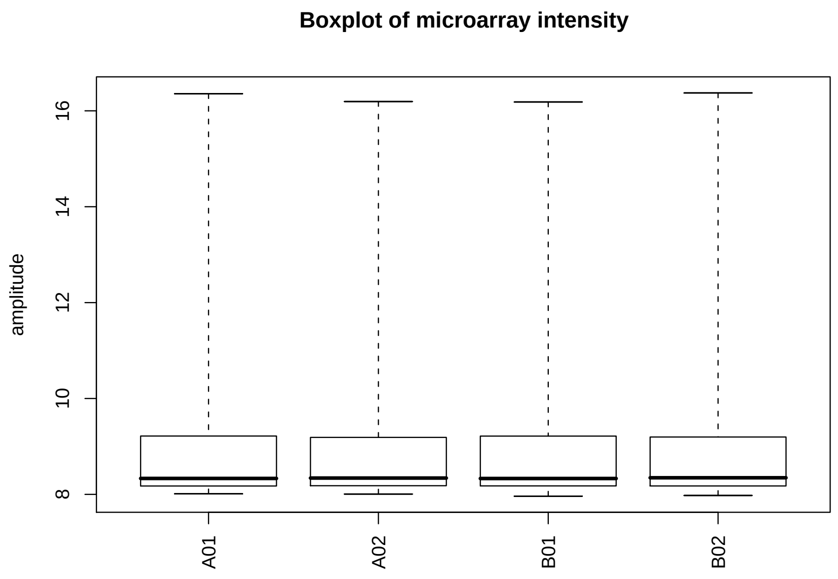


Figure 13: Density plot of Illumina microarrays after normalization

```
> plot(lumi.N.Q, what='pair')                              ## pairwise plots
```

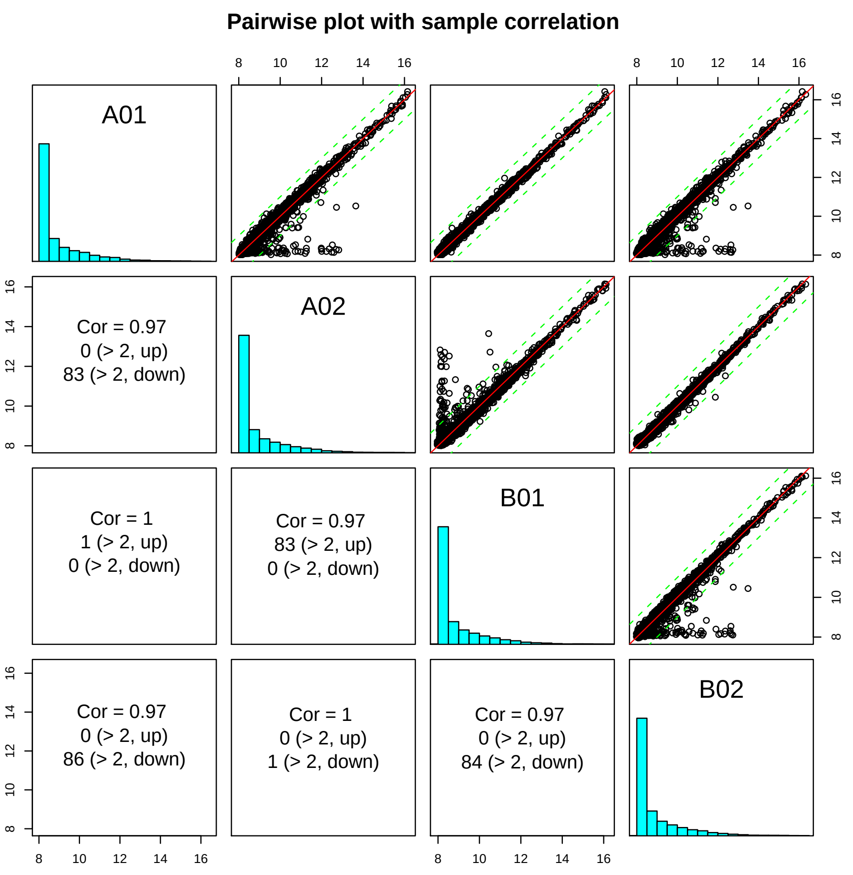


Figure 14: Pairwise plot with microarray correlation after normalization

```
> plot(lumi.N.Q, what='MAplot')                    ## plot the pairwise MAplot
```

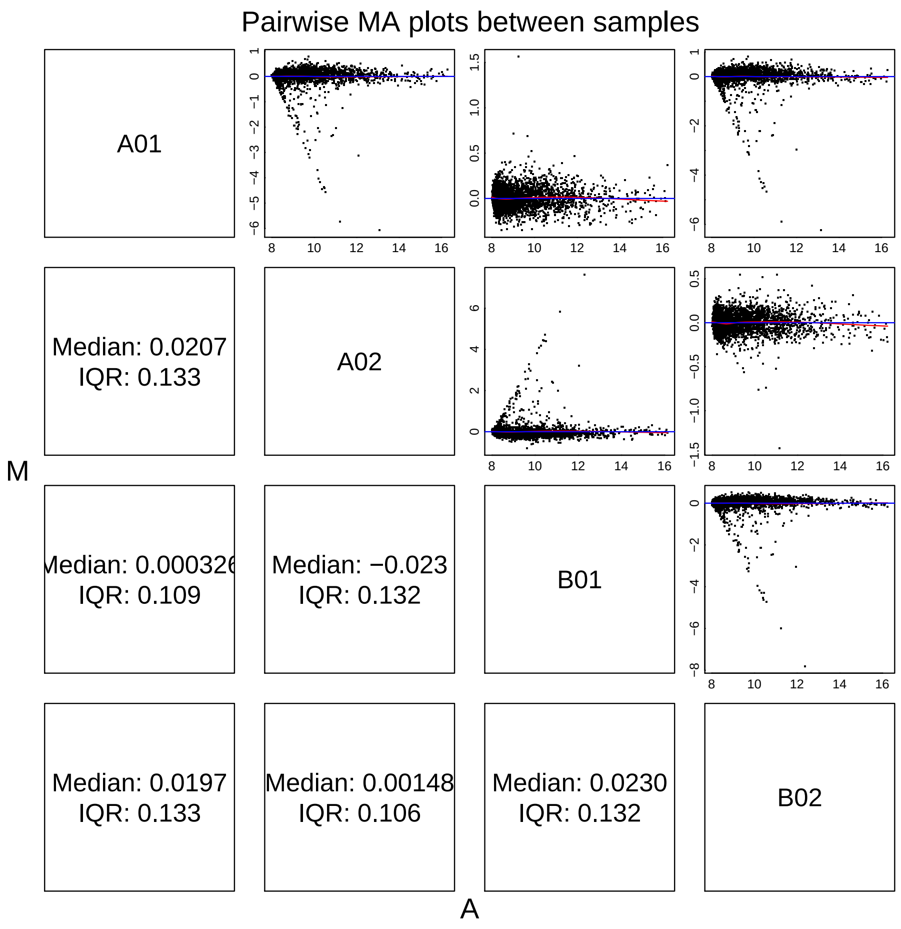


Figure 15: Pairwise MAplot after normalization

```
> ## plot the sampleRelation using hierarchical clustering
> plot(lumi.N.Q, what='sampleRelation')
```

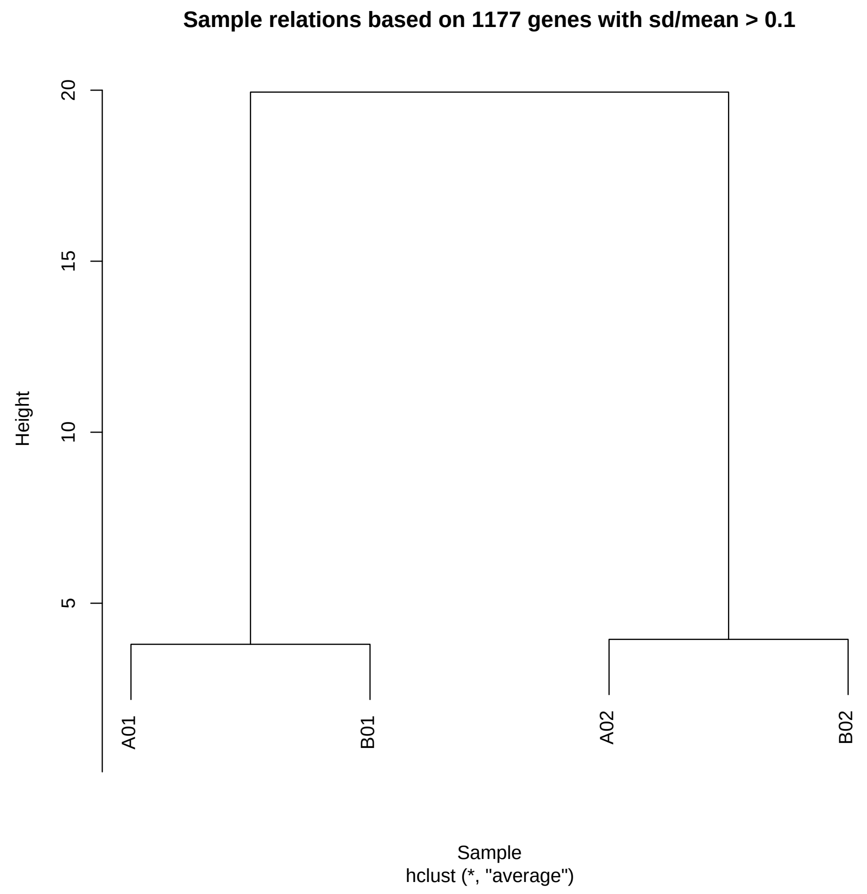


Figure 16: Sample relations after normalization

```
> ## plot the sampleRelation using MDS
> plot(lumi.N.Q, what='sampleRelation', method='mds', color=c("01", "02", "01", "02"))
```

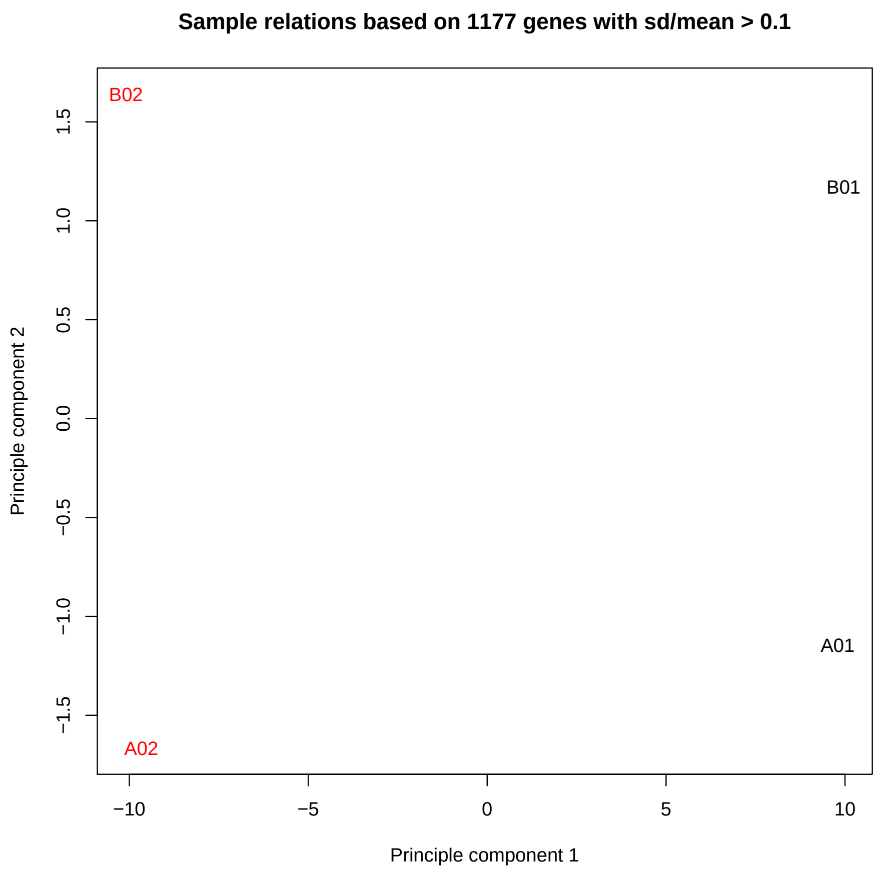


Figure 17: Sample relations after normalization

multi-steps and produced the same results lumi.N.Q.

```
> ## Do all the default preprocessing in one step
> lumi.N.Q <- lumiExpresso(example.lumi)

Variance Stabilizing Transform method: vst
Normalization method: rsn

Variance stabilizing ...
2007-10-03 14:44:56 , processing array  1
2007-10-03 14:44:56 , processing array  2
2007-10-03 14:44:56 , processing array  3
2007-10-03 14:44:56 , processing array  4
done.
Normalizing ...
2007-10-03 14:44:56 , processing array  1
2007-10-03 14:44:57 , processing array  2
2007-10-03 14:44:57 , processing array  3
2007-10-03 14:44:57 , processing array  4
done.
Quality control after preprocessing ...
done.
```

Users can easily customize the processing parameters. For example, if the user wants to do ”quantile” normalization instead of ”rsn” normalization, the user can run the following code. For more details, please read the help document of `lumiExpresso` function.

```
> ## Do all the default preprocessing in one step
> lumi.N.Q <- lumiExpresso(example.lumi, normalize.param=list(method='quantile'))

Variance Stabilizing Transform method: vst
Normalization method: quantile

Variance stabilizing ...
2007-10-03 14:44:57 , processing array  1
2007-10-03 14:44:57 , processing array  2
2007-10-03 14:44:57 , processing array  3
2007-10-03 14:44:57 , processing array  4
done.
Normalizing ...
done.
Quality control after preprocessing ...
done.
```

4.8 Inverse VST transform to the raw scale

Figure 11 shows VST is very close to log2 in the high expression range. In convenience, users usually can directly use `2^x` to approximate the data in raw scale and estimate the fold-change. For the users concern more in the low expression range, we also provide the function `inverseVST` to resume the data in the raw scale. Need to mention, the inverse transform should be performed after statistical analysis, or else it makes no sense to transform back and forth. The `inverseVST` function can directly applied to the **LumiBatch** object after `lumiT` with VST transform, or VST transform plus RSN normalization (default method of `lumiN`). For the RSN normalized data, the inverse transform is based on the parameters of the Target Array because the Target Array is the benchmark data and is not changed after normalization. Other normalization methods, like quantile or loess, will change the values of all the arrays. As a result, no inverse VST transform available for them. Users may use some kind of approximation for the quantile normalized data by themselves. Here we just provide some examples of VST parameters retrieving and inverse VST transform.

```
> ## Parameters of VST transformed LumiBatch object
> names(attributes(lumi.T))
```

```
 [1] "history"           "controlData"       "QC"
 [4] "assayData"         "phenoData"         "featureData"
 [7] "experimentData"    "annotation"        ".__classVersion__"
[10] "class"             "vstParameter"      "transformFun"
```

```
> ## VST parameters: "vstParameter"  and  "transformFun"
> attr(lumi.T, 'vstParameter')
```

```
           a          b        g Intercept
A01 2.396259 0.02244804 1.480568  4.166654
A02 3.574381 0.02063079 1.505113  4.094366
B01 6.513429 0.02172944 1.554504  3.614302
B02 6.878816 0.02030299 1.566239  3.626431
```

```
> attr(lumi.T, 'transformFun')
```

```
    A01     A02     B01     B02
"asinh" "asinh" "asinh" "asinh"
```

```
> ## Parameters of VST transformed and RSN normalized LumiBatch object
> names(attributes(lumi.N))
```

```
 [1] "history"           "controlData"       "QC"
 [4] "assayData"         "phenoData"         "featureData"
 [7] "experimentData"    "annotation"        ".__classVersion__"
[10] "class"             "vstParameter"      "transformFun"
[13] "targetArray"
```

```
> ## VSN "targetArray" , VST parameters: "vstParameter"  and  "transformFun"
> attr(lumi.N, 'vstParameter')
```

```
         a          b          g  Intercept
6.51342851 0.02172944 1.55450441 3.61430210
```

```
> attr(lumi.N, 'transformFun')
```

```
    B01
"asinh"
```

```
> ## After doing statistical analysis of the data, users can recover to the raw scale for
> ## Inverse VST to the raw scale
> lumi.N.raw <- inverseVST(lumi.N)
```

5 Handling large data sets

Several users asked about processing large data set, e.g., over 100 samples. Directly handling such big data set usually will cause ”out of memory” error in most computers. In this case, when read the BeadStudio output file, we can ignore the ”beadNum” (related columns. The function `lumiR` provides a parameter called ”columnNameGrepPattern”. we can set the string grep pattern of ”detection” and ”beadNum” as NA. You can also ignore ”detection” columns. However, the ”detection” information is useful for the estimation of present count of each probe and used in the VST parameter estimation.

Here is some example code:

```
## load the data with empty detection and beadNum slots
>  x.lumi <- lumiR("fileName.txt", columnNameGrepPattern=list(beadNum=NA))
```

Another good news is that the `vst` and `rsn` functions in the *lumi* package can sequentially process the data and handle such large data set.

The solution can be like this:

1. Read the data file by smaller batches (e.g. 10 or just one by one), and then do the variance stabilization, i.e., `lumiT` function, for each data batch.

2. Pick one sample as the target array for normalization and then using ”RSN” normalization method to normalize all batches of data using the same target array.

3. Combine the normalized data. (In order to save memory, the user can first remove those probes not expressed in all samples.)

In the `rsn` function, there is a parameter called ”targetArray”, which is the model for other chips to normalize. It can be a column index, a vector or a LumiBatch object with one sample. In our case, we need to use one LumiBatch object with one sample as the ”targetArray”. The selection of the target array is flexible. We suggest to choose the one most similar to the mean of all samples. For convenience, we can also just select the first sample as ”targetArray” (suppose it has no quality problem). The selected target array will also be used for all other data batches. Since different data batches use the same target array as model, the results are comparable and can be combined!

Here is the example code:

```
## Read in the Batch ith data file, suppose named as "fileName.i.txt"
> x.lumi.i <- lumiR("fileName.i.txt")
## variance stabilization (using vst or log2 transform)
> x.lumiT.i <- lumiT(x.lumi.i)
## select the "targetArray"
## This target array will also be used for other batches of data.
## For convenience, here we just select the first sample as targetArray.
> targetArray <- x.lumiT.i[,1]
## Do RSN normalization
> x.lumiN.i <- lumiN(x.lumiT.i, targetArray=targetArray)
```

The normalized data batches can be combined by using function Rfunction-combine(x, y).

6 Performance comparison

We have selected the Barnes data set [3], which is a series dilution of two tissues at five different dilutions, to compare different preprocessing methods. In order to better compare the algorithms, we selected the samples with the smallest dilution difference (the most challenging comparison), i.e., the samples with the dilution ratios of 100:0 and 95:5 (each condition has two technical replicates) for comparison. For the Barnes data set, because we do not know which of the signals are coming from 'true' differentially expressed genes, we cannot use an ROC curve to compare the performance of different algorithms. Instead, we evaluated the methods based on the concordance of normalized intensity profile and real dilution profile of the selected probes. More detailed evaluations with other criteria and based on other data sets can by found in our paper [1].

Following Barnes et al. (2005)[3], we defined a concordant gene (really a concordant probe) as a signal from a probe with a correlation coefficient larger than 0.8 between the normalized intensity profile and the real dilution profile (five dilution ratios with two replicates at each dilution ratio). If a selected differentially expressed probe is also a concordant one, it is more likely to be truly differentially expressed. Figure 18 shows the percentage of concordant probes among the selected probes, which were selected by ranking the probes' p-value (calculated based on *limma* package) from low to high. We can see the VST transformed data outperforms the Log2-transformed and VSN processed data. For the normalization methods, RSN and quantile normalization have similar performance for the VST transformed data, and RSN outperforms quantile for the Log transformed data.

Please see another vignette in the lumi package: `"lumi_vST_evaluation.pdf"` for more details of the evaluation of VST (Variance Stabilizing Transformation).

7 Gene annotation

Illumina microarray provides the TargetID or the ProbeID to identify the measurements. The TargetID is used as a public identifier by Illumina and is supposed to be stable. The problem of the TargetID is that it can correspond to several different probes, which are supposed to match the same gene. Due to the binding affinity difference or alternative splicing, the probes corresponding the the sample TargetID may have quite different expression levels and patterns. If we use TargetID to identify the measurements, then we cannot differentiate the difference between these probes. Another problem of using the TargetID is that the mapping between the TargetID and probes could be changed with our better understanding of the gene. Moreover, the TargetID used by Illumina microarray is not consistent among different versions of arrays. For instance, the same 50mer sequence has two different TargetIDs used by Illumina: `"GI_21070949-S"` in the `Mouse_Ref-8_V1` chip and `"scl022190.1_154-S"` in the `Mouse-6_V1` chip. This causes difficulties when combining clinical microarray data collected over time using different versions of the chips.

In order to get unique mapping between microarray measurements and probes, using ProbeID is preferred. However, the ProbeID of Illumina is not stable. It is changing between different versions, even between different batches of Illumina microarrays. To solve these problems, we designed a nucleotide universal iden-

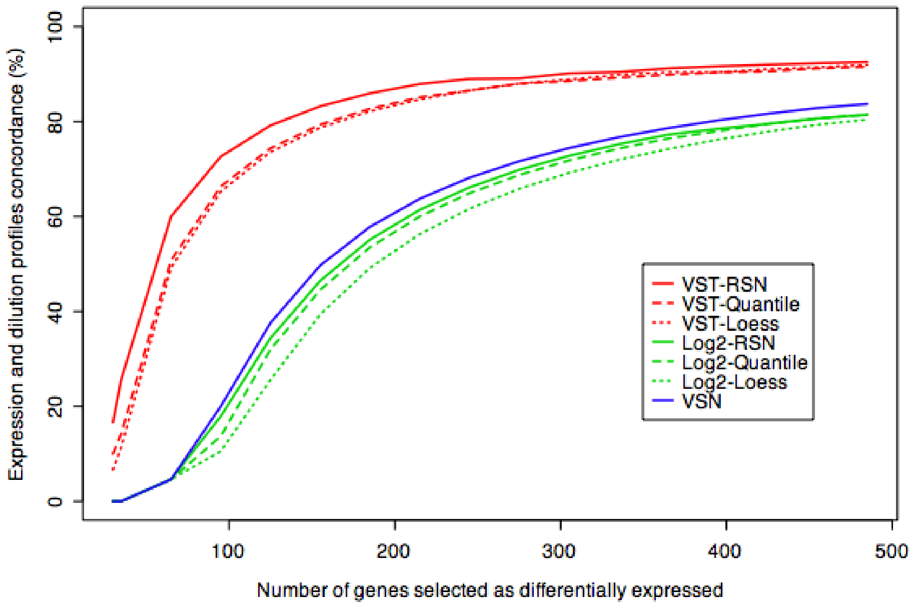


Figure 18: Comparison of the concordance between the expression and dilution profiles of the selected differentially expressed genes

tifier (nuID), which encodes the 50mer oligonucleotide sequence and contains error checking and self-identification code. By using nuID, all the problems mentioned above can be easily solved. For details, please read [2].

7.1 Examples of nuID

```
> ## provide an arbitrary nucleotide sequence as an example
> seq <- 'ACGTAAATTTCAGTTTAAAACCCCCCG'
> ## create a nuID for it
> id <- seq2id(seq)
> print(id)
```

```
[1] "YGwPOvwBVW"
```

The original nucleotide sequence can be easily recovered by `id2seq`

```
> id2seq(id)
```

```
[1] "ACGTAAATTTCAGTTTAAAACCCCCCG"
```

The nuID is self-identifiable. `is.nuID` can check the sequence is nuID or not. A real nuID

```
> is.nuID(id)
```

```
[1] TRUE
```

An random sequence

```
> is.nuID('adfqeqe')

[1] FALSE
```

7.2 Illumina microarray annotation packages

Because all the Illumina microarrays use 50-mers, by using the nuID universal identifier, we are able to build one annotation database for different versions of the human (or other species) chips. Moreover, the nuID can be directly converted to the probe sequence, and used to get the most updated refSeq matches and annotations. Annotation packages indexed by nuID for different Illumina expression chips can be downloaded from Bioconductor.

The Illumina annotation packages are produced by using *AnnBuilder* with small modification. As a result, the format of the package is the same as Affymetrix annotation package, lots of packages designed for Affymetrix can also be used for Illumina annotation package. The mappings between TargetID to nuID and ProbeID to nuID are also included in the Illumina annotation packages. Thus, we can easily mapping between the nuID and TargetID or ProbeID.

Need to mention, currently there are two sets of Illumina annotation packages in Bioconductor. The Illumina annotation packages mentioned here are named as ”lumixxxx”, e.g. ”lumiHumanV2” and are maintained by us. There are another set of packages, named as ”illuminaxxxx”. These packages are indexed based on Illumina TargetID. They can also be used together with *lumi* package if the BeadStudio output file is also indexed with TargetID (file name includes `"gene_profile"`). They have no relation with nuID and cannot be used when the BeadStudio output files are indexed with Illumina ProbeID (file name includes `"probe_profile"`).

Here is some examples:

```
> ## load lumi annotation package
> lib <- 'lumiHumanV1'                # Huamn lumi annotation package version one
> if(require(GO) & require(annotate) & require(lib, character.only=TRUE)) {
+         GOId <- 'GO:0004816'                # asparagine-tRNA ligase activity
+         probe <- lookUp(GOId, lib, 'GO2ALLPROBES')
+         # probes under 'GO:0004816'  category
+         probe
+ }

$`GO:0004816`
                  IEA                   TAS                   IEA 
"WVUU7XyNw3ucXzwdEk" "WVUU7XyNw3ucXzwdEk" "inoI_vCgCRVU6SIR5E" 

> # specify a nuID
> nuId <- 'WVUU7XyNw3ucXzwdEk'
> if (require(annotate) & require(lib, character.only=TRUE)) {
+        # get the gene symbol of nuId
+        getSYMBOL(nuId, lib)
+ }

WVUU7XyNw3ucXzwdEk 
            "NARS" 
```

Mapping from nuID to TargetID

```
> nuId <- "WVUU7XyNw3ucXzwdEk"
> if (require(lib, character.only=TRUE))
+         nuID2targetID(nuId, lib=lib)
```

```
$WVUU7XyNw3ucXzwdEk
[1] "GI_7262387-S"
```

Mapping from TargetID to nuID

```
> targetID <- "GI_7262387-S"
> if (require(lib, character.only=TRUE))
+         targetID2nuID(targetID, lib=lib)
```

```
        GI_7262387-S
"WVUU7XyNw3ucXzwdEk"
```

7.3 Transfer Illumina identifier annotated data into nuID annotated

As the annotation packages include the mappings between TargetID to nuID and ProbeID to nuID. We can easily map the targetID (or Probe Id) to nuID. The function can automatically check whether targetID or Probe Id was used in the text data file, and convert them as nuID. Function `addNuId2lumi` can transfer a TargetID or Probe Id indexed **LumiBatch** object as an nuID indexed **LumiBatch** object. And the mapping between the nuID and TargetID is kept in the featureData of the **LumiBatch** object. If a **LumiBatch** object has already been nuID indexed, the function will do nothing.

```
> if (require(lumiHumanV1)) {
+         lumi.N <- addNuId2lumi(lumi.N, lib='lumiHumanV1')
+ }
```

```
[1] "The lumiBatch object is already nuID annotated!"
```

The **LumiBatch** object can also be directly transferred as nuID indexed at the very beginning of inputting data using `lumiR`. For example:

```
> ## load the data
> # example.lumi <- lumiR(fileName, lib='lumiHumanV1')           # Not run
```

8 A use case: from raw data to functional analysis

Figure 19 shows the data processing flow chart of the use case. Since the classes in *lumi* package are inherited from class **ExpressionSet**, packages and functions compatible with class **ExpressionSet** or accepting matrix as input all can be used for *lumi* results. Here we just give two examples: using *limma* to identify differentiated genes and using *GOstats* to annotate the significant genes.

We use the Barnes data set [3] as an example, which has be created as a Bioconductor experiment data package *lumiBarnes*. The Barnes data set

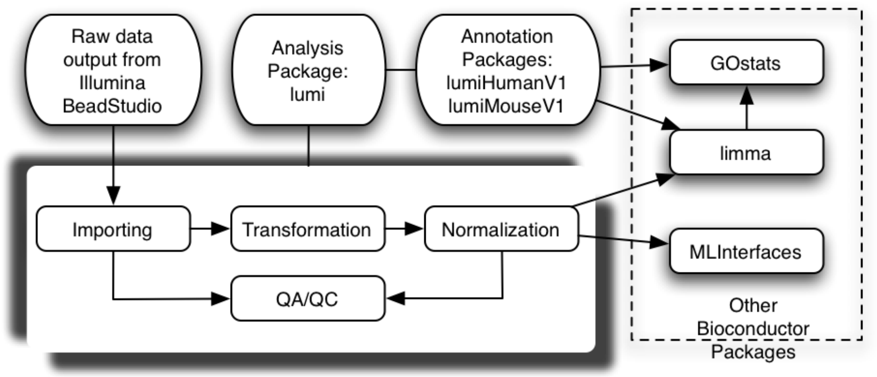

Raw data output from Illumina BeadStudio
Analysis Package: lumi
Annotation Packages: lumiHumanV1 lumiMouseV1
GOstats
limma
MLInterfaces
Other Bioconductor Packages
Importing
Transformation
Normalization
QA/QC

Figure 19: Flow chart of the use case

measured a dilution series of two human tis-sues, blood and placenta. It includes six samples with the titration ratio of blood and placenta as 100:0, 95:5, 75:25, 50:50, 25:75 and 0:100. The samples were hybridized on HumanRef-8 BeadChip (Illumina, Inc) in duplicate. We select samples with titration ratio, 100:0 and 95:5 (each has two technique replicates) in this data set to evaluate the detection of differential expressions.

8.1 Preprocess the Illumina data

```
> library(lumi)
> ## specify the file name
> # fileName <- 'Barnes_gene_profile.txt'  #  Not run
> ## load the data
> # example.lumi <- lumiR(fileName, lib='lumiHumanV1')         # Not run

> ## load saved data
> data(example.lumi)
> ## sumary of the daa
> example.lumi
> ## summary of quality control information
> summary(example.lumi, 'QC')

> ## preprocessing and quality control after normalization
> lumi.N.Q <- lumiExpresso(example.lumi, QC.evaluation=TRUE)
> ## summary of quality control information after preprocessing
> summary(lumi.N.Q, 'QC')
```

8.2 Identify differentially expressed genes

Identify the differentiated genes based on moderated t-test using *limma*.

Retrieve the normalized data

```
> dataMatrix <- exprs(lumi.N)
```

To speed up the processing and reduce false positives, remove the unexpressed genes

```
> presentCount <- detectionCall(example.lumi)
> selDataMatrix <- dataMatrix[presentCount > 0,]
> selProbe <- rownames(selDataMatrix)
```

```
> ## Specify the sample type
> sampleType <- c('100:0', '95:5', '100:0', '95:5')
> if (require(limma)) {
+         ## compare '95:5' and '100:0'
+         design <- model.matrix(~ factor(sampleType))
+         colnames(design) <- c('100:0', '95:5-100:0')
+         fit <- lmFit(selDataMatrix, design)
+         fit <- eBayes(fit)
+       ## Add gene symbols to gene properties
+        if (require(lumiHumanV1) & require(annotate)) {
+            geneSymbol <- getSYMBOL(fit$genes$ID, 'lumiHumanV1')
+             fit$genes <- data.frame(fit$genes, geneSymbol=geneSymbol)
+        }
+         ## print the top 10 genes
+         topTable(fit, coef='95:5-100:0', adjust='fdr', number=10)
+
+         ## get significant gene list with FDR adjusted p.values less than 0.01
+         p.adj <- p.adjust(fit$p.value[,2])
+         sigGene.adj <- selProbe[ p.adj < 0.01]
+         ## without FDR adjustment
+         sigGene <- selProbe[ fit$p.value[,2] < 0.001]
+ }
```

```
                     ID geneSymbol    logFC        t      P.Value    adj.P.Val
3080 EY761AIG0XSLUfnuyc        CGA 5.858622 80.52721 5.010639e-19 1.843862e-15
1116 ol_iQkR.siio.kvH6k      PLAC4 5.384036 78.46605 7.029591e-19 1.843862e-15
3772 WlCoF7taz2MeYf3l6I       SDC1 4.491916 70.12367 3.049078e-18 5.331822e-15
47   NSjRKdq2eSGf0ur4aQ       PRG2 4.353223 66.61294 5.959749e-18 6.678618e-15
2520 QaYYojcJJvVElV3I98       DLK1 4.055541 64.87604 8.412367e-18 6.678618e-15
1401 6QNThLQLd61eU6IXhI       PSG9 4.233081 64.75119 8.626438e-18 6.678618e-15
3831 TueuSaiCheWBxB6B18      KISS1 4.375865 64.58994 8.911614e-18 6.678618e-15
4693 iz6rhffqh2qnreOge4      GDF15 4.598652 63.80724 1.044777e-17 6.851127e-15
1027 uioiKiIlzFXx8k5EC4        CRH 4.109536 62.07365 1.496392e-17 8.722305e-15
3236 Q.oCSr13l5wQlRuhS0      FSTL1 4.047173 60.30258 2.182613e-17 1.078753e-14
            B
3080 33.70487
1116 33.41479
3772 32.12398
47   31.51764
2520 31.20204
1401 31.17893
3831 31.14901
4693 31.00238
```

```
1027 30.66930
3236 30.31667
```

Based on the significant genes identified using *limma* or t-test, we can do further analysis, like GO analysis (*GOstats* package) and machine learning (*MLInterface* package). Next, we will use GO analysis as an example.

8.3 Gene Ontology analysis

Based on the significant genes identified using *limma* or t-test, we can further do Gene Ontology annotation. We can use package *GOstats* to do the analysis.

Do Hypergeometric test of Gene Ontology based on the significant gene list (for e. Table 1 shows the significant GO terms of Molecular Function with p-value less than 0.01. Here only show the significant GO terms of BP (Biological Process). For other GO categories MF(Molecular Function) and CC (Cellular Component), it just follows the same procedure.

```
> if (require(GOstats) & require(lumiHumanV1)) {
+
+       ## Get the locuslink Id of the gene
+       sigLL <- unique(unlist(mget(sigGene, env=lumiHumanV1ENTREZID, ifnotfound=NA)))
+       sigLL <- as.character(sigLL[!is.na(sigLL)])
+       params <- new("GOHyperGParams",
+               geneIds= sigLL,
+               annotation="lumiHumanV1",
+               ontology="BP",
+               pvalueCutoff= 0.01,
+               conditional=FALSE,
+               testDirection="over")
+
+       hgOver <- hyperGTest(params)
+
+       ## Get the p-values of the test
+       gGhyp.pv <- pvalues(hgOver)
+
+       ## select the Go terms with p-value less than 0.001
+       sigGO.ID <- names(gGhyp.pv[gGhyp.pv < 0.001])
+
+       ## Here only show the significant GO terms of BP (Molecular Function)
+       ##         For other categories, just follow the same procedure.
+       sigGO.Term <- getGOTerm(sigGO.ID)[["BP"]]
+ }
```

9 Session Info

```
> toLatex(sessionInfo())
```

- R version 2.6.0 (2007-10-03), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=en_US;LC_NUMERIC=C;LC_TIME=en_US;LC_COLLATE=en_US;LC_MONETARY=en_US;LC`

	GO ID	Term	p-value	Significant Genes No.	Total Genes No.
1	GO:0009611	response to wound...	8.4244e-06	42	443
2	GO:0006955	immune response	8.8296e-06	68	859
3	GO:0006952	defense response	1.7525e-05	72	945
4	GO:0006950	response to stres...	1.9132e-05	81	1103
5	GO:0009607	response to bioti...	5.0811e-05	72	976
6	GO:0009613	response to pest,...	7.2813e-05	45	533
7	GO:0006954	inflammatory resp...	0.00025402	25	250
8	GO:0009605	response to exter...	0.00026005	46	580
9	GO:0051707	response to other...	0.00040553	45	575
10	GO:0051674	localization of c...	0.00082563	30	348
11	GO:0006928	cell motility	0.00082563	30	348
12	GO:0040011	locomotion	0.00099205	30	352

Table 1: GO terms, p-values and counts.

- Base packages: base, datasets, graphics, grDevices, methods, stats, tools, utils
- Other packages: affy 1.16.0, affyio 1.6.0, annotate 1.16.0, AnnotationDbi 1.0.0, Biobase 1.16.0, DBI 0.2-3, GO 1.99.1, limma 2.12.0, lumi 1.4.0, lumiHumanV1 1.3.1, mgcv 1.3-27, preprocessCore 1.0.0, RSQLite 0.6-3, xtable 1.5-1

10 Reference

1. Lin, S.M., Du, P., Kibbe, W.A., ”Model-based Variance-stabilizing Transformation for Illumina Mi-croarray Data”, under review

2. Du, P., Kibbe, W.A. and Lin, S.M., ”nuID: A universal naming schema of oligonucleotides for Illumina, Affymetrix, and other microarrays”, Biology Direct 2007, 2:16 (31May2007).

3. Barnes, M., Freudenberg, J., Thompson, S., Aronow, B. and Pav-lidis, P. (2005) ”Experimental comparison and cross-validation of the Affymetrix and Illumina gene expression analysis platforms”, Nucleic Acids Res, 33, 5914-5923.