Description of the maDB package

Johannes Rainer*

April 20, 2005

Tyrolean Cancer Research Institute
Innrain 66, 6020 Innsbruck, Austria, `http://www.tcri.at`
and Institute for Genomics and Bioinformatics, Graz University of Technology,
8010 Graz, Austria, `http://www.genome.tugraz.at`

Contents

1 Introduction

The package *maDB* provides functions to create a micro array database and to store micro array experiments (normalized expression values) along with sample information and annotation into it. The micro array database should be flexible enough to store one channel arrays like Affymetrix Genechips as well as two color arrays into it. Micro array experiments can be reloaded, once stored into the database, into R as whole experiment, or only a subset of specific genes and / or samples can be retrieved. As an additional feature it is possible to search for genes with a specific gene expression or regulation pattern over a set for samples

*johannes.rainer@tugraz.at

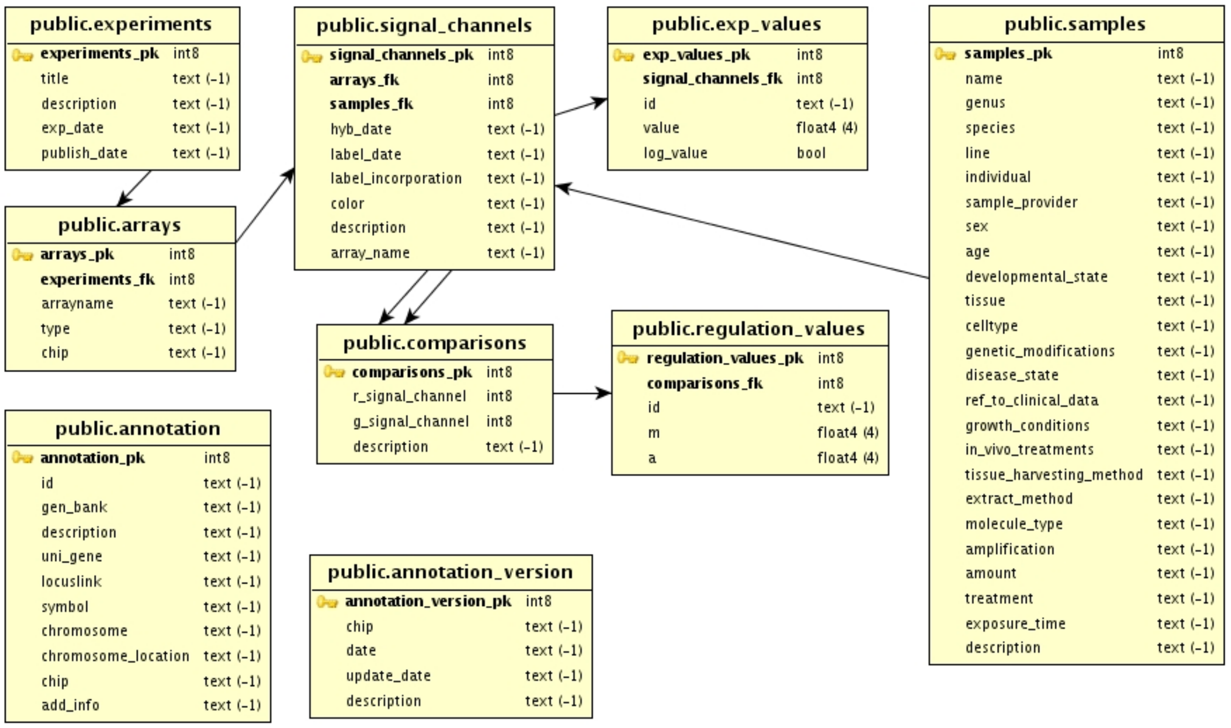

Figure 1: The maDB database model.

(arrays). As expression or regulation pattern template a custom template can be used, as well as the pattern of a gene in the database.

This package is not meant to replace the existing micro array database systems, it should simply allow to store micro array experiments in one place together with sample information and annotation, so that for example different people can work from different workstations with the same datasets.

The database model is shown in figure 1. Attributes with the ending *_pk* and *_fk* mark primary keys and foreign keys respectively. To describe this simple model in a few words: each micro array experiment consists of a set of arrays, where each of the arrays has one (Affymetrix) or more (two or more color arrays) signal channels. In each signal channels there are measured a number of n expression values (one expression value per gene, the `exp_values` table should contain normalized expression values, also for two color arrays). The signal channels link to the samples table (one sample can be hybridized onto more than one signal channels in the case of a technical replicate). Again two signal channels can be combined in one comparison and result in the regulation (M) and average expression (A) values which are stored in the `regulation_values` table. The annotation for the IDs can be stored in a database table called `annotation`.

2 Storing a micro array experiment into the database

First of all a connection to the PostgreSQL database has to be established (using the `dbConnect` function from the *RdbiPgSQL* package) and a empty database has to be created (this can be done or with the `createdb` function in a Linux console, or by the SQL call `CREATE DATABASE dbname`). Before creating the database it is needed to delete any databases with the same name (it is therefore also important to use another database name). To avoid the logging of every SQL call the `log.level` is set to `ERROR`, so only possible errors are logged to the log file.

```
> library(maDB)

Loading required package: Biobase
Loading required package: tools
Welcome to Bioconductor
         Vignettes contain introductory material.  To view,
         simply type: openVignette()
         For details on reading vignettes, see
         the openVignette help page.
Loading required package: affy
Loading required package: reposTools
Loading required package: limma
Loading required package: pgUtils
Loading required package: Rdbi
Loading required package: RdbiPgSQL

> con.su <- dbConnect(PgSQL(), host = "localhost", user = "postgres",
+     dbname = "template1")
> result <- dbGetResult(dbSendQuery(con.su, "SELECT datname FROM pg_database"))
> if (sum(result[, "datname"] == "madb") > 0) {
+     dbSendQuery(con.su, "DROP DATABASE madb")
+ }

status = 1
status.string = PGRES_COMMAND_OK
rows = 0
columns = 0
is.binary = FALSE
command.response = DROP DATABASE

> dbSendQuery(con.su, "CREATE DATABASE madb")

status = 1
status.string = PGRES_COMMAND_OK
```

```
rows = 0
columns = 0
is.binary = FALSE
command.response = CREATE DATABASE
```

```
> dbDisconnect(con.su)
> con <- dbConnect(PgSQL(), host = "localhost", user = "postgres",
+     dbname = "madb")
> log.level <- "ERROR"
```

In the *real life* the connections should not be established as user *postgres*, and the PostgreSQL database should be configured in the way, that at least a password has to be submitted.

The micro array database tables will be created automatically upon the first insertion of a micro array experiments into the database. Lets assume that you got a object from the type `exprSet` from the *Biobase* package after normalization (eg using `rma` from the *affy* package). In this vignette we use a subset of genes (100 probe sets) from a micro array study using Affymetrix GeneChips. Basically the samples are from two B–ALL (B–cell acute lymphoblastic leukemia) patients and two T–ALL patients. From each patient peripheral blood was taken before start of treatment, and after 6 hours GC (glucocorticoid) treatment. Those samples (sample before treatment and after 6 hours GC treatment) are hybridized onto the chips used in this example.

The `exprSet` object is transformed into a `EexprSet` object which inherits all attributes from it and which extends its functionality (for example to draw MA plots using `drawMA` or to store the object into the database).

```
> data("smallALL")
> class(NormChips)

[1] "exprSet"
attr(,"package")
[1] "Biobase"

> NormChips <- EexprSet(NormChips)
> class(NormChips)

[1] "EexprSet"
attr(,"package")
[1] "maDB"
```

Before storing the experiment into the database it is useful to describe the samples that were hybridized onto the chips. This information can be stored into a list of `Samples` objects that can be added to the `@samples` slot of the `EexprSet` object. The linkage between the signal channel and the samples has to be established with `SignalChannel` objects.

```
> Sample1 <- new("Sample", name = "B-ALL-13", individual = "13KKI",
+     tissue = "PB", species = "hs", exposure.time = "0h")
> Sample2 <- new("Sample", name = "B-ALL-13", individual = "13KKI",
+     in.vivo.treatments = "GC", tissue = "PB", species = "hs",
+     exposure.time = "6h")
> Sample3 <- new("Sample", name = "B-ALL-17", individual = "17KKI",
+     tissue = "PB", species = "hs", exposure.time = "0h")
> Sample4 <- new("Sample", name = "B-ALL-17", individual = "13KKI",
+     in.vivo.treatments = "GC", tissue = "PB", species = "hs",
+     exposure.time = "6h")
> Sample5 <- new("Sample", name = "T-ALL-20", individual = "20KKI",
+     tissue = "PB", species = "hs", exposure.time = "0h")
> Sample6 <- new("Sample", name = "T-ALL-20", individual = "20KKI",
+     in.vivo.treatments = "GC", tissue = "PB", species = "hs",
+     exposure.time = "6h")
> Sample7 <- new("Sample", name = "T-ALL-25", individual = "25KKI",
+     tissue = "PB", species = "hs", exposure.time = "0h")
> Sample8 <- new("Sample", name = "T-ALL-25", individual = "25KKI",
+     in.vivo.treatments = "GC", tissue = "PB", species = "hs",
+     exposure.time = "6h")
> TheSamples <- list(Sample1, Sample2, Sample3, Sample4, Sample5,
+     Sample6, Sample7, Sample8)
> SigChannels <- getSignalChannels(NormChips)

Got one color micro arrays...

> for (i in 1:length(SigChannels)) {
+     SigChannels[[i]]@sample.index <- i
+ }
> publishToDB(NormChips, con, exp.name = "maDB_Vignette", signal.channels = SigChannel
+     samples = TheSamples, v = FALSE)

Setting the data types for all attributes to   TEXT

Setting the data types for all attributes to   TEXT

Setting the data types for all attributes to   TEXT

Setting the data types for all attributes to   TEXT

Inserting the sample into the database
Inserting the sample into the database
Inserting the sample into the database
Inserting the sample into the database
Inserting the sample into the database
Inserting the sample into the database
Inserting the sample into the database
Inserting the sample into the database
Experiment added to the experiments table...
```

If it is desired to compare the expression values between the chips, it is possible to calculate regulation values (M values). The function `dbCalculateRegulations` calculates those regulation values within a micro array experiment and inserts the regulation values back into the database (the regulation values (M) and average expression values (A) into the `regulation_values` database table, the information about the signal channels used, in the comparisons into the `comparisons` table).

The function `dbGetExperimentInfo` can be used to get a list of available experiments that are stored into the database, or to read a short information from a specific experiment out of the database.

```
> dbGetExperimentInfo(con)
```

```
You have not submitted an experiment title, please submit one of the following that are
  experiments_pk         title description exp_date             publish_date
1              1 maDB_Vignette            18011977 Wed Apr 20 15:43:44 2005
```

```
> info <- dbGetExperimentInfo(con, "maDB_Vignette")
> colnames(info)
```

```
 [1] "arrays_pk"          "chip"               "arrays_fk"
 [4] "signal_channels_pk" "array_name"         "samples_fk"
 [7] "name"               "individual"         "line"
[10] "tissue"             "treatment"          "in_vivo_treatments"
[13] "exposure_time"
```

Regulation values will be calculated by comparing the expression values from the 100 genes of the chips after 6 hours GC–treatment with those before treatment for each patient. These regulation values are stored into the `regulation_values` database tables.

```
> info[, c("name", "in_vivo_treatments", "exposure_time")]
```

```
      name in_vivo_treatments exposure_time
1 B-ALL-13               <NA>            0h
2 B-ALL-13                 GC            6h
3 B-ALL-17               <NA>            0h
4 B-ALL-17                 GC            6h
5 T-ALL-20               <NA>            0h
6 T-ALL-20                 GC            6h
7 T-ALL-25               <NA>            0h
8 T-ALL-25                 GC            6h
```

```
> dbCalculateRegulations(con, "maDB_Vignette", comparisons = list(c(2,
+     1), c(4, 3), c(6, 5), c(8, 7)), v = FALSE)
```

To get an information of the comparisons that are available for a micro array experiment, the `dbGetComparisons` can be used.

```
> Comparisons <- dbGetComparisons(con, "maDB_Vignette")
> Comparisons[, c("name.red", "exposure_time.red", "name.green",
+     "exposure_time.green")]

     name.red   exposure_time.red name.green exposure_time.green
[1,] "B-ALL-13" "6h"              "B-ALL-13" "0h"
[2,] "B-ALL-17" "6h"              "B-ALL-17" "0h"
[3,] "T-ALL-20" "6h"              "T-ALL-20" "0h"
[4,] "T-ALL-25" "6h"              "T-ALL-25" "0h"
```

3 Reloading a set of arrays from the database

A whole micro array experiment or only a subset of arrays from an micro array experiment can be retrieved from the database using the `loadFromDB` function. In the example below only those arrays are loaded from the database, on which samples from the patient T–ALL–25 were hybridized onto.

```
> info <- dbGetExperimentInfo(con, "maDB_Vignette")
> info.reduced <- info[info[, "name"] == "T-ALL-25", ]
> TALL25 <- loadFromDB(EexprSet(), con, "maDB_Vignette", pk = info.reduced[,
+     "signal_channels_pk"], v = FALSE)
> TALL25

Expression Set (exprSet) with 
	100 genes
	2 samples
		 phenoData object with 0 variables and 0 cases
	 varLabels
```

The function `drawMA` can be used to draw MA plots from objects of the type `EexprSet`. The parameters *r* and *g* represent the index of the signal channel in the `exprs` slot that should be used as *red* and *green* signal channel. The figure 2 represents the MA plot of the 100 probe sets of the 6 hours GC treated sample of patient T–ALL–25 against the sample before treatment.

```
> drawMA(TALL25, r = 2, g = 1)
```

```
taking the chip 0007_001_25KKI0Pb_h01_SJ_220704.CEL as the green color array and the chi
building a dummy weights matrix...
```

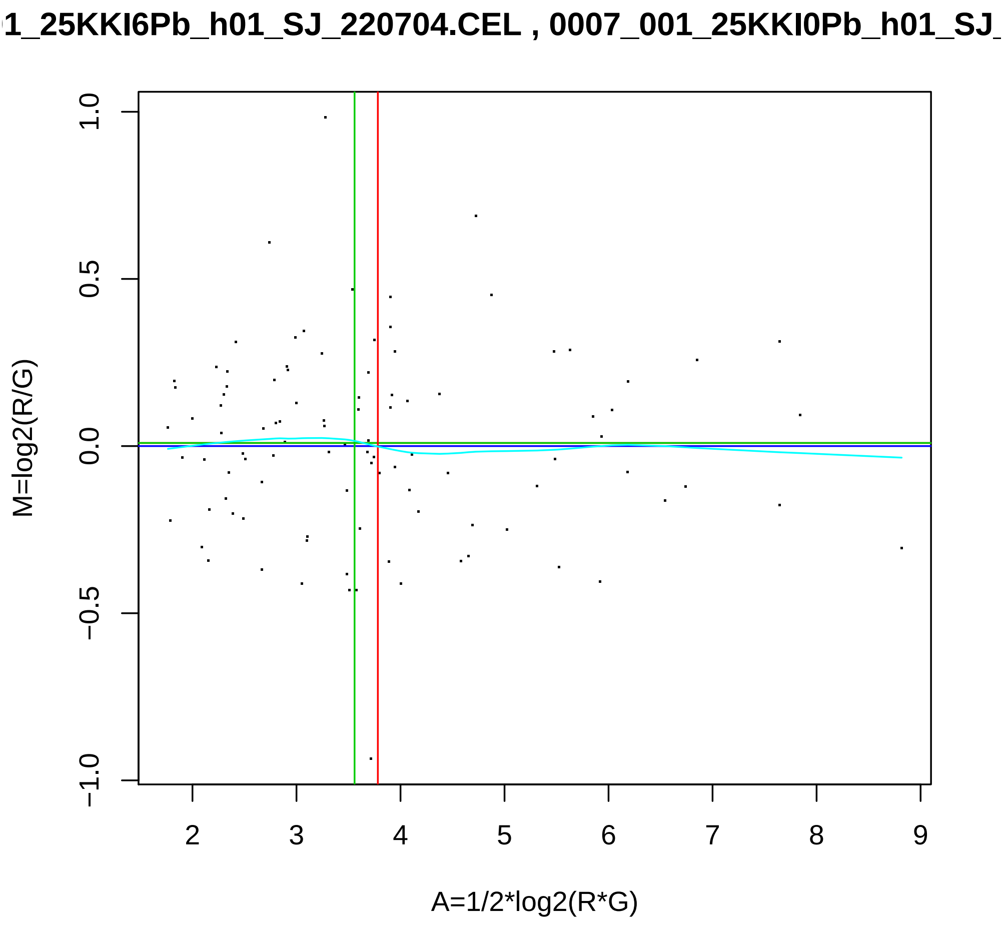

Figure 2: MA plot of patient T–ALL-25 samples.

3.1 Loading expression values for a set of genes from the database

The function `getEDB` can be used to fetch the expression values from a set of genes (in a set of samples) from the database. The code below loads the expression values for the first 5 probe sets in the samples of patient B–ALL–13 from the database. Basically it is needed to first identify the signal channels on which the desired samples were hybridized and to submit the appropriate primary keys of the `signal_channels` database table to the function.

```
> info <- dbGetExperimentInfo(con, "maDB_Vignette")
> info.reduced <- info[info[, "name"] == "B-ALL-13", ]
> EValues <- getEDB(con, id = rownames(exprs(NormChips))[1:5],
+     signal.channels.pk = info.reduced[, "signal_channels_pk"],
+     v = FALSE, column.names = c("name", "exposure_time"))
> EValues

           B-ALL-13, 0h B-ALL-13, 6h
1553914_at      2.87426      3.16978
1553915_at      1.81656      1.95677
1553917_at      3.36801      3.05621
1553918_at      3.36252      3.16360
1553919_at      3.73338      3.71397
```

3.2 Loading regulation values for a set of genes from the database

The function to load regulation values for a set of genes from a set of comparisons from the database is called `getMDB` and works just like the `getEDB` function.

```
> info <- dbGetComparisons(con)
> info.reduced <- info[info[, "name.red"] == "B-ALL-13" | info[,
+     "name.red"] == "B-ALL-17", ]
> MValues <- getMDB(con, id = rownames(exprs(NormChips))[1:5],
+     comparisons.pk = info.reduced[, "comparisons_pk"], v = FALSE,
+     column.names = c("name", "exposure_time"))
> MValues

           UniqueID     B-ALL-13, 6h B-ALL-17, 6h
1553914_at "1553914_at" "0.29552"    "0.49591"
1553915_at "1553915_at" "0.14021"    "0.18615"
1553917_at "1553917_at" "-0.3118"    "-0.12686"
1553918_at "1553918_at" "-0.19892"   "0.16644"
1553919_at "1553919_at" "-0.01941"   "-0.10506"
```

4 Inserting the annotation into the database

To insert the annotation of the IDs that identify the sequences on the arrays the function `dbUpdateAnnotation` can be used. Basically this function can also be used to update the annotation that is already stored in the database (for example after a new UniGene release). In the case of updating the annotation the old annotation from the database will automatically backed up into a file. To check how much the annotation in the database differs from a new one (for example of a new annotation package release) the function `checkForDifferentAnnotation` can be used.

The annotation created in this example bases on the annotation of the *hgu133plus2* meta data package and the annotation table is created with the functions of the *annaffy* package.

```
> IDs <- dbGetResult(dbSendQuery(con, "SELECT DISTINCT id FROM exp_values"))[,
+     "id"]
```

The code above gets a unique list of IDs from the `exp_values` database table (as IDs the rownames of the `@exprs` slot of the `exprSet` object are used).

```
> library(annaffy)

Loading required package: GO
Loading required package: KEGG
Loading required package: annotate

> AnnotationTable <- matrix(ncol = 4, nrow = length(IDs))
> colnames(AnnotationTable) <- c("id", "gen_bank", "uni_gene",
+     "symbol")
> AnnotationTable[, 1] <- IDs
> AnnotationTable[, 2] <- getText(aafGenBank(IDs, "hgu133plus2"))

[1] "You have package hgu133plus2 but the incorrect version"
Loading required package: hgu133plus2

> AnnotationTable[, 3] <- getText(aafUniGene(IDs, "hgu133plus2"))
> AnnotationTable[, 4] <- getText(aafSymbol(IDs, "hgu133plus2"))
```

This (short) annotation table can be inserted using the `dbUpdateAnnotation` function. Allowed columns (and column names) are *id, gen_bank, description, uni_gene, locuslink, symbol, chromosome, chromosome_location.* Future versions of the *maDB* package will be more flexible concerning the annotation of the IDs.

For the later annotation of the IDs using the database the `dbGetAnnotation` or `getAnnotation` functions can be used.

```
> dbUpdateAnnotation(con, data = AnnotationTable, chip = NormChips@annotation,
+     v = FALSE)
```

```
Setting the data types for all attributes to   TEXT

Setting the data types for all attributes to   TEXT

> dbGetAnnotation(con, id = IDs[1:5], v = FALSE, chip = "hgu133plus2")

     id           gen_bank    description uni_gene    locuslink symbol
[1,] "1553914_at" "NM_153227" NA          "Hs.447547" NA        "MGC34800"
[2,] "1553915_at" "NM_173577" NA          ""          NA        "C10orf126"
[3,] "1553917_at" "NM_152671" NA          "Hs.173939" NA        "PIP5K3"
[4,] "1553918_at" "NM_152506" NA          "Hs.350679" NA        "C21orf129"
[5,] "1553919_at" "NM_173520" NA          "Hs.380224" NA        "C9orf62"
     chromosome
[1,] NA
[2,] NA
[3,] NA
[4,] NA
[5,] NA
```

It is also possible to search for all IDs (probe sets) that target the same gene.

```
> dbGetAnnotation(con, id = "CD24", chip = "hgu133plus2", search.col = "symbol",
+     v = FALSE)

  id            gen_bank description uni_gene    locuslink symbol chromosome
1 "208651_x_at" "M58664" NA          "Hs.375108" NA        "CD24" NA
2 "266_s_at"    "L33930" NA          "Hs.375108" NA        "CD24" NA
```

5 Searching for genes with similar expression or regulation pattern

Clustering genes according to their regulation or expression pattern in a set of samples is nothing new in the micro array analysis. The function `dbSearchSimilarPattern` performs basically the same procedure, that is to compare the expression or regulation pattern of a gene with those of other genes. The similarity is measured (like in the HCL (hierarchical clustering) analysis) using different distance measurement methods (euclidian, pearson correlation ...). But instead of clustering the genes the function simply measures the distances (similarity) of a template gene expression or regulation pattern and those of the genes in the database. As a template a gene pattern or a custom template can be used.

In the example below genes that show a similar expression pattern like the CD19 gene (which is a B–lymphocyte antigen) in the 8 samples stored in the database are searched. With the parameter `include.by.name` the samples can be specified, that should be included in the similarity calculation (in our case all 4 patients (and therefore all 8 samples)). By default the function uses euclidian distance measurement to define the similarity between vectors, but also other methods can be used. The `Similarity` which is returned by the function can also be used to plot the similar gene expression patterns (see figure 3).

```
> CD19Sim <- dbSearchSimilarPattern(con, id = "206398_s_at", include.by.name = c("T-AL
+     "T-ALL-25", "B-ALL-13", "B-ALL-17"), include.values = TRUE,
+     v = FALSE)
> dbGetAnnotation(con, id = names(CD19Sim@distances)[1:5], chip = "hgu133plus2",
+     columns = "symbol", v = FALSE)

     id             symbol
[1,] "206398_s_at"  "CD19"
[2,] "208651_x_at"  "CD24"
[3,] "266_s_at"     "CD24"
[4,] "1553993_s_at" "MED25"
[5,] "1553971_a_at" "GATS"

> dbDisconnect(con)
> con <- NULL
> plot(CD19Sim)
```

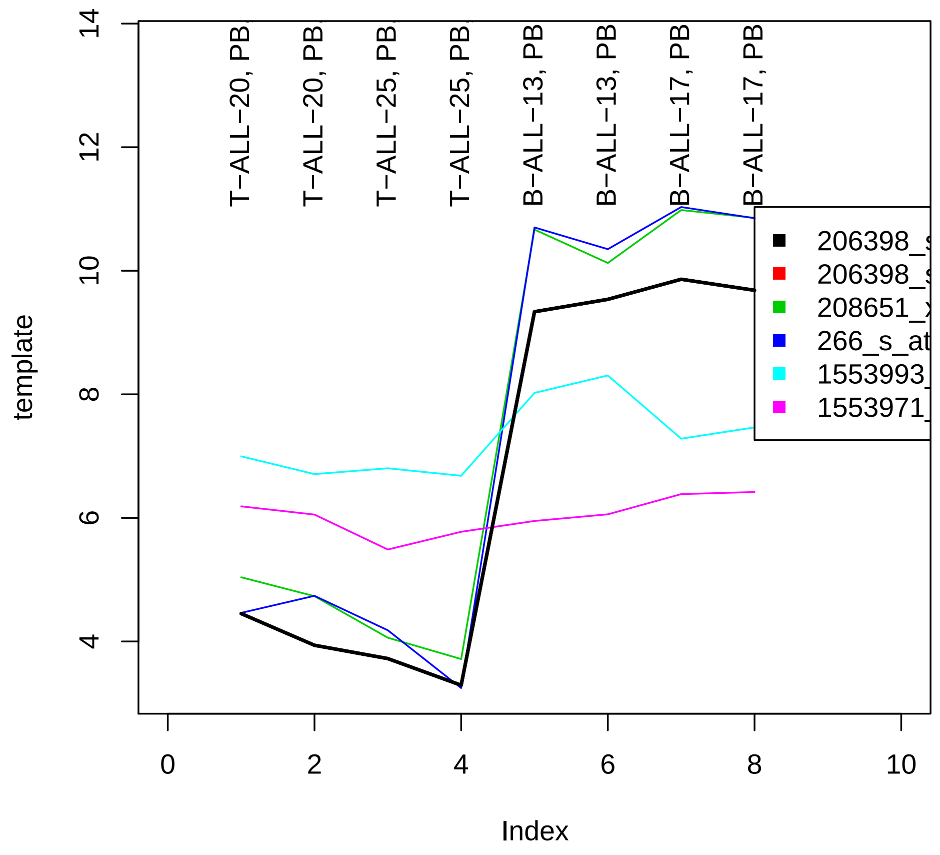

Figure 3: Genes wit similar expression pattern as CD19. (black line: expression pattern of the probe set 206398_s_at (CD19), the other lines correspond to the 5 genes with the most similar expression pattern as the CD19 gene).