Model Based Analysis of Tiling Arrays
The rMAT package.

Charles Cheung*and Raphael Gottardo†and Arnaud Droit‡

July 6, 2010

A step-by-step guide in the analysis of tiling array data using the rMAT package in R

Contents

*cykc@interchange.ubc.ca
†raphael.gottardo@ircm.qc.ca
‡arnaud.droit@ircm.qc.ca

Part I
Licensing

Under the Artistic license 2.0, you are free to use and redistribute this software. However, we ask you to cite the following paper if you use this software for publication.

> A. Droit, C. Cheung, and R. Gottardo, "RMATâĂŞan R/Bioconductor package for analyzing ChIP-chip experiments." Bioinformatics (Oxford, England), vol. 26, iss. 5, pp. 678-679, 2010.

Part II
Introduction

In our guide, we include examples of code that we hope will help you when using the rMAT package. The examples are kept at the basic level for ease of understanding. Some of the options in the functions have been set by default. To learn more about the exact parameters and usage of each function, you may type `help(FUNCTION_NAME)` of the function of interest in R after the rMAT package is loaded.

The probe sequence information of an Affymetrix tiling array is stored in the .BPMAP file, while the corresponding expression values (intensity signals) of each experiment is stored separately in each .CEL file. The BPMAP file contains different sequences that describe different contents in the array. For instance, the first sequence may contain probes from chromosome 1 while the second sequence may contain probes from chromosome 2. Each probe would include information such as its Perfect Match base pair sequence (ie. AGCTTCGAAGCTTCGAAGCTTCGAG), location on chromosome, X and Y coordinates, etc. The CEL file does not carry any information about the design of the array; but simply X and Y coordinates and expression values, as well as other auxiliary columns. For each array experiment (ie. mock, treated with reagent X, treated with reagent Y), we have one CEL file. BPMAP and CEL files are stored in binary format and require a parser (reader) to read its content meaningfully. We make use of the `affxparser` software for this purpose, although we provide convenient wrappers for most of the necessary functions.

The common goal in analyzing ChIP-chip data, and more generally tiling array data, is to find activities (DNA-protein interaction, transcription, etc) in specific chromosomal regions. This package focus on detecting DNA-protein interactions from ChIP-chip experiments. Though, many of the functions are more general than that, e.g. parsing/normalization. Major analysis steps are described below.

Part III
Loading the rMAT Package

To load the rMAT package in R, we type

```
> library(rMAT)
```

Part IV
Loading in the data

The next step in a typical analysis is to load-in/read data from Affymetrix CEL files. The data used in this example are available in this package in inst/doc folder.
In this documentation, the path for the data is the /rMAT/inst/doc folder.

Part V
Reading BPMAP and CEL files

Reading the design of tiling array

In order to read-in appropriate data values from the CEL/BPMAP files, we first need to understand their content. To understand the design, we would explore the header section of the BPMAP file using the function ReadBPMAPAllSeqHeader. ReadBPMAPAllSequence takes in the filename of the BPMAP file as an argument. The filename is formatted as a string literal (characters) in unix path format and stored in the variable BPMAP-File, which is then used by **ReadBPMAPAllSeqHeader** to specify which BPMAP file to read.

```
> pwd <- ""
> path <- system.file("doc/Sc03b_MR_v04_10000.bpmap", package = "rMAT")
> bpmapFile = paste(pwd, path, sep = "")
> seqHeader <- ReadBPMAPAllSeqHeader(bpmapFile)
> print(seqHeader)

$SeqName
[1] "chr1"

$GroupName
[1] "Sc"

$version
[1] "Oct_2003"

$probeMapping
[1] 0

$seqNum
[1] 0

$NumHits
[1] 10000
```

Specifying the filenames

From the above header content, the information we want to obtain is the direct mapping from sequence number to chromosome number. Sequence numbers are stored in the seqNum column while chromosome numbers can be read from the Name column, which describes the content of the `sequence`. We would like to read the BPMAP and CEL files and merge them by X and Y coordinate so information such as probe sequence and location along the chromosome would pair up with the corresponding expression value.
We have already specified the location of the BPMAP file in BPMAP-File variable, so now let's specify the location of the CEL files. Because BPMAPCelParser allows us to parse multiple CEL files simultaneously, we can store the location of multiple files in a vector using `c()` each separate by `","` .

```
> pathCEL <- system.file("doc/Swr1WTIP_Short.CEL", package = "rMAT")
> arrayFile <- paste(pwd, c(pathCEL), sep = "")
```

Calling BPMAPCelParser

We are now ready to call and use the BPMAPCelParser.

```
> ScSet <- BPMAPCelParser(bpmapFile, arrayFile, verbose = FALSE,
+     groupName = "Sc")
```

the 'groupeName' argument corresponding to the genome name used. In this example, we specified saccharomyces cerevisiae genome (Sc). If no groupName is specified, all sequences will be read, including Affymetrix controls, etc. You probably don't want that.

This function returns an object of class tilingSet containing all necessary information: probe sequences, genomic positions, chromosomes as well as the probe intensities.

The list of vectors of the merged data are now stored in ScSet. Let's explore the (partial) content of ScSet.

```
> summary(ScSet)
```

```
Genome interrogated:  Sc03b_MR_v04_10000
Chromosome(s) interrogated: chr1
Sample name(s):  Swr1WTIP_Short
The total number of probes is:  10000
Preprocessing Information
  - Transformation: log
  - Normalization: none
```

We are now ready to normalize the raw data. Normalization is a procedure to transform raw data into the so-called normalized expression data so expression values from different tiling arrays become more comparable.

Part VI
Normalization

The 'NormalizeProbes' function allows users to normalize expression values of different experiments with one command, as long as all those experiments use the same BPMAP tiling design file. We can load these raw expression values in batch using `cbind()`. NormalizeProbes also requires users to specify the sequence vector. In this case, it is a vector of characters containing

the 25 base pair sequence of each probe. (Right now, Normalization works for reading 25mer only.)

For a complete list of parameters for NormalizeProbes, please refer to `help(NormalizeProbes)`. We are now ready to run the command.

```
> ScSetNorm <- NormalizeProbes(ScSet, method = "MAT", robust = FALSE,
+     all = FALSE, standard = TRUE, verbose = FALSE)
```

The user can choose from 'MAT', or 'PairBinned' normalization method. The 'PairBinned' option takes into account interaction between adjacent pairs along the probe as covariates for the linear regression. Both model require the same number of parameters. For more details on the other options, please refer to the man pages.

The output in this example is saved in `ScSetNorm`.

Let's explore the (partial) content of ScSetNorm.

```
> summary(ScSetNorm)

   Genome interrogated:  Sc03b_MR_v04_10000
   Chromosome(s) interrogated: chr1
   Sample name(s):  Swr1WTIP_Short
   The total number of probes is:  10000
   Preprocessing Information
     - Transformation: log
     - Normalization: MAT standardized
```

Part VII
Finding the Enriched Regions

After normalization, we are ready to compute the MatScores and identify enriched regions. There are various ways to call enriched regions, based on p-values, FDR threshold and MATscore thresholds. On the command below, we use the a MATScore threshold of 1.

For a comprehensive list of parameters you can adjust in callEnrichedRegions, please refer to `help(MATScore)`. Another note is that if FDR is used, threshold should be set in the range between 0 and 1.

The 'computeMATScore' function is first used to compute the scores and return a 'RangedData' object, which can then use exported to a 'wig' file and/or uploaded to a genome browser using `rtracklayer`.

```
> RD <- computeMATScore(ScSetNorm, cName = NULL, dMax = 600, verbose = TRUE)

** Finished processing  10000  probes on  1  arrays **

> Enrich <- callEnrichedRegions(RD, dMax = 600, dMerge = 300, nProbesMin = 8,
+     method = "score", threshold = 1, verbose = FALSE)
```

1 Creating an annotation graphic

rMAT results can benefit from integrated visualisation of the genomic information. We have decided to use the `rtrackalayer` or `GenomeGraphs` package. This last package uses the `biomaRt` package to deliver queries to Ensembl e.g. gene/transcript structures to viewports of the grid package, resulting in genomic information plotted together with your data.

To load the GenomeGraphs and rtracklayer packages in R, we type

```
> library(GenomeGraphs)
> library(rtracklayer)
```

2 Plotting a Gene with rtracklayer

```
> genome(Enrich) <- "sacCer2"
> names(Enrich) <- "chrI"
> session <- browserSession("UCSC")
> track(session, "toto") <- Enrich
> subEnrich <- Enrich[2, ]
> view <- browserView(session, range(subEnrich) * -2)
```

3 Plotting a Gene with GenomeGraphs

If one wants to plot annotation information from Ensembl then you need to connect to Ensembl Biomart using the useMart function of the `biomaRt` package.

```
> mart <- useMart("ensembl", dataset = "scerevisiae_gene_ensembl")
```

If you are interested in plotting a whole gene region, you should create a GeneRegion object. In the example below we will retrieve the genes of the chromsome (I) between 1 and 200000. We added a genomic axis as well to give us the base positions.

```
> genomeAxis <- makeGenomeAxis(add53 = TRUE, add35 = TRUE)
> minbase <- 1
> maxbase <- 50000

> genesplus <- makeGeneRegion(start = minbase, end = maxbase, strand = "+",
+     chromosome = "I", biomart = mart)
> genesmin <- makeGeneRegion(start = minbase, end = maxbase, strand = "-",
+     chromosome = "I", biomart = mart)
```

We create a Generic Array for chromosome I only

```
> RD1 <- RD[space(RD) == "chr1", ]
> Enrich1 <- Enrich[space(Enrich) == "chr1", ]
> MatScore <- makeGenericArray(intensity = as.matrix(score(RD1)),
+     probeStart = start(RD1), dp = DisplayPars(size = 1, color = "black",
+         type = "l"))
> rectList <- makeRectangleOverlay(start = start(Enrich1), end = end(Enrich1),
+     region = c(1, 4), dp = DisplayPars(color = "green", alpha = 0.1))

> gdPlot(list(score = MatScore, `Gene +` = genesplus, Position = genomeAxis,
+     `Gene -` = genesmin), minBase = minbase, maxBase = maxbase,
+     labelCex = 1, overlays = rectList)
```

Part VIII
Appendix: Installing rMAT

To build the `rMAT` package from source, make sure that the following is present in your system:

- GNU Scientific Library (GSL)
- Basic Linear Algebra Subprograms (BLAS)

GSL can be downloaded at `http://www.gnu.org/software/gsl/`. In addition, the package uses BLAS to perform basic vector and matrix operations. Please go to `http://www.netlib.org/blas/faq.html#5` for a list of optimized BLAS libraries for a variety of computer architectures. For instance, Mac users may use the built-in vecLib framework, while users of Intel machines may use the Math Kernel Library (MKL). A C compiler is needed to build the package as the core of the `rMAT` function is coded in C.

For the package to be installed properly you might have to type the following command before installation:

```
export LD_LIBRARY_PATH='/path/to/GSL/:/path/to/BLAS/':$LD_LIBRARY_PATH
```

which will tell **R** where your GSL and BLAS libraries (see below for more details about BLAS libraries) are. Note that this might have already been configured on your system, so you might not have to do so. In case you need to do it, you might consider copying and pasting the line in your `.bashrc` so that you do not have to do it every time.

Now you are ready to install the package:

```
R CMD INSTALL rMAT_x.y.z.tar.gz
```

The package will look for a BLAS library on your system, and by default it will choose gslcblas, which is not optimized for your system. To use an optimized BLAS library, you can use the `--with-blas` argument which will be passed to the `configure.ac` file. For example, on a Mac with vecLib pre-installed the package may be installed via:

```
R CMD INSTALL rMAT_x.y.z.tar.gz --configure-args="--with-blas='-
framework vecLib'"
```

On a 64-bit Intel machine which has MKL as the optimized BLAS library, the command may look like:

```
R CMD INSTALL rMAT_x.y.z.tar.gz --configure-args="--with-blas='-
L/usr/local/mkl/lib/em64t/ -lmkl -lguide -lpthread'"
```

where `/usr/local/mkl/lib/em64t/` is the path to MKL.

If you prefer to install a prebuilt binary, you need GSL for successful installation. Finally, as of version 2.1.0, rMAT makes use of the Grand Central Dispatch to normalize arrays in parallel. The Grand Central Dispatch technology is available on Apple Snow Leopard operating system.