Description of exonmap: simple analysis and annotation tools for Affymetrix exon arrays

MichałJ Okoniewski, Tim Yates, Crispin J Miller

June 11, 2007

Contents

1 Introduction

The package *exonmap* is intended to support various forms of data analysis for Affymetrix Exon microarrays. It includes a variety of routines for translating between probesets, exons, genes and transcripts, and makes use of a relational database (X:MAP) to define these relationships for the current genome assembly. X:MAP is built using Ensembl and Affymetrix annotation data, along with custom probeset to genome mappings.

Genome mappings were generated by searching probe sequences against the entire human (or mouse) genome and building database tables representing their hit locations and hit specificity. These are placed alongside data describing exon, transcript and gene

relationships. Most of this is hidden from the user; the package uses a series of functions (e.g. `probeset.to.exon`) that manage the underlying database queries.

The package provides graphics routines for plotting individual genes, and for colouring them by expression level or fold-change, and functions are also provided to link to the X:MAP web-based front end, at `http://xmap.picr.man.ac.uk`.

2 Initial processing of exon array data

Exonmap makes use of the *affy* package; a basic understanding of the library and its vignette is a good idea. We also assume that the reader knows how the Affymetrix system works. If not, a brief introduction can be found at http://bioinf.picr.man.ac.uk/; a more detailed description is in the Affymetrix MAS manual at http://www.affymetrix.com.

Although this package is primarily to support annotation, it does contain some basic utility functions to make it easy to load and begin to explore exon array data. The following section exists simply to provide a quick route to a list of differentially expressed probesets; alternative strategies are of course possible, and you may choose to skip this section and use your own approach.

3 Reading in data and generating expression calls

The first thing you need to do is to get R to use the *exonmap* package by telling it to load the library:

```
> library(exonmap)
> library(affy)
```

R needs to know about the replicates in your experiment, so we must also load some descriptive data that says which arrays were replicates and also something about the different experimental conditions you were testing. This means that *exonmap* needs *two things*:

1. your .CEL files, and
2. a white-space delimited file describing the samples that went on them.

By default, this file is called *covdesc*. The first column should have no header, and contains the names of the .CEL files you want to process. Each remaining column is used to describe something in the experiment you want to study. For example you might have a set of chips produced by treating a cell line with two drugs. Your *covdesc* file might look like something like this:

	treatment
ctrl1.cel	n
ctrl2.cel	n
ctrl3.cel	n
a1.cel	a
a2.cel	a
a3.cel	a
b1.cel	b
b2.cel	b
b3.cel	b
ab1.cel	a.b
ab2.cel	a.b

This is similar to the approach taken by *simpleaffy*.

The easiest way to get going is to:

1. Create a directory, move all the relevant *CEL* files to that directory
2. Create a *covdesc* file and put it in the same directory
3. If using linux/unix, start R in that directory.
4. If using the Rgui for Microsoft Windows make sure your working directory contains the *Cel* files (use "File -> Change Dir" menu item).
5. Load the library.

Exon array CEL files may be read using the function `read.exon`. In all cases an experiment description file (covdesc) must be present.

In addition, a CDF metadata package must be specified. Versions of CDF metadata for mouse and human exon arrays can be downloaded from http://xmap.picr.man.ac.uk. The CDF metadata cannot include control or backround probesets if you are going to process it with RMA or plier.

For example, to get started, you might run something like:

```
> raw.data <- read.exon()
> if (exists(raw.data)) {
+     raw.data@cdfName <- "exon.pmcdf"
+     x.rma <- rma(raw.data)
+ }
```

The CDF files, *exon.pmcdf* for Human Exon 1.0ST array and *mouseexonpmcdf* for Mouse Exon 1.0 ST arrays, available from `http://bioinformatics.picr.man.ac.uk` have been prepared by processing the ASCII CDF files from Affymetrix, using the *(*makecdfenv) and *(*altcdfenvs). They include PM probes only. Probesets representing genomic and antigenomic background and control probesets have also been removed.

4 Pairwise comparison of expression data

The function pc provides fast pairwise comparisons for exprSet objects.

```
> data(exonmap)
> pc.exonmap <- pc(x.rma, "group", c("a", "b"))
```

pc produces an object of class PC that has two slots: *fc*, for the log2 fold change and *tt* containing a t-test p-value. For the purpose of this vignette, we use these to select significant probesets, although other more in-depth approaches are of course possible. For example:

```
> sigs <- names(fc(pc.exonmap))[abs(fc(pc.exonmap)) > 1 & tt(pc.exonmap) <
+     1e-04]
> length(sigs)
```

```
[1] 31
```

5 Translation routines for genes, transcripts, exons and probesets

The X:MAP database can be queried in a number of ways using translation functions. All of them have the form X.to.Y, where X and Y may be a vector of gene, transcript, exon or probeset identifiers. See, for example, ?probeset.to.gene for more details. All the functions produce, by default, a vector of the identifiers resulting from the specified mapping. More information can be generated by setting the parameter *vector.out* to *FALSE*, in which case, a data frame is returned. If *unique* is true, duplicates are removed before the result is returned.

```
> xmapDatabase("Human")
```

Switching to human database... done.

```
> sig.exons <- probeset.to.exon(sigs)
> length(sig.exons)
```

[1] 21

```
> sig.genes <- probeset.to.gene(sigs)
> length(sig.genes)
```

[1] 7 (These numbers are so small because there are only 7 genes represented in the example dataset).

6 More details

`probeset.details`,`exon.details`,`transcript.details` and `gene.details` can all be used to extract detailed annotation, given the appropriate set of identifiers.

7 Probeset filtering

Probesets can be filtered according to the number and quality of their matches to the genome. Match statistics can be displayed with `probeset.stats`.

The hit, exon and gene scores are calculated using all the probes in the probesets (usually 4) by finding the number of their matches to genome, exons and genes - and multiplying the minimum value for the probe within a probeset with a maximum. Thus the first probeset in the example is "intronic" as it matches 1 gene, but no exons. The second one is "exonic" and well defined - it matches exactly 1 exon in 1 gene, and only hits the entire genome exactly once. The third one is a "multitarget" probeset because it includes at least one probe that matches two locations in the genome. The fourth one is "intergenic", because it matches the genome once, but does not hit an annoted gene.

These four types of probesets can be selected or excluded from a probeset list using the `select.probewise` and `exclude.probewise` functions. For example, to find probesets that hit within genes, but outside regions annotated as exons by ensembl:

```
> select.probewise(sigs, filter = "intronic")
```

[1] "3388403"

In a similar way, a probeset list can be filtered to get rid of multiply targeting probesets (i.e. those annotated by X:MAP to hit in more than one place on the genome):

```
> sigs.nomt <- exclude.probewise(sigs, filter = "multitarget")
```

8 Plotting genes of interest

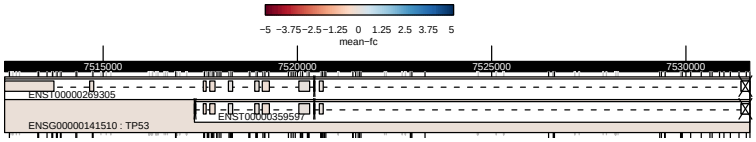
At it's simplest,

```
> plot.gene("ENSG00000141510")
```

will generate a plot of the specified gene (in this case, TP53). Black lines represent well behaved probesets that hit the genome only once, grey lines represent those that hit in more than one place. The outer box represents the gene, each of the inner boxes represents a transcript; each contain exons. Hopefully, the plot is relatively self explanatory.

The plot function can also colour the plot using expression data:

```
> plot.gene("ENSG00000141510", x.rma, 1:3, 4:6)
```



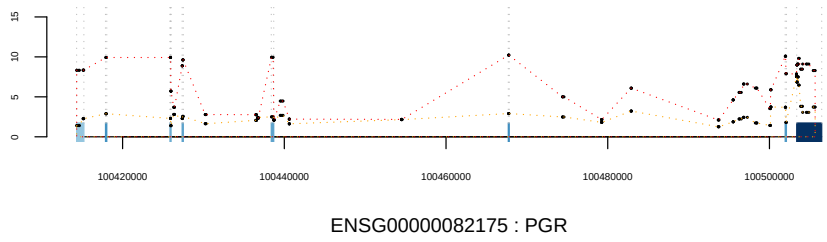
will compute the fold changes between arrays 1:3 and 4:6. All well behaving exon-matching probesets are found, and the mean value used to colour the gene. The process is repeated for each transcript and each exon. By default transcripts aren't coloured.

The approach to averaging can be changed and, raw intensity or t-test p-score values cab plotted instead; see `?plot.gene` for more details. It is also possible to pre-scale the colouring to the fold-change for the gene, so that differential expression for each exon is plotted relative to the gene-average, (using the parameter `scale.to.gene`).

Sometimes, an exon is drawn with a cross through it. This happens when the function can't find a well behaving probeset targeting that exon and represents the fact that there is no data to plot.

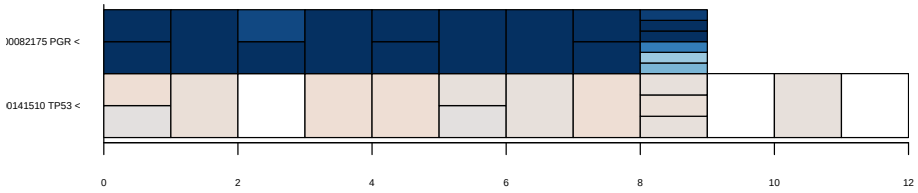
Another utility to visualize the expression of a gene is `?plot.gene.graph`. It creates a line-plot for a specified gene, including intronic probesets. For example:

```
> plot.gene.graph("ENSG00000082175", x.rma, 4:6, 1:3, draw.exon.border = F,
+     scale.to.gene = F, ylim = c(0, 16), type = "median-int")
```



Heatmap style plots can also be generated with the `gene.strip`.

```
> gene.strip(c("ENSG00000141510", "ENSG00000082175"), x.rma, 1:3,
+     4:6, type = "mean-fc")
```



Here, each row corresponds to a gene, and each exon is plotted in exon-order along the X axis. The plot is coloured as before; exons for which a uniquely matching probeset cannot be found are, by default, coloured white. When multiple probesets hit the same exon, these are stacked vertically within that exon's rectangle. The parameter `plot.introns` can be used to change the plotting behaviour so that introns are shown, and exons are positioned relative to their nucleotide position within the gene.

9 Splicing index and splicing ANOVA

Splicing index and splicing ANOVA have also been implemented, as described in the Affymetrix white paper: "Alternative transcript analysis methods for exon arrays".

Splicing index is the ratio of probeset gene expression versus gene-level expression. It is calculated for genes, but produces values assigned to probesets:

```
> si <- splicing.index(x.rma, c("ENSG00000141510", "ENSG00000082175"),
+     "group", c("a", "b"))
```

`splanova` is an implementation of the MIDAS approach suggested by Affymetrix. It produces an object with F-values and significance of alternative splicing, for each probeset and treatment in a multi-treatment experiment.