GeneRegionScan

Lasse Folkersen and Diego Diez

October 28, 2009

1 GeneRegionScan

This *GeneRegionScan* package contains functions for investigating smaller regions of the genome, consisting of one or a few genes, in datasets made with Affymetrix microarrays. While this does not make use of the entire set of genes, in the way that most microarray tools are designed, it does provide increased resolution and control over *all* data from the region of interest. Importantly this means that probe level data is the main unit of investigation, and a class *ProbeLevelSet* is therefore introduced to include descriptions of the sequence of each probe, and from which probe set it is derived. Functions are included to build *ProbeLevelSet*s and to analyse them in relation to their pData.

2 Example of building a ProbeLevelSet

In this section we will walk through the various ways a *ProbeLevelSet* can be created. An `exampleProbeLevelSet` is provided in the package, so if you want to go straight to testing of analysis functionality you can skip to the next section. An example data set is provided.

The first thing we will need, is to know which probeset IDs to analyse. This information can be copy-pasted from various database websites, such as for example `http://www.netaffx.com`, but it can also be obtained with the function `getProbesetsFromRegionOfInterest` if we have the annotation library files for our array type and we know at what position in the genome we want to look. This is quicker than web-lookup for larger regions.

In this example, we are interested in the gene FN1 and a region extending 1000 kb around it. There is no hard limit on the maximal region size, but the ProbeLevelSets can become very big if you choose a large region. From `http://www.affymetrix.com/products_services/arrays/specific/exon.affx` we download the file HuEx-1_0-st-v2.na26.hg18.transcript.csv, which contains information on each metaprobeset in the array type that this example uses. We also download the HuEx-1_0-st-v2.r2.dt1.hg18.full.mps, which contains information on the probesets found in each metaprobeset.

```
> library(GeneRegionScan)

> transcriptClustersFile <- "~/HuEx-1-0-st-v2.na26.hg18.transcript.csv"
> mpsToPsFile <- "~/HuEx-1-0-st-v2.r2.dt1.hg18.full.mps"
> listOfProbesets <- getProbesetsFromRegionOfInterest("HuEx-1-0-st-v2",
+     "chr2", 215889955, 216106710, transcriptClustersFile = transcriptClustersFile,
+     mpsToPsFile = mpsToPsFile)
```

If we were working with arrays, such as the 3'IVT type arrays where there existed a .db annotation database within the Bioconductor framework, this would have been made much simpler by a call to the relevant database. No downloads would therefore have been necessary, except for the relevant .db file. However, for 3'IVT arrays there would also have been fewer probesets, so perhaps we could just have noted them directly from the web.

Having decided which probe sets we are interested in extracting probe level data from, we turn to the cel files. In the example you will need to change at least clfPath and pgfPath to match the paths of some cel files on your computer. If you are lucky the `aptCelExtractPath` is already included as a binary in the package and can be left out. Because of space limitations, this is only true for some operating systems where people are usually very lazy or very busy.

```
> aptCelExtractPath <- "~/apt-bin/apt-cel-extract"
> clfPath <- "~/HuEx-1-0-st-v2.r2.clf"
> pgfPath <- "~/HuEx-1-0-st-v2.r2.pgf"
> myProbeLevelSet <- getLocalProbeIntensities(listOfProbesets,
+     "~/test-cels", aptCelExtractPath = aptCelExtractPath, pgfPath = pgfPath,
+     clfPath = clfPath)
```

This command will run for some time depending on the memory of the computer in use. By calling Affymetrix Power Tools instead of extracting the intensities ourselves, using a tool suchs as `readCelIntensities` from the *affxparser* package, we ensure that the memory requirements are drastically lessened for the normalization step. Otherwise, we would have to load the entire dataset into R, even though we only needed a few probes. Affymetrix Power Tools are available from `http://www.affymetrix.com/partners_programs/programs/developer/tools/powertools.affx`.

In addition to obtaining the probe level intensities and normalizing them to all probes in the cel file using quantiles normalization, the `getLocalProbeIntensities` also parses the pgf-file supplied and returns the sequence of each of the probes of interest. These are saved in the *featureData* of the *ProbeLevelSet* and can be obtained as follows:

```
> getSequence(myProbeLevelSet, 1:5)
```

If you have access to a remote computer with more memory than your local machine, you can use the `getServerProbeIntensities` function, which is a wrapper around `getLocalProbeIntensities` with the extra functionality that it sends the hard calculations to the remote computer and automatically retrieves the *ProbeLevelSet*. However, as specified in the documentation the funciton, you will need a few extra command line tools to negotiate the network.

The *ProbeLevelSet* inherits the *ExpressionSet* and all the usual methods can be used with it, for example if we want to include pData. In this package, there is a specialized possibility of adding pData called `addSnpPdata`. A main goal of the package is to be able to investigate splice variants and genotype connections. This function will parse a SNP data file, which must be of the same format as exported by `www.hapmap.org`. It will add its per-sample data content to the pData section of the ProbeLevelSet / ExpressionSet, and its metadata to the notes of the set.

3 Example of using a ProbeLevelSet

In the *GeneRegionScan* an `exampleProbeLevelSet` is included. This set was built as specified in the section above, but contains some fictive pData that we will use in this section.

```
> data(exampleProbeLevelSet)
> pData(exampleProbeLevelSet)[1:3, ]
```

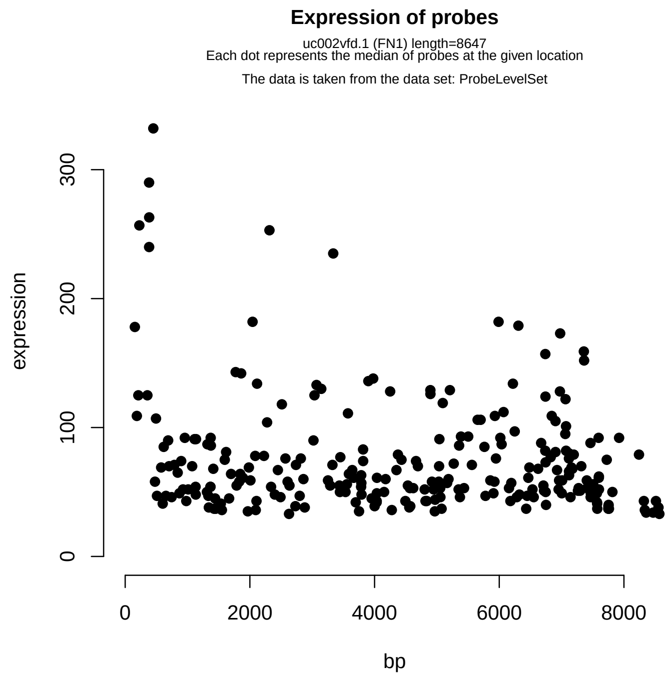
```
         age    case gender genotype1 genotype2 genotype3
X100.cel  60 Disease Female        CC        CT        AG
X101.cel  70 Disease Female        CC        TT        AG
X102.cel  80 Control   Male        CT        CT        AA
```

The data also contains two nucleotide sequences: `mrna` and `genomic`. The first one is the mRNA sequence of FN1, an isoform known as fibronectin 1 isoform 2 preproprotein. FN1 has at least 16 different known isoforms. The sequence was loaded by a call to `readFASTA` from the package *Biostrings*, using a regular FASTA-formatted text file as argument. The second sequence is the DNA sequence of the gene, with one FASTA entry per exon. We will return to that later.

The first thing we might want to do was to get an idea if any parts of the gene are not expressed at all. In this case, we might be investigating the wrong isoform, and would perhaps be better of finding another mRNA sequence. We do this by the simplest possible call to `plotOnGene`:

```
> plotOnGene(exampleProbeLevelSet, mrna)
```

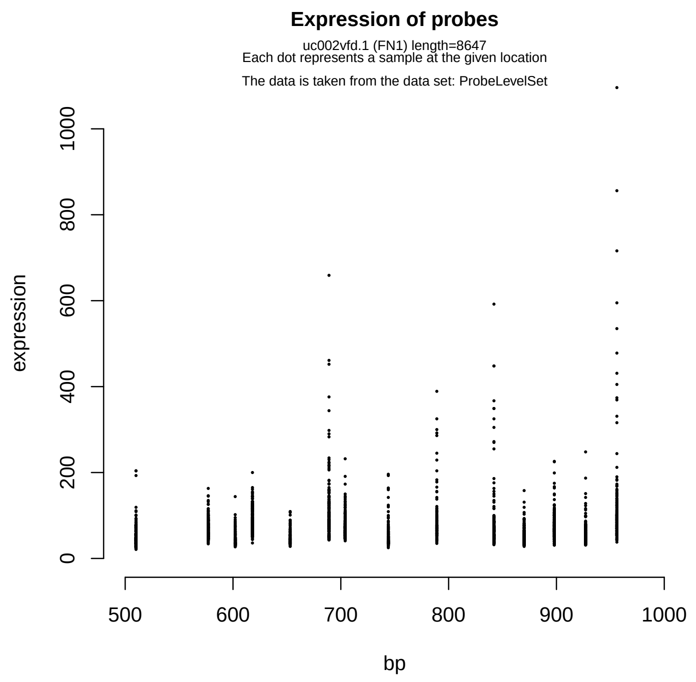
```
[1] "Probe data for 660 probes found as featureData in the expressionset ProbeLevelSet"
[1] "Investigating 8647 bp sequence from uc002vfd.1 (FN1) length=8647"
[1] "Found 229 probes matching in matchForwardAntisense"
[1] "Found 25 probes matching in matchReverseSense"
```



This produces a plot where the x-axis shows the length of gene (in this case the object `mrna`) in bp and the y-axis shows the quantiles normalized probe intensity level i.e. the expression level. With command just given, each dot on the plot is the median of expression levels in all samples for the probe at the location indicated on the x-axis. Summarising by median is a default mechanism, but perhaps we would be interested in seeing more detail for some part of the gene:

```
> plotOnGene(exampleProbeLevelSet, mrna, summaryType = "dots",
+     interval = c(500, 1000))
```

```
[1] "Probe data for 660 probes found as featureData in the expressionset ProbeLevelSet"
[1] "Investigating 501 bp sequence from uc002vfd.1 (FN1) length=8647"
[1] "Found 14 probes matching in matchForwardAntisense"
```



This will zoom in on the first 500 bp of the gene, while preserving the x-axis labels, and show the expression level of each single sample as a dot. This is the maximum resolution possible with Affymetrix arrays, but it is rarely seen, since data is most often summarised by probeset or metaprobeset. In the case of Human Exon ST 1.0 arrays, like here, a probeset is roughly equal to an exon - or four probes. This can be seen by adding the exon structure to the plot:

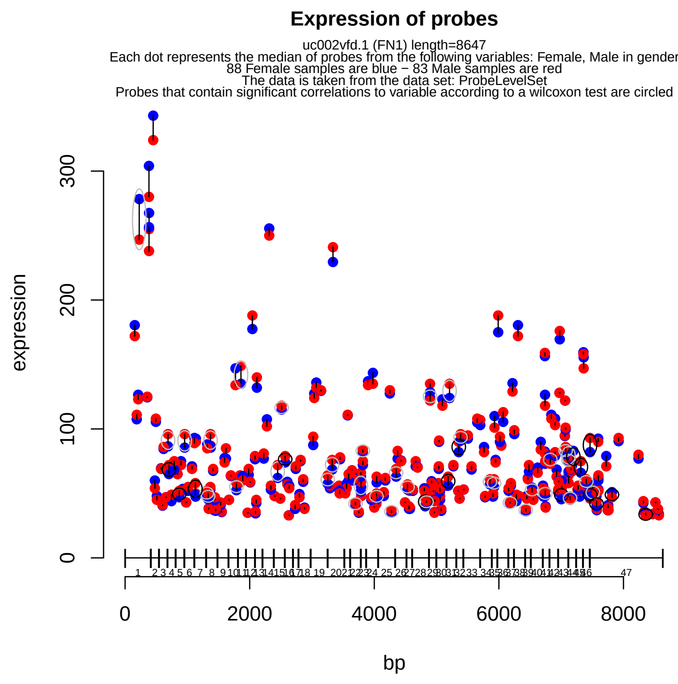
```
> exonStructure(mrna, genomic)
```

A sequence like the one in `genomic`, which is divided by exons, allows the visualization of the exon structure on the plots that we just made. By far the easiest way to obtain sequences like this, is to go to `http://genome.ucsc.edu`, find your gene of interest, download first the mRNA sequence as FASTA, and the genomic sequence as FASTA. When downloading genomic sequence it will ask for the "Sequence Retrieval Region Options". Make sure you specify exons only and "One FASTA record per region".

Instead of just looking at summarized expression values, we will often want to include sample condition from pData in the plot. This is done with the 'label' argument, specifying one of the pdata column names. In example below, 'gender' has been given as the label. In addition, the new argument 'testType' has been used. This specifies an optional test to employ when searching for differences. Using this argument will highlight probes that change significantly between the label groups. In the example below, there is very few probes with significant change between genders. The ones that are, are indicated by grey and black circles.

```
> plotOnGene(exampleProbeLevelSet, mrna, label = "gender", testType = "wilcoxon",
+     verbose = FALSE)
```

```
[1] "Found 229 probes matching in matchForwardAntisense"
[1] "Found 25 probes matching in matchReverseSense"

> exonStructure(mrna, genomic)
```



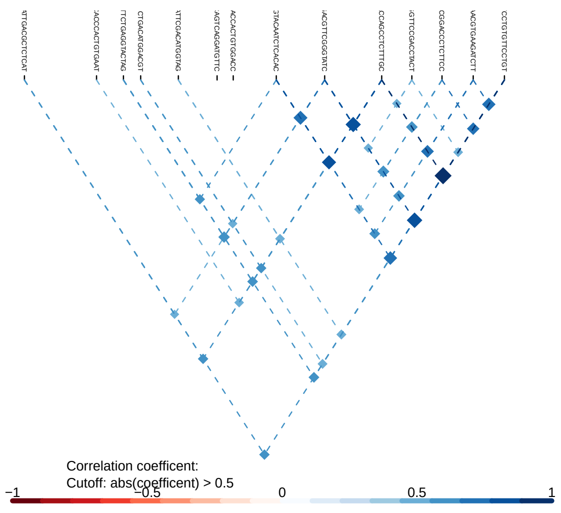
If the case were that the entire gene was significantly regulated between two groups of interest, then a standard summarization could just as well have shown it. However, if only some exons of the gene are regulated, such as in the case of a transcript isoform, perhaps unknown, that is changed, this kind of plot will reveal it. One example is included here:

```
> plotOnGene(exampleProbeLevelSet, mrna, label = "genotype1", testType = "linear model",
+     verbose = FALSE)

[1] "Found 229 probes matching in matchForwardAntisense"
[1] "Found 25 probes matching in matchReverseSense"

> exonStructure(mrna, genomic)
```

Try to run the same example using genotype2 and genotype3 as labels as well. While the two first genotypes are not related specifically to the localization of the probes in the gene, the third genotype would only be picked up by investigations like this, and could be a case of a hitherto unknown transcript isoform.

The final functionality to be demonstrated here, is the `plotCoexpression`. This function can be used to investigate coexpression. It works by calculating the Pearson correlation coefficient of all possible pairwise combinations of probes found in the gene. It then shows a colour map of all pairings with coefficents above the `correlationCutoff`. Furthermore, the function can be instructed to print the probe level information found in the `featureData` on screen. (In the example below we use a subset of the gene, because it takes quite a while to calculate correlation of all pairings of probes in long genes)

```
> mrna_subset <- mrna
> mrna_subset[[1]]$seq <- substr(mrna_subset[[1]]$seq, 500, 1000)
> plotCoexpression(exampleProbeLevelSet, gene = mrna_subset, correlationCutoff = 0.5,
+     probeLevelInfo = c("probeid", "sequence"), verbose = FALSE)
```



Finally the function `geneRegionScan` can be used. This is a wrapper that merges the functionalities we have seen and allows the visualization of multiple genes in the same pdf-file. It has been tested with up to four genes and is useful when analysing regions with complicated interactions between different neighbouring genes.

```
> geneRegionScan(exampleProbeLevelSet, gene = mrna, genomicData = genomic,
+     label = "genotype3", testType = "linear model", correlationCutoff = 0.5,
+     probeLevelInfo = c("probeid", "sequence"), verbose = FALSE)
```

Session Info

```
> sessionInfo()
```

```
R version 2.10.0 (2009-10-26)
x86_64-unknown-linux-gnu

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=C              LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] tools     stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
[1] GeneRegionScan_1.2.0 Biostrings_2.14.0    IRanges_1.4.0
[4] Biobase_2.6.0

loaded via a namespace (and not attached):
[1] affxparser_1.18.0  RColorBrewer_1.0-2
```