CAGExploreR Vignette

Emmanuel Dimont

January 11, 2014

Contents

1 Basic Workflow Example

1.1 Installation

To install **CAGExploreR**, you need to have R version 3.0.1 or newer (`http://www.r-project.org/`). You can download the package from `https://github.com/edimont/CAGExploreR/` into a local directory (we'll assume that this directory is *D:/Downloads*, but this can be anything). There are two files called **CAGExploreR_1.0.tar.gz** (main source code) and **DTGenAnnot_1.0.tar.gz** (hg19 genome annotation). There is no need to unpack or decompress the contents of these files. Next open R and run the following code (you only have to do this once). Since **CAGExploreR** depends on several other packages, you will need to install them first. You may be asked if you want to update some previously installed packages but this does not affect **CAGExploreR** in any way. When downloading from **CRAN** you may be asked to choose a mirror to download from. Any selection is fine.

```
# Packages from Bioconductor
source("http://bioconductor.org/biocLite.R")
biocLite("edgeR")
# Packages from CRAN
install.packages(c("R2HTML", "data.table", "rbamtools"))
```

Once the dependencies are installed, we can now install the packages from source, the order is important! Be sure to set the working directory to the one that contains the downloaded package files.

```r
setwd("D:/Downloads")  #Locate the folder with the downloaded package files
install.packages("DTGenAnnot_1.0.tar.gz", type = "source")  #Install this first
install.packages("CAGExploreR_1.0.tar.gz", type = "source")
```

Total time for installing the package and all other dependencies can take several minutes.

1.2 Loading the package

Once the package is installed, you can load it just like any other R package, simply type:

```r
library(CAGExploreR)

## Loading required package:  rbamtools
## Loading required package:  R2HTML
## Loading required package:  data.table
## Loading required package:  edgeR
## Loading required package:  limma
## Loading required package:  DTGenAnnot
```

R may report that certain functions are masked, this is normal.

1.3 Loading Data

CAGExploreR comes pre-loaded with tables that consist of CAGE-Seq tag counts at promoter regions from two samples: the MCF7 breast cancer and the A549 lung cancer cell lines. The example workflow presented here will use this dataset to demonstrate the capabilities of **CAGExploreR**. For convenience, you can choose to load this data directly from the package and skip the next section. However in practice when analyzing your own personal datasets, you would need to complete the next section.

1.3.1 Obtaining tag counts from raw BAM files

For this example, the CAGE-Seq BAM files can be downloaded from the ENCODE website `http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeRikenCage/`. In addition to the files below you will also need to download the corresponding index files (*.bai).

- wgEncodeRikenCageA549CellPapAlnRep1.bam
- wgEncodeRikenCageA549CellPapAlnRep2.bam
- wgEncodeRikenCageMcf7CellPapAlnRep1.bam
- wgEncodeRikenCageMcf7CellPapAlnRep2.bam

Files 1-2 come from the A549 cell line and files 3-4 come from the MCF7 cell line. You will notice that these files were generated from RNA that was extracted from the whole cell with two replicates from each condition.

Once we have these files in our local folder, we can load their names into R. It is important to provide IDs to each file in the format "**sample name**.*replicate number*". Files that correspond to replicates from a single condition should have the same prefix and different numbered suffix separated by a dot as shown below. In the example code below, we assume that the BAM and BAI files are all saved in the **D:/bam** directory.

```r
setwd("D:/bam")  #set the working directory to where the bam and bai files are located
files = dir()[grep("bam", dir())]
my.bai.files = dir()[grep("bam.bai", dir())]
my.bam.files = setdiff(files, my.bai.files)
my.ids = c("a549.1", "a549.2", "mcf7.1", "mcf7.2")
```

You can also manually enter the names of the files as follows:

```
my.bai.files = c("wgEncodeRikenCageA549CellPapAlnRep1.bam.bai", "wgEncodeRikenCageA549CellPapAlnRep2.bam
    "wgEncodeRikenCageMcf7CellPapAlnRep1.bam.bai", "wgEncodeRikenCageMcf7CellPapAlnRep2.bam.bai")
my.bam.files = c("wgEncodeRikenCageA549CellPapAlnRep1.bam", "wgEncodeRikenCageA549CellPapAlnRep2.bam",
    "wgEncodeRikenCageMcf7CellPapAlnRep1.bam", "wgEncodeRikenCageMcf7CellPapAlnRep2.bam")
my.ids = c("a549.1", "a549.2", "mcf7.1", "mcf7.2")
```

The next step is to load promoter and gene region definitions. **CAGExploreR** provides both **FANTOM5** and **MPromDB** promoter regions. In this example we shall be using the **FANTOM5** regions which are more comprehensive. User-supplied regions can also be used as long as their structure and format corresponds to the ones shown here. Let's take a look at the promoters that we just loaded to see how the file is structured.

```
data(F5.hg19.promoters)  #this object contains both promoter and gene regions
head(F5.hg19.promoters)

   chr strand  start    end      gene
1 chr1      - 910579 917517 C1orf170@
2 chr1      - 917466 917485  C1orf170
3 chr1      - 917507 917517  C1orf170
4 chr1      - 934342 935552     HES4@
5 chr1      - 934985 934997      HES4
6 chr1      - 935277 935309      HES4
```

Each row corresponds to a promoter region that is assigned to a gene, usually by virtue of its genomic vicinity. Coordinates are based on the hg19 genome build. Gene names are HGNC-approved names only. You will notice that some gene names contain the character @ at the end. This corresponds to rows which are "full-gene" regions that are calculated by taking the union of the whole gene plus all promoters defined for that gene. These are used for calculating "coverage quality" in downstream analysis. Whole gene regions are obtained from the **txdb.hsapiens.ucsc.hg19.knowngene** Bioconductor package. NOTE: not all genes have "full-gene" regions available, this will depend on the reference database used.

The **FANTOM5** promoter set contains a total of 77207 promoters across 14449 multi-promoter genes, out of which 14253 genes have full gene regions available for "coverage quality" assessment. The average number of promoters per gene is 5.3434 with an average promoter region length of 26 bases (sd=21).

For comparison, the **MPromDB** promoter set contains a total of 12200 promoters across 4436 multi-promoter genes, out of which 4424 genes have full gene regions available for "coverage quality" assessment. The average number of promoters per gene is 2.7502 with an average promoter region length of 784 bases (sd=767).

Now that we have our input files and the promoter region definitions, we need to quantify the number of CAGE-Seq tags that map to each of these regions.

```
my.promoters = definePromoters(F5.hg19.promoters)
my.genes = defineGenes(F5.hg19.promoters)

mcf7a549.raw.counts.F5 = countTags(my.bam.files, my.bai.files, my.ids, my.genes, my.promoters)
# This takes 2-10 minutes per BAM file
```

1.3.2 Loading data included with CAGExploreR

For the purposes of this exercise to save time, we can load the tag counts table from within the package. Let's take a look at what this object looks like:

```
data(mcf7a549.raw.counts.F5)
mcf7a549.raw.counts.F5  #use of the head() optional here

$depth
  a545.1   a545.2   mcf7.1   mcf7.2
28855537 18239960 52947757 36259720

$counts
                             region     gene a545.1 a545.2 mcf7.1 mcf7.2
    1: chr10:100143322..100174982,- PYROXD2@    457    330   1154   3530
    2: chr10:100174900..100174956,-  PYROXD2      5      4      1     21
    3: chr10:100174957..100174982,-  PYROXD2    282    200    925    648
    4: chr10:100216834..100995635,-   HPSE2@     27     33    185     54
    5: chr10:100995440..100995474,-    HPSE2      0      0      0      0
   ---
91456:      chrY:7141999..7142040,+     PRKY      1      1      0      0
91457:      chrY:7142047..7142063,+     PRKY      0      0      0      0
91458:      chrY:7142069..7142108,+     PRKY      1      2      0      0
91459:      chrY:7142137..7142158,+     PRKY      0      0      0      0
91460:      chrY:7142165..7142189,+     PRKY      0      0      0      0
```

The first element of this list **depth**, shows the total number of tags mapped in each library. The second element **counts** is a data.frame showing the number of CAGE-Seq tags mapping to each library and promoter or gene region. User-supplied count tables such as this one can also be provided directly instead of counting from BAM files, as long as all of the necessary data is available (see next section).

For the purposes of this vignette, we will only use a subset of the data to make downstream calculations run faster. Let's subset our data to only those regions on chromosome 22:

```
get = which(osc2info(mcf7a549.raw.counts.F5$counts$region)$chr == "chr22")
mcf7a549.raw.counts.F5$counts = mcf7a549.raw.counts.F5$counts[get]
```

1.4 Loading your own counts data - Creating a DGEList Object

Whether or not you already have your own CAGE tag count tables, you will need to perform this step. In order to make downstream analysis more streamlined, we take advantage of the **DGEList** object from the **edgeR** package that conveniently groups useful data together neatly. To convert the data we have already to DGEList, we do the following. Similarly, if you already have your own count data, you can convert your data into a **DGEList** object.

```
#osc2info converts genomic regions in the format "chr:start..end,strand" to an R list
annotations = osc2info(mcf7a549.raw.counts.F5$counts$region)
my.ids = c("a549.1","a549.2","mcf7.1","mcf7.2")

my.data = DGEList(

        counts = mcf7a549.raw.counts.F5$counts[,-c(1,2),with=FALSE], #remove columns 1&2
        lib.size = mcf7a549.raw.counts.F5$depth,
        group = my.ids,                          #will not pool replicates
        #group = c("a549","a549","mcf7","mcf7"), #pools replicates
        #group = select(my.ids,".",1),           #pools replicates (alternative)

        genes = data.frame(
                chr = annotations$chr,
                strand = annotations$strand,
                start = annotations$start,
                end = annotations$end,
                gene = mcf7a549.raw.counts.F5$counts$gene
                ),

        remove.zeros = FALSE
)
```

A **DGEList** object requires four main inputs that we provide here: the counts table, the total library sizes (can be omitted but we discourage this), a grouping table that provides sample group annotation, and a table of "gene" definitions. **DGEList** objects were originally developed for use in differential gene expression data analysis, that is the reason why the word "gene" is used here, however in our case these are both promoter regions *and* gene regions for coverage analysis. The last option should be used to specify that we do not want to discard promoters or genes with all zero counts.

A crucial thing to mention here is that the way the **group** option is specified determines whether or not, and how replicates are combined in downstream analysis. In the example here we specify **my.ids** as the group identifiers, and because these are all unique, this implies that the replicates are not going to be pooled, but kept separate in later steps. In the example, alternatives are provided that have been commented out (using #) which would be used if pooling of replicates was desired. The first alternative where we explicitly specify groups, means to say that those samples that share the same group id, will be pooled, i.e. the first 2 and the last 2 replicates will be combined. The last alternative gives the same result but demonstrates the use of the **select** function which extracts the prefix of the sample id preceding the period. This syntax may be useful when there are a large number of samples available.

Now let's take a look at what this object looks like. This output should be familiar to **edgeR** users, but it's basically the same information in more compact form.

```
my.data

An object of class "DGEList"
$counts
  a545.1 a545.2 mcf7.1 mcf7.2
1     44     30    262    232
2     98     61    576    357
3      7      3     95     33
4      5     13     18      8
5      0      0      0      0
1885 more rows ...

$samples
        group lib.size norm.factors
a545.1 a549.1 28855537            1
a545.2 a549.2 18239960            1
mcf7.1 mcf7.1 52947757            1
mcf7.2 mcf7.2 36259720            1

$genes
    chr strand    start      end    gene
1 chr22      + 17565841 17565896  IL17RA
2 chr22      + 17565841 17596584 IL17RA@
3 chr22      + 17565903 17565929  IL17RA
4 chr22      - 17597189 17602265  CECR6@
5 chr22      - 17601758 17601792   CECR6
1885 more rows ...
```

There are two additional optional steps that can be performed on this object using **edgeR** functionality. Samples can be normalized for RNA composition bias, and negative binomial dispersion coefficients calculated. This will affect the statistical significance of downstream results. We find that this is generally quite stringent and reduces the number of significant results obtained, and for exploratory analyses we do not recommend doing this. This can be done as follows, however for our example, we will skip them. Please refer to **edgeR** documentation for further details.

```
# To normalize samples
my.data = calcNormFactors(my.data)
# To estimate negative binomial dispersion (common or tagwise)
my.data = estimateCommonDisp(my.data, verbose = T)
my.data = estimateTagwiseDisp(my.data)
```

1.5 Handling Replicates

The user has the choice whether or not to pool replicates. We recommend pooling if inference is directed at genes with very low expression in which case pooling can increase their tag counts. However for most cases we do not recommend to pool. Whether one pools or not, the output is affected mainly at the visualization stage. The advantage of not pooling is that results can be filtered to contain only those in which there is agreement between replicates.

As mentioned previously, sample IDs follow the "**sample name**.*replicate number*" format, where replicates coming from the same sample have common sample name and different replicate number separated by a period. In this example there will be no pooling performed because of the way we defined our groups in the **DGEList** object in the previous step. NOTE: this step is required irrespective of whether one decides to pool or not. In addition, here we use the term "replicates" loosely, allowing the user to decide which samples to pool for analysis. In this example we are interested in comparing MCF7 and A549 cell lines to one another, but another choice of pooling might involve comparing between cancer v.s. normal or early-time point v.s. late-time point, etc. The user has the flexibility to pool samples any way they wish and this is achieved by assigning the proper group ids in the **DGEList** object "samples" section after it is created (previous section).

```
# This step is REQUIRED even if not pooling replicates
data.not.pooled = pool(my.data)
data.not.pooled

An object of class "DGEList"
$counts
  a545.1 a545.2 mcf7.1 mcf7.2
1     44     30    262    232
2     98     61    576    357
3      7      3     95     33
4      5     13     18      8
5      0      0      0      0
1885 more rows ...

$samples
        group lib.size norm.factors effective.lib.size
a545.1 a549.1 28855537            1           28855537
a545.2 a549.2 18239960            1           18239960
mcf7.1 mcf7.1 52947757            1           52947757
mcf7.2 mcf7.2 36259720            1           36259720

$genes
```

```
    chr strand    start      end    gene
1 chr22      + 17565841 17565896  IL17RA
2 chr22      + 17565841 17596584 IL17RA@
3 chr22      + 17565903 17565929  IL17RA
4 chr22      - 17597189 17602265  CECR6@
5 chr22      - 17601758 17601792   CECR6
1885 more rows ...

$common.dispersion
[1] 0

$tagwise.dispersion
[1] 0 0 0 0 0
1885 more elements ...

$sub.counts
     a545.1 a545.2 mcf7.1 mcf7.2
[1,]     44     30    262    232
[2,]      7      3     95     33
[3,]      0      0      0      0
[4,]      0      0      3      0
[5,]      0      0      1      0
1568 more rows ...

$super.counts
     a545.1 a545.2 mcf7.1 mcf7.2
[1,]     98     61    576    357
[2,]      5     13     18      8
[3,]     86    103   1335    438
[4,]     14     21     57     21
[5,]      8      6     24    227
312 more rows ...

$pooled.sub.counts
                           a549.1 a549.2 mcf7.1 mcf7.2   gene dispersion
chr22:17565841..17565896,+     44     30    262    232 IL17RA          0
chr22:17565903..17565929,+      7      3     95     33 IL17RA          0
chr22:17601758..17601792,-      0      0      0      0  CECR6          0
chr22:17602160..17602175,-      0      0      3      0  CECR6          0
chr22:17602200..17602265,-      0      0      1      0  CECR6          0
1568 more rows ...

$pooled.super.counts
                           a549.1 a549.2 mcf7.1 mcf7.2     gene dispersion
chr22:17565841..17596584,+     98     61    576    357   IL17RA          0
chr22:17597189..17602265,-      5     13     18      8    CECR6          0
chr22:17618410..17646177,-     86    103   1335    438    CECR5          0
chr22:17660192..17739198,-     14     21     57     21    CECR1          0
chr22:18043133..18073647,+      8      6     24    227 SLC25A18          0
312 more rows ...

$pooled.samples
       lib.size effective.lib.size
a549.1 28855537           28855537
```

```
a549.2 18239960            18239960
mcf7.1 52947757            52947757
mcf7.2 36259720            36259720
```

The **DGEList** object is now augmented with additional data. Because we did not perform sample normalization, the effective library size is equal to the original library size. Similarly, because we did not calculate negative binomial dispersion coefficients, they are taken to be zero. We also see that we now have **sub.counts** and **super.counts**. In CAGE, the "sub" (subset) counts correspond to promoter level counts, and the "super" (superset) counts are the gene region counts. This terminology is used to allow more general applications of **CAGExploreR** to other scenarios where counts are separated into subsets and supersets, e.g. subsets could be exon regions, etc. We also have some additional "pooled" data elements that show pooled counts across sample replicates: "pooled.sub.counts" are the pooled promoter-level counts, and "pooled.super.counts" are the gene-level counts and in essence, the gene expression.

1.6 Obtaining Promoter Composition Results

Once we have a data object that contains the pooled CAGE-Seq tag counts for all samples across all genes and promoters, we can now obtain differential promoter composition (DPC) statistics.

1.6.1 A broad view: Gene-level statistics

This step can take several minutes when working on the whole genome level. By default, this function call will generate gene-level statistics, let's take a look:

```
results = diffcomp(data.not.pooled)
head(results)

        entropy.Reduction    pvalue       fdr geneHetero  coverage
RASL10A            1.0000 0.000e+00 0.000e+00    0.07092   0.2794
CSF2RB             1.0000 1.000e+00 1.000e+00    0.02999   0.1274
NEFH               1.0000 1.000e+00 1.000e+00    0.12958   0.1214
MGAT3              0.4511 3.748e-02 8.666e-02    0.01650   0.1674
OSBP2              0.4447 3.114e-25 4.414e-24    0.03778   0.1270
RTDR1              0.3916 5.899e-02 1.317e-01    0.04126   0.2689
                                                              dominant.promoter.switch
RASL10A                              chr22:29711645..29711694,-|chr22:29711704..29711718,-
CSF2RB  chr22:37309662..37309694,+|chr22:37318050..37318074,+|chr22:37318082..37318109,+
NEFH                                 chr22:29876197..29876215,+|chr22:29884834..29884866,+
MGAT3   chr22:39853258..39853330,+|chr22:39868762..39868785,+|chr22:39868786..39868797,+
OSBP2                                chr22:31090839..31090889,+|chr22:31199037..31199051,+
RTDR1   chr22:23484149..23484160,-|chr22:23484167..23484191,-|chr22:23484246..23484283,-
        RepAgree
RASL10A        1
CSF2RB         2
NEFH           2
MGAT3          1
OSBP2          1
RTDR1          2
```

This is the main table showing DPC results. Remember that the motivation behind detecting DPC is to determine if the set of promoters for a gene are being utilized differently across conditions.

- **entropy.Reduction** is the main effect measure for DPC and the table is sorted by this value, with genes with the highest DPC showing up first. A value of 0 corresponds to no DPC, i.e. the relative transcription occurring at the different promoters is the same across conditions. Conversely, a value of 1 would indicate maximal DPC and that each condition transcribes a gene from a unique promoter.

- **pvalue** is the corresponding one-sided p-value testing the null hypothesis that the entropy reduction is zero.

- **fdr** The p-value is corrected for multiple comparisons using the Benjamini-Hochberg method by default. The user can supply any correction method supported by the **p.adjust** function in R.

- **geneHetero** is an entropy-based measure of gene expression heterogeneity across conditions. A value of 0 corresponds to no differential gene expression between conditions, and a value of 1 means that one or more particular conditions have non-zero gene expression while in another condition(s) the gene is not expressed at all.

- **coverage** compares the number of tags mapped to promoter regions to the total number of tags mapped to the entire gene region. The value reported is the mean across conditions. Since CAGE-Seq tags are strictly associated with the 5′ mRNA cap, as long as the promoter regions are defined correctly, we do not expect to see any tags aligning to other nearby non-promoter regions in theory. A value of 1 would mean that the promoter regions defined capture all of the tags found in the entire gene region. In practice however, there is noise present and we do expect some tags mapping outside our known promoters. It should be noted that a significant deviation from 1 could point to the existence of a novel promoter that is not accounted for by the current definitions. A value close to 0 would mean that for that particular gene, the promoter definitions are inadequate and the results suspicious. This could also occur if the gene region was not defined correctly. Values greater than 1 are possible if the promoter definitions overlap one another. In either case, a value of greater than 1 or close to 0 imply that the results for that gene should be discarded and the promoter definitions adjusted as they do not provide a complete and unequivocal view of DPC. We recommend to filter results to values of coverage between 0.1 and 1. If a gene does not have a gene region defined in the promoter definitions file (e.g. GENE@), then coverage cannot be calculated and NA will appear.

- **dominant.promoter.switch** shows whether the dominant promoter switches between conditions. This is in the form of a pipe (|) delimited list of promoters if there is a switch in dominant promoters and is empty otherwise.

- **RegApree** is an integer indicating how well the replicates agree in terms of their promoter composition data. The PC data is clustered gene by gene and then the dendrogram cut into the number of distinct conditions available (in our example, 2). **RepAgree** is the maximum number of different conditions that appear in each arm of the cut tree. In essence, a value of 1 means that all replicates cluster together perfectly.

Let's subset our results to only those genes that pass the FDR-adjusted p-value cutoff of 0.001, have coverage between 0.1 and 1 and whose replicates cluster according to their respective condition. Beneath the output table are some examples of useful statistics of interest.

```
significant = subset(results, fdr < 0.001 & coverage >= 0.1 & coverage <= 1 & RepAgree == 1)
head(significant)

        entropy.Reduction    pvalue       fdr geneHetero coverage
RASL10A            1.0000 0.000e+00 0.000e+00    0.07092   0.2794
OSBP2              0.4447 3.114e-25 4.414e-24    0.03778   0.1270
BCL2L13            0.3674 0.000e+00 0.000e+00    0.18833   0.7360
MAPK1              0.2009 3.847e-22 4.325e-21    0.03032   0.2386
TXNRD2             0.1885 4.713e-62 1.182e-60    0.03850   0.6957
GAL3ST1            0.1806 1.924e-23 2.241e-22    0.28014   0.7440
                                     dominant.promoter.switch RepAgree
RASL10A chr22:29711645..29711694,-|chr22:29711704..29711718,-        1
OSBP2   chr22:31090839..31090889,+|chr22:31199037..31199051,+        1
BCL2L13 chr22:18121493..18121535,+|chr22:18189536..18189560,+        1
MAPK1   chr22:22221658..22221669,-|chr22:22221900..22221963,-        1
TXNRD2  chr22:19868692..19868724,-|chr22:19929321..19929356,-        1
GAL3ST1 chr22:30960879..30960895,-|chr22:30970588..30970622,-        1

# This is how many significant, high-coverage genes we are left with:
nrow(significant)

[1] 33

# This is the number of genes in which the dominant promoter switches between conditions:
sum(significant$dominant.promoter.switch != "")

[1] 13

# This is the number of genes that have differential promoter composition but very little
# to no differential gene expression:
sum(significant$geneHetero < 0.01)
```

```
[1] 1

# Differences in gene expression and differences in promoter composition do not appear to
# be correlated:
cor(significant$entropy.Reduction, significant$geneHetero)

[1] 0.01275
```

1.6.2 A detailed view: Promoter-level statistics

Now that we've found which genes have DPC and to what extent, we may be interested in finding out exactly which promoters are involved and the magnitude of their switching. To get such promoter-level statistics we make use of the **detailed** option. By default, results for all genes are displayed. Because up until now we did not pool our replicates, at this stage the analysis will make pairwise comparisons between conditions at the replicate level. This may not be of interest for most cases and it is here that we will perform pooling so that our results display pairwise comparisons between conditions are opposed to replicates.

```
my.data2 = DGEList(

        counts = mcf7a549.raw.counts.F5$counts[,-c(1,2),with=FALSE], #remove columns 1&2
        lib.size = mcf7a549.raw.counts.F5$depth,
        #group = my.ids,                          #will not pool replicates
        group = c("a549","a549","mcf7","mcf7"), #pools replicates
        #group = select(my.ids,".",1),           #pools replicates (alternative)

        genes = data.frame(
                chr = annotations$chr,
                strand = annotations$strand,
                start = annotations$start,
                end = annotations$end,
                gene = mcf7a549.raw.counts.F5$counts$gene
                ),

        remove.zeros = FALSE
)

data.pooled = pool(my.data2)

results.detailed = diffcomp(data.pooled,detailed=TRUE)
head(results.detailed,15)

                                                        comparison    gene theilU
1  chr22:29711645..29711694,-|chr22:29711704..29711718,-#a549|mcf7 RASL10A 1.0000
2  chr22:29711645..29711694,-|chr22:29711434..29711441,-#a549|mcf7 RASL10A 1.0000
3  chr22:29711645..29711694,-|chr22:29711476..29711484,-#a549|mcf7 RASL10A 1.0000
4  chr22:29711645..29711694,-|chr22:29711485..29711500,-#a549|mcf7 RASL10A 1.0000
5  chr22:29711704..29711718,-|chr22:29711434..29711441,-#a549|mcf7 RASL10A 1.0000
6  chr22:29711704..29711718,-|chr22:29711476..29711484,-#a549|mcf7 RASL10A 1.0000
7  chr22:29711704..29711718,-|chr22:29711485..29711500,-#a549|mcf7 RASL10A 1.0000
8  chr22:29711434..29711441,-|chr22:29711476..29711484,-#a549|mcf7 RASL10A 1.0000
9  chr22:29711434..29711441,-|chr22:29711485..29711500,-#a549|mcf7 RASL10A 1.0000
10 chr22:29711476..29711484,-|chr22:29711485..29711500,-#a549|mcf7 RASL10A 1.0000
11 chr22:37309662..37309694,+|chr22:37318082..37318109,+#a549|mcf7  CSF2RB 0.5794
12 chr22:37318050..37318074,+|chr22:37318082..37318109,+#a549|mcf7  CSF2RB 0.5794
13 chr22:37309662..37309694,+|chr22:37309631..37309642,+#a549|mcf7  CSF2RB 0.5794
14 chr22:37309662..37309694,+|chr22:37317717..37317729,+#a549|mcf7  CSF2RB 0.5794
15 chr22:37309662..37309694,+|chr22:37317772..37317784,+#a549|mcf7  CSF2RB 0.5794
         OR log2OR   pvalue     fdr
1  0.007092 -7.140 0.001002 0.01328
2  0.021277 -5.555 0.023843 0.17458
3  0.021277 -5.555 0.023843 0.17458
4  0.021277 -5.555 0.023843 0.17458
```

```
5  3.000000  1.585 0.539312 0.93785
6  3.000000  1.585 0.539312 0.93785
7  3.000000  1.585 0.539312 0.93785
8  1.000000  0.000 1.000000 1.00000
9  1.000000  0.000 1.000000 1.00000
10 1.000000  0.000 1.000000 1.00000
11 0.111111 -3.170 0.169871 0.59593
12 0.111111 -3.170 0.169871 0.59593
13 0.333333 -1.585 0.539312 0.93785
14 0.333333 -1.585 0.539312 0.93785
15 0.333333 -1.585 0.539312 0.93785
```

In this example we have a total of 4862 comparisons made. NOTE: some values such as the entropy reduction may change after pooling.

- **Comparison** specifies which pair of promoters and conditions are being compared.
- **Gene** specifies the gene.
- **theilU** is the entropy reduction.
- **OR** is the odds ratio and measures the strength of switching in promoter composition between the two conditions and promoters being compared. This ratio shows how much larger (or smaller) the odds of observing tags in the first v.s. the second promoter in the first condition are as compared to the odds of observing tags in the first v.s. the second promoter in the second condition. A value of 1 means that the odds are the same in both conditions, and so promoter composition does not change for this set of promoters when comparing these two conditions.
- **log2OR** is the log-base-2 transformed odds ratio. This transformation is useful to make it clearer which direction the odds ratio is in: if the OR is less than 1, the log will have a negative sign, and if the OR is greater than 1, the log will be positive. The log odds ratio also makes the scale more easily understood, varying from 0 (no switching) to infinity (maximum switching). You will notice that all results are sorted within genes by decreasing log odds ratio (irrespective of sign) so that most significant switches show up first within gene blocks.
- **P-value** corresponds to the odds ratio and tests whether it is significantly different from 1 and conversely whether the log odds ratio is significantly different from 0. Note that this is different from the p-value that corresponds to the overall entropy reduction for a given gene.
- **FDR** is the Benjamini-Hochberg adjusted p-value. Just as before, the user can supply any method for correcting for multiple comparisons as long as it is supported by the **p.adjust** function in R.

Now let's subset these results to only those significant, high-coverage genes with replicate agreement that we found in the previous section. In addition, let's also just look at those genes in which the dominant promoter switches.

```
significant.genes = rownames(subset(results, fdr < 0.001 & coverage >= 0.5 & coverage <= 1 &
    RepAgree == 1 & dominant.promoter.switch != ""))
sig.results.detailed = subset(results.detailed, gene %in% significant.genes)

# We are down to just this number of pair-wise promoter comparisons:
nrow(sig.results.detailed)

[1] 338

head(sig.results.detailed)
```

```
                                                     comparison    gene theilU
81 chr22:18121541..18121552,+|chr22:18189504..18189531,+#mcf7|a549 BCL2L13 0.3623
82 chr22:18189536..18189560,+|chr22:18121541..18121552,+#mcf7|a549 BCL2L13 0.3623
83 chr22:18121562..18121585,+|chr22:18189504..18189531,+#mcf7|a549 BCL2L13 0.3623
84 chr22:18121493..18121535,+|chr22:18189504..18189531,+#mcf7|a549 BCL2L13 0.3623
85 chr22:18189536..18189560,+|chr22:18121562..18121585,+#mcf7|a549 BCL2L13 0.3623
86 chr22:18189536..18189560,+|chr22:18121493..18121535,+#mcf7|a549 BCL2L13 0.3623
         OR log2OR     pvalue       fdr
81 6.426e+02  9.328  2.117e-10  9.359e-09
82 2.133e-03 -8.873 4.288e-112 2.606e-109
83 3.233e+02  8.337  5.370e-09  2.072e-07
84 2.378e+02  7.894  2.953e-08  1.040e-06
85 4.240e-03 -7.882  0.000e+00  0.000e+00
86 5.764e-03 -7.439  0.000e+00  0.000e+00
```

1.7 Generating Plots

1.7.1 Visualizing DPC in a gene of interest

In order to visualize differential promoter composition in a gene of interest, we need to specify the (un)pooled data object and the gene we want to look at. Let's see what happens in one of our top genes, *GAL3ST1*. Here we choose to use the non-pooled data so that we could see what happens at the replicate level. You can try and see what happens if you choose the pooled data version. NOTE: it can take a few seconds to generate the full figure when plotting for the first time.

```
plotcomp(data.not.pooled, "GAL3ST1")

 [1] "chr22:30960879..30960895,-" "chr22:30970588..30970622,-"
 [3] "chr22:30968839..30968859,-" "chr22:30968813..30968829,-"
 [5] "chr22:30970513..30970541,-" "chr22:30956754..30956769,-"
 [7] "chr22:30961419..30961468,-" "chr22:30968796..30968805,-"
 [9] "chr22:30970644..30970658,-" "chr22:30960863..30960867,-"
[11] "chr22:30970560..30970577,-" "chr22:30953574..30953584,-"
[13] "chr22:30953587..30953595,-"
null device
          1
```

The figure is composed of 3 parts: (a) promoter composition, (b) gene expression and (c) promoter location plots.

(a) is the main promoter composition plot and has the conditions in rows and promoters in columns, each having a different color. The size of each colored bar specifies the extent to which transcription occurs from that promoter (measured as a proportion of all tags mapping to all promoters) and the number of CAGE-seq tags mapping to that promoter region is printed inside the bar. The number for the promoter which has the largest expression is colored white to make it more clearly visible. The conditions are clustered together by similarity in these composition profiles. In our example, the clustering shows that replicates cluster together, and when comparing more than two conditions, this can help identify samples that share similar promoter composition profiles. We can clearly see in this example that MCF7 cells preferentially utilize the "orange" promoter, whereas A549 cells preferentially utilize the "red" one in addition to a host of others, indicating a very strong switch in promoter composition between the two conditions.

(b) has the same row structure as in (a) and shows the gene expression for the conditions and is measured in tags per million (tpm). This is calculated by taking all of the CAGE-Seq tags mapping to the entire gene region (if it is available) relative to all tags mapped in this condition, multiplied by a

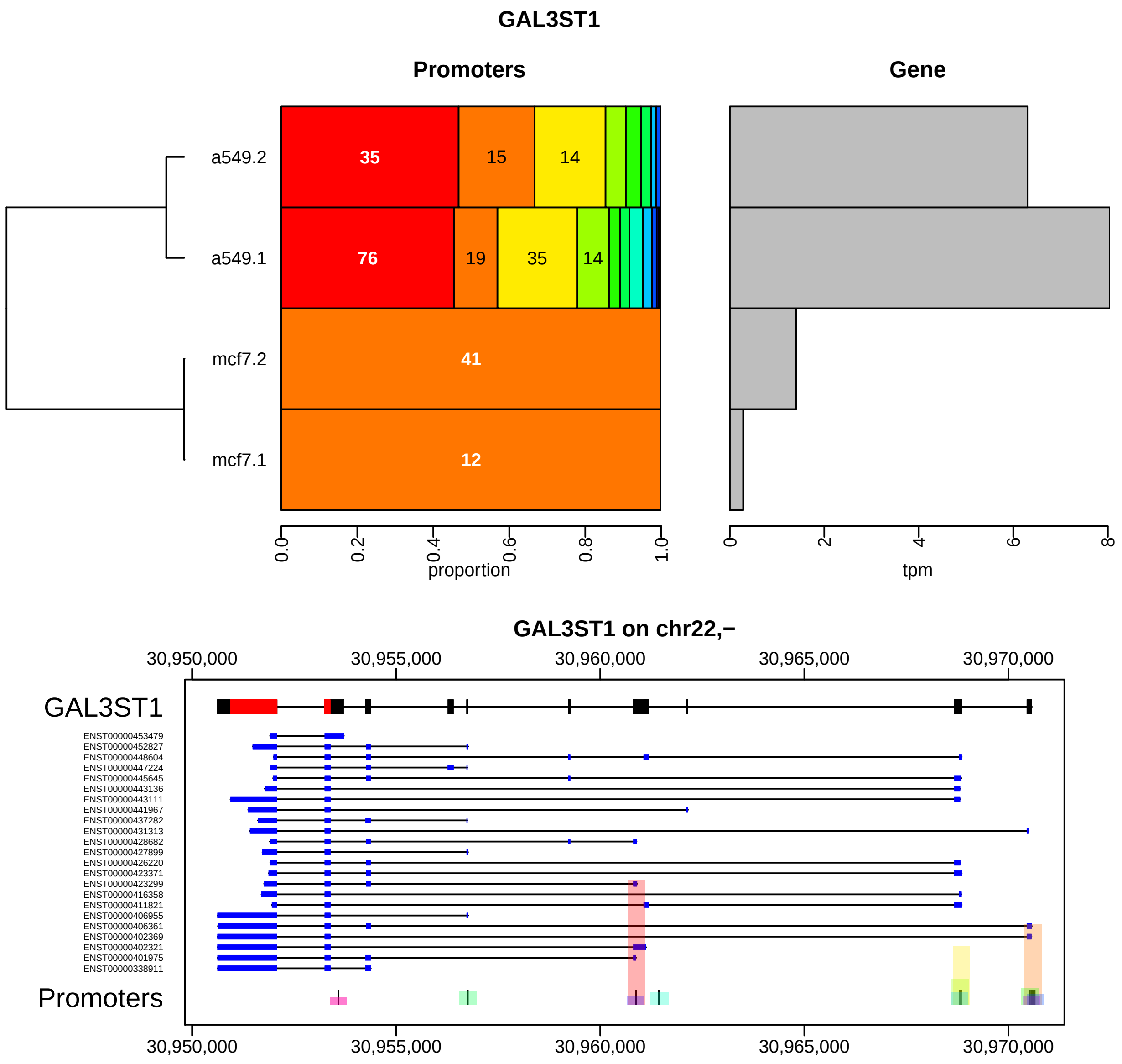

Figure 1: Differential promoter composition for GAL3ST1 gene with 13 promoters

million. If **edgeR** sample normalization was performed previously, then the effective library size is used. If the gene region coordinates have not been specified in the promoter definitions, then the total number of tags are taken by summing across all promoters available. Together with (a), these plots simultaneously demonstrate the extent of DPC and differential gene expression for the gene of interest.

(c) is a genomic region plot showing where the promoters are located relative to the gene model and known transcript models from ENSEMBL. The gene model is shown in black on top (known CDS are in red), the transcripts in blue in the middle, and the promoter regions in various colors at the bottom, where the colors correspond to the same color-code used in panel (a) for clarity. Each promoter region also has a colored vertical bar around it, the height of which gives a sense of the relative amount of transcription occurring from the promoters when summing over all conditions. NOTE: the coordinates for this plot are chosen in such a way as to fit all of the promoter regions comfortably on the screen, and so this means that most of the time, only the 5 prime end of the gene or another portion of it is visible. In this example we are on the negative strand and transcription is going from right to left. We can see that MCF7 cells transcribe the full-length transcript, whereas A549 cells have a more diverse collection, preferring a shorter transcript that skips 3 upstream exons.

After a call to **plotcomp()**, the coordinates of the promoter regions are displayed in the R console which we can see above. The promoters are ordered in the same way as in the plot, left to right as displayed.

1.7.2 Making multiple plots in HTML

Instead of making plots for each gene one by one, we can perform batch generation into an HTML document:

```r
html.report(data.not.pooled, rownames(significant)[1:30])
```

We once again have to specify the pooled data object, followed by the names of the genes we would like the plots of. In this example we specify to make plots for the top 30 significant genes from the significant subset our results table. Similarly we can specify any number of genes in any order that are of interest (e.g. a list of transcription factors, etc.) This will create an HTML file called *Switch Report.html* in the current directory, together with a folder called *Figures* that will contain individual plots inside. NOTE: if such a file/directory exists already, all files will be overwritten. You can change the names of report files and figure directory easily however:

```r
html.report(data.not.pooled, my.genes, fig.dir = "myFigures", report.name = "my.report")
```

1.7.3 Genome-wide DPC volcano plot

One possible way to visualize DPC on a genome-wide basis is to generate a volcano plot that displays the effect measure on the x-axis and the statistical significance on the y-axis. Since the effect measured as an odds ratio is one-sided, we can make it extend across the entire x-axis by taking its logarithm. Similarly we can take the negative of the logarithm of the multiplicity corrected p-value for the y-axis. This way, more statistically significant results are higher up, and larger effect measures are away from the origin.

```r
plot(y=-log(results.detailed$fdr,base=10),x=results.detailed$log2OR,
xlim=c(-10,10),ylim=c(0,50),col=rgb(0,100,0,60,maxColorValue=255),

pch=20,ylab="-log10(q-value)",xlab="log2(odds ratio)",
main="MCF7 v.s. A549 cell lines")

abline(v=0,h=0)
```

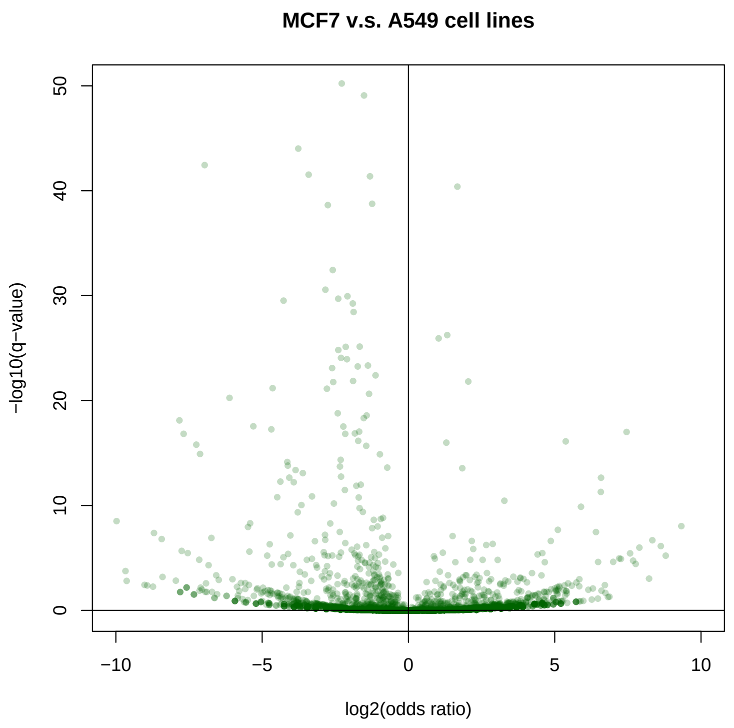

Figure 2: Genome-wide volcano plot for comparing differential promoter composition between 2 cell lines