The REIDS Package: Vignette

Marijke Van Moerbeke

April 13, 2018

Contents

1 Introduction

This vignette describes how to use the REIDS package starting from the raw microarray data contained in .CEL files. For the computation of the REIDS model we do recommend the use of a supercomputer as it requires memory and time for the data to be processed. All other methods can be conducted in reasonable time on an ordinary laptop but those who have a high-performance computing (HPC) service freely available have the advantage to use this service for the regular analysis as well. We will demonstrate an example step by step after giving a short introduction of the used methodology. The package accompagnies the papers by Van Moerbeke et al. (2017) and Van Moerbeke and Kasim (2018) which introduce the REIDS model.

2 Methodology

The Random Effects for Identification of Differential Splicing (REIDS) model is situated in a mixed model framework. Alternative splicing detection can now be formulated as variance decomposition in a random effects model with a random intercept per exon taking the gene expression into account. The following can be said. The *between array variability* of an alternatively spliced exon would be higher than the *within array variability* among the exons of the same transcript cluster. A non-alternatively spliced exon would have a between array variability that is at most the within array variability across all exons of the same transcript cluster. These hypotheses can be formulated as a two-stage mixed effect model for alternative splicing detection.

The model is fitted on the observed PM probe intensities levels as:

$$log2(PM_{ijk}) = p_j + c_i + b_{ik} + \epsilon_{ijk}, \tag{1}$$

Here, parameter p_j denotes the effect of probe j, c_i is the overall gene effect of array i and b_{ik} is an exon specific deviation from this overall effect. The background noise $\epsilon_{ijk(j)} \sim N(0, \sigma^2)$ captures the within array variability with σ^2. Differential expression between the arrays is expected to be negated by incorporating the gene effect parameter c_i. By consequence, the exon specific parameters show the deviation of a particular exon from its corresponding gene level. If there is only a small deviation, it can be said that the exon is present in the sample. A large deviation however shows that the exon likely absent. Note that the exon specific deviations are random effects assumed to be normally distributed, $b_{ik} \sim N(\mathbf{0}, \mathbf{D})$. The $K \times K$ covarianc matrix D contains the exon specific signals with τ_k^2 on its diagonal and K the number of exons in a transcript cluster.

The advantage of a mixed model formulation for alternative splicing detection is the existence of a standard score to quantify the trade-off between signal and noise. We term this score an "exon score" but it is also known as the intra-cluster correlation (ICC) of linear mixed models. The score is defined as the ratio of the exon specific signal to the noise across all exons of a transcript cluster. For a probeset k of a transcript cluster this is formulated as:

$$\rho_k = \tau_k^2/(\sigma^2 + \tau_k^2),$$

where σ^2 is the same for all probesets belonging to the same transcript cluster. It intuitively follows from the definition of the exon score that an equity threshold between exon specific signal and transcript level noise is 0.5. This could be used as a threshold for identifying an alternatively spliced probeset among the probesets of a transcript cluster. A value of $\rho_k > 0.5$ puts more weight in favor of probeset k to be alternatively spliced. Note that the threshold for the exon score could be adjusted to the relative amount of signal in the data. Given that probeset k has been identified to have substantial between array variability, we propose to use the estimated random effects b_{ik} as an "array score" for identifying arrays in which the alternatively spliced exon is expressed. Tissues with an enrichment of probeset k are expected to have array scores greater than zero as they resisted shrinkage towards the overall gene level effects.

The parameters of the proposed mixed effects model are estimated within the Bayesian framework with vague proper priors since the full conditional posterior distributions for the parameters of interest are known. More information can be found in Van Moerbeke et al. (2017) and Van Moerbeke and Kasim (2018).

3 Preprocessing of the Microarray

3.1 Annotation File Creating

In order for the REIDS function to work an exon annotation file, a transcript annotation file and a junction annotation file are needed. These can be created by using the "HTA-2_0.na35.hg19.probeset.csv" file and the python scripts included in the package. Run the script either in the folder which contains "HTA-2_0.na35.hg19.probeset.csv" or alter the python script to include the path to the file.

The first python script will create exon and transcript annotations. The output files are respectively "HTA-2_0_ExonAnnotations.txt", "HTA-2_0_TranscriptAnnotations.txt", "LineIndexing_ExonAnnot.txt" and "LineIndexing_TrAnnot.txt". The former two files contain the annotations, the latter two files are indexing files which are used in the R code to prevent the loading of the entire files. The script is executed as:

```
python HTA-2_0_ExonAndTranscriptAnnotations.py
```

The second python script will create junction associations based on the exon annotations. The output files are respectively "HTA-2_0_JunAnnotations.txt" and "LineIndexing_JAnnot.txt". The script is executed as:

```
$ python HTA-2_0_JunAssociations.py
```

The first script will not take long, the second script will (a day). Please feel free to alter the code to speed things up or retrieve junction associations from the exon annotations in the R code directly (changes are needed in the JunctionAssesment function). Other mappings than Affymetrix' can be used as well if the files have the structure.

The "HTA-2_0_ExonAnnotations.txt" is:

```
TC01000001 JUC01000001 + EX01051517 3.0
TC01000001 JUC01000001 + EX01051518 5.0
TC01000001 JUC01000002 + EX01051516 3.0
TC01000001 JUC01000002 + EX01051515 5.0
....
TC01000001 PSR01000002 + EX01022607
TC01000001 PSR01000002 + EX01055113
TC01000001 PSR01000004 + EX01022608
TC01000001 PSR01000004 + EX01051517
```

The "HTA-2_0_TranscriptAnnotations.txt" is:

```
TC01000001 JUC01000001 + EX01051517 TR01023991
TC01000001 JUC01000001 + EX01051518 TR01023991
TC01000001 JUC01000002 + EX01051516 TR01023992
TC01000001 JUC01000002 + EX01051515 TR01023992
...
TC01000001 PSR01000002 + EX01055113 TR01009585
TC01000001 PSR01000002 + EX01022607 TR01021558
TC01000001 PSR01000002 + EX01022607 TR01023991
TC01000001 PSR01000002 + EX01022607 TR01015438
TC01000001 PSR01000002 + EX01022607 TR01023992
```

The "HTA-2_0_JunAnnotations.txt" is:

```
TC01000205 PSR01003404 JUC01001839 3
TC01000205 PSR01003405 JUC01001839 5
TC01000205 PSR01003405 JUC01001844 3
TC01000205 PSR01003407 JUC01001844 exclusion
TC01000205 PSR01003413 JUC01001829 3
TC01000205 PSR01003413 JUC01001833 3
TC01000205 PSR01003413 JUC01001835 3
```

```
TC01000205 PSR01003413 JUC01001840 3
TC01000205 PSR01003413 JUC01001842 3
TC01000205 PSR01003413 JUC01001843 3
```

The exon and transcript annotations can thus easily be replaced by, for example, Ensemble annotations.

3.2 Creating the .CDF file

The .CDF file necessary to process the array with the R package aroma.affymetrix is created by combining the .clf en .pgf file of the Affymetrix library. Credit goes to the arome.affymetrix maintainers and writes of the necessary function which I bundle here. The following steps are required to create the .CDF file:

1. Combine the .pgf and .clf information with "combineProbeInfo.pl":

   ```
   $ perl combineProbeInfo HTA-2_0.r3
   ```

   The output consists of two files: "HTA-2_0.r3.probeflat" and "HTA-2_0.r3.psr".

2. Run "ID_Name_TC.py" to combine probe set ID, probe set name and TC ID. This function retrieves information of "HTA-2_0.na35.hg19.probeset.csv" and the "HTA-2_0.r3.psr" file.

   ```
   $ python ID_Name_TC.py
   ```

   The output is one file: "PSID_Name_TCID.txt".

3. Create an empty file .flat file and run "addGeneId_2.pl" on the "HTA-2_0.r3.probeflat" file and "PSID_Name_TCID.txt" with the empty file as third argument.

   ```
   $ perl addGeneId_2 HTA-2_0.r3.probeflat PSID_Name_TCID.txt HTA-2_0.r3.flat
   ```

   The output will be written to "HTA-2_0.r3.flat".

4. Run "Flat2CDF.R" in R on the created outfile to obtain the .CDF file. The output file is "HTA-2_0,r3.cdf".

(Note: I had to rename "HTA-2_0,r3.cdf" to "HTA-2_0,r.cdf" in order for aroma.affymetrix to recognize the .cdf file and process the .CEL files)

4 Preprocessing of the Data

The processing of the data starts from the raw .CEL files with functions of the `aroma.affymetrix` (Bengtsson et al., 2008) package. Whether you are working on a HPC or on your laptop, to be able to use this package, you have to make sure to have the correct folder structure. In the current working directory two folders have to be set up. The first folder is a rawData folder. This folder contains another folder with the name of your data set. In our case this will be "TissueData". This "TissueData" folder should contain a last folder with as name the chipType used to produce the data (eg "HuEx-1_0-st-v2") and herein the .CEL files should be placed. The second folder is the "annotationData" folder. Herein a folder with name "chipTypes" should be made which contains folders for each chip type with the respective names. In the folder of each chip type the corresponding .cdf file should be saved. Figure 1 shows the structure of the directories with as working directory a folder named "REIDS". An R script can now be created in the current working directory such that the functions have acces to the created folders. The function used to perform the data processing is `DataProcessing` and is a wrapper of several functions of the `aroma.affymetrix` package which is imported by the `REIDS` package. In order to obtain the data to perform the REIDS model on, the raw .CEL files are background corrected with the rma background correction and normalization is performed with the quantile normalization. The data is returned as a data frame with one line per probe. There are thus multiple lines per probeset (one for each probe of the probeset).

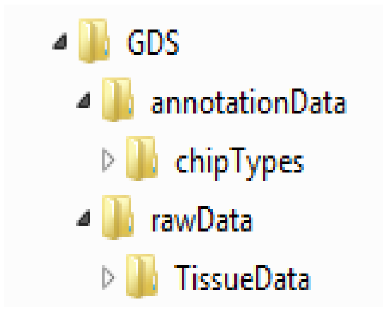

Figure 1: *Folder Structure for the* `aroma.affymetrix` *package*

The first colum contains the gene IDs and the second column the probe set IDs. All other columns contain the sample values of the probes. If requested, also a gene and exon level summarization is performed on the data with the rma summarization method. Further, the option is provided to perform the FIRMA model (Purdom et al., 2008) on the data as well. For an data the function specified in a regular R script is as follows:

```
> library(REIDS)
> DataProcessing(chipType="HTA-2_0",tags="*,r",Name="HTAData",
+                ExonSummarization=TRUE,GeneSummarization=TRUE,FIRMA=TRUE,
+                location="HTAData",verbose=TRUE)
```

The tags parameter refers to a tag that was added to the .cdf file. Make sure to have created a folder named "HTAData" in your working directory for the results to be saved. This is the loctaion in the function above. The function as formulated above will create four data frames. One with the backgouncorrected and normalized probe intensities, one with the gene level summarized values, one with the exon level summarized values and a last one with the values of the FIRMA model. If one has a HPC available, we can call upon the previous function via the DataProcessing.R and DataProcessing.pbs scripts available in the doc folder. Be carefull to fill in the requirements for your specific project. After completing the .pbs file, a qsub command on this batch file should get the function running.

If you do not have access to a HPC service, the next processing steps can be skipped and the data is a proper format for the REIDS model implemented in the `REIDSFunction` function. If you do have access to an HPC service and you wish to use it, some more processing of the data is necessary. The strength of using the HPC server is that calculations can be parallelized. In our case where we run the REIDS model gene-by-gene, the server needs a file in which every line correspond to one gene and contains all the information of that gene. This way the cluster can read several lines, etc several genes simultaneously and distribute them over multiple cores who perform the calculations independently. First, a pivot transformation will be conducted to convert the data into a file with one line per gene. All information concerning one gene is thus gathered into this line. The first column of the returned file contains the gene ID, the second column contains the exon IDs of all the exons of that gene. The third colum indicates the number of probes per exon, the fourth contains the values of those probes per sample and the last column contains the sample names.

```
> load("HTAData.RData")
> PivotTransformData(Data=HTAData,GeneID=HTAData$GeneID,ExonID=HTAData$ExonID,
+                savecsv=FALSE,Name="HTAData_Pivot",location="HTAData")
> GeneID=unique(as.character(HTAData$GeneID))
> GeneTable=data.frame("GeneID"=GeneID)
> save(GeneTable,file="GeneTable.RData")
```

A final file is created with the begin positions and lengths of the lines of the previous file. The function for this indexing of the lines is not an `R` function but a `python` function. The doc folder contains the `Line_Indexer.py` function and its corresponding .pbs file. The last line is where the function is called.

```
python Line_Indexer_Pivot.py --output_file HTAData_LineIndex.csv HTAData_Pivot.csv
```

After output_file the first name is the name to be given to the transformed file. The second file is the file to be transformed. Make sure that the function is run in the same folder where the file to be transformed is saved. The HPC server will now work from these begin positions, read a line from the first file and process the corresponding gene with the REIDS model. The second file is not necessary for the calculations of the HPC server but it makes life easier for the server as it does not need to read in all the information on all the genes.

Finally, we compute the DABG p-values with APT.

```
apt-probeset-summarize -a dabg -p HTA-2_0.r3.pgf -c HTA-2_0.r3.clf -b
HTA-2_0.r3.antigenomic.bgp-o ./APT_DABG Path/To/Folder/*.CEL"
```

The file "PSID_Name_TCID.txt" was created during the making of the .CDF file and contains a probe set ID, the probe set name and the corresponding TC ID in each line.

```
> ProbesetID_ProbesetName=read.table("PSID_Name_TCID.txt")
> DABG=read.table("APT_DABG/dabg.summary.txt",header=TRUE,sep="\t")
> #DABG in all samples
> Low=apply(DABG,1,function(x) all(x>=0.05))
> LowUIDs=DABG[,1][Low]
> LOW_Probesets_All=unique(ProbesetID_ProbesetName[which(
+ ProbesetID_ProbesetName[,1]%in%LOWUIDs),2])
> save(LOW_Probesets_All,file="Low_AllSamples.RData")
> #Per Group
> Low_GSamples=list()
> for(g in 1:length(Groups)){
+         Low=apply(DABG,1,function(x) all(x[Groups[[g]]]>=0.05))
+         LowUIDs=DABG[,1][Low]
+         LOW_Probesets_Samples=unique(ProbesetID_ProbesetName[which(
+         ProbesetID_ProbesetName[,1]%in%LOWUIDs),2])
+         Low_GSamples[[g]]=LOW_Probesets_Samples
+ }
> save(Low_GSamples,file="Low_GSamples.RData")
```

After this final function, the data is ready to be used for the REIDS model.

5 The REIDS Model

The data is now in an adequate format for the REIDS model. If you do not have acces to an HPC service, the `REIDSFunction` function can be performed on the data frame returned by the `DataProcessing` function. The `REIDSFunction` function consists of two subfunctions. The first is the `iniREIDS` which filters out non-informative probesets with the I/NI calls model (Kasim et al., 2010). The option is available whether the filtering should be performed or not. The second function is the `REIDSmodel_intern` and performs the actual REIDS model, possibly on the output of `iniREIDS`. A list with one element per gene is returned. Per gene, there is list with three elements if `iniREIDS` is performed. Otherwise, two elements are available per gene. The optional element per gene is a data frame with the exons and a logical value indicating whether the exon is informative or not with `TRUE` or `FALSE` respectively. The other two elements contain the exon scores and the array scores of the exons. The function is:

```
> load("GeneTable.RData")
> load("Low_AllSamples.RData")
> REIDSFunction(geneIDs=GeneTable,Name="HTA-2.0", Indices="HTAData_lineindex.csv",
+                 DataFile="HTAData_Pivot.csv",nsim=5000,informativeCalls=FALSE,
+                 rho=0.5,Low_AllSamples=Low_AllSamples,Location="Output")
>
>
```

The non-cluster version of the REIDS model is recommended to be used only on a small data set or a subset of a larger one. The `REIDSFunction_ClusterVersion` is an adaption of the REIDS model to be used on a HPC server. The cluster will parallelize the computations. This implies that several genes will be processed simultaneously. The process starts by reading a line of the file created by the python function `Line_Indexer`. This contains a begin position and length of that line of the file made by `PivotTransfornmation`. Next, the corresponing line will be looked for and read into the script. This line enholds all the information of one gene and with some processing, a small data frame of this gene is produced. This subset is then processed by the `REIDSFunction_ClusterVersion` function. For each processed gene a .RData file will be saved with the output of the REIDS model. We will combine these later. The entire procedure is coded in the REIDSFunction_ClusterVersion.R file in the doc folder of this pacakge. A corresponding .pbs file is also available. By submitting the file of the `Line_Indexer` function as the data and the REIDSFunction_ClusterVersion.pbs file as a batch file to a worker framework on the HPC server the function should be executed. The computation time depends from data set to data set. For the tissue data, 24 hours should be adequate. The long computational time is due a just a few genes that have a large number of exons. After 24 hours, five gene were not returned and therefore left out of the analysis.

Finally, the indiviudal gene outcomes of the `REIDSFunction_ClusterVersion` should be binded together into one larger data set. This is done by the `CreateOutput` function. It is necessary to provide a file of the gene IDs of which the outcomes were produced. The doc folder contains this file for the tissue data. The function will read the gene ID, look for the corresponding file of the REIDS model and add it to a list. Eventually it will return a list with one element per gene. The function can be run on the HPC service with the CreateOutput.R and CreateOutput.pbs files in the doc folder. A qsub command on the .pbs file gets the function started.

6 Analysis

If you are interested in the results of a particular gene and/or exon the `Search` function allows you to get that specific data out of the output. Given exon IDs, it is capable of retrieving the corresponding gene IDs as well in case the gene ID is not known beforehand. The function can be run on the REIDS ouput but also on more analyzed data frames as for example after performing a test between the samples.

```
> exonID <- c("PSR01003414","PSR02012920","PSR12000150")
> load("HTA-2.0_REIDS_Output.RData")
> Results=Search(WhatToLookFor=data.frame(ExonID=exonID),Data=REIDS_Output,
+                  AggregateResults=FALSE,NotFound=NULL)
```

In order to identify exons that are alternatively spliced, their exon scores and array scores should be investigated. The array scores can be tested between groups of interest of the samples. If the data consists of paired samples, the mean paired differences can be investigated. We have written a function `ASExons` that performs a simple t-test or F-test on the array scores. The p-values are adjusted for multiplicty. The function has the option to already filter out probesets that do not pass the exon score threshold and only consider those that do. If also a significance level is specified, non-significant p-values will be left out of the results. A data frame with one line per exon is returned. The columns contain the gene ID, the exon ID, the test statistic, a p-value and an adjusted p-value. If the groups are paired also the mean paired difference is given.

```
> load("HTA-2.0_REIDS_Output.RData")
> HTAData_1vs2=ASExons(Data=HTA-2.0,REIDS_Output,Exonthreshold=0.5,
+                  Groups=list(c(10:18),c(19:27)),paired=FALSE,
+                  significancelevel=0.05)
> save(HTAData_1vs2,file="HTAData_1vs2.RData")
```

Corresponding files for use on the HPC server are also here available in the doc folder.

We now add the junction information to discover more on the transcript composition.

```
> load("HTAData_1vs2.RData")
> load("HTAData.RData")
> load("Low_GSamples.RData")
> load("Low_AllSamples.RData")
> for(i in 3:ncol(HTAData)){
+         HTAData[,i]=as.numeric(as.character(HTAData[,i]))
+ }
> ASPSR_PSR=HTAData_1vs2[HTAData_1vs2$adj.p.value<0.05,]
> length(unique(ASPSR_PSR$ExonID))
> #ExonAnnotations
> EI=read.table("LineIndexing_ExonAnnot.txt",header=FALSE,sep="\t",
+                 stringsAsFactors=FALSE)
> #JunctionAnnotations
> JI=read.table("LineIndexing_JAnnot.txt",header=FALSE,sep="\t",
+                 stringsAsFactors=FALSE)
> #Transcript Annotations
> TrI=read.table("LineIndexing_TrAnnot.txt",header=FALSE,sep="\t",
+                 stringsAsFactors=FALSE)
> REIDS_JunctionAssesment(ASProbeSets=unique(ASPSR_PSR$ExonID),
+                 JAnnotI=JI,EAnnotI=EI,TrAnnotI=TrI,Data=HTAData,Groups=list(c(3,4,5),
+                 c(6,7,8)),Low_AllSamples,Low_GSamples,Plot=FALSE,Name="HTA-2.0")
>
```

The output file is .txt file with the following structure:

```
TC05001924 PSR05026575 Const ------0.24 EX05012493 -
TC06001355 PSR06016183 AS ----REIDS 0.60 EX06004488 Alternative First'
TC06001355 PSR06016180 AS -JUC06008083 -JUC06008083 REIDS -2.134 EX06024125 Alternative 3'
TC06001355 PSR06016179 AS -JUC06008083 -JUC06008083 REIDS -2.30 EX06024124 Alternative Last
TC06001387 PSR06016269 Const -----0.30 EX06011670 -
TC06001387 PSR06016268 Const JUC06008109 ----0.25EX06017092 -
TC06001387 PSR06016266 AS JUC06008109 ---REIDS -1.25 EX06026585 Alternative Last
```

Alternatively spliced exons can be selected from the file. Further, information regarding the junctions, AS type, fold change and exon annotation is available as well.

Information of the probe sets can be visualized by the means of plots. ExpressionLevelPlot.R plots a specific probe set with its probe set intensities, summarized exon level and gene level values.

```
> load("HTAData.RData")
> load("ExonLevel.RData")
> load("GeneLevel.RData")
> ExpressionLevelPlot(GeneID="TC12000010",ExonID="PSR12000150",Data=HTAData,
+                 GeneLevelData=GeneLevel,ExonLevelData=ExonLevel,groups=
+                 list(c(1:3),c(4:6)),ylabel="",title="PSR12000150")
>
```

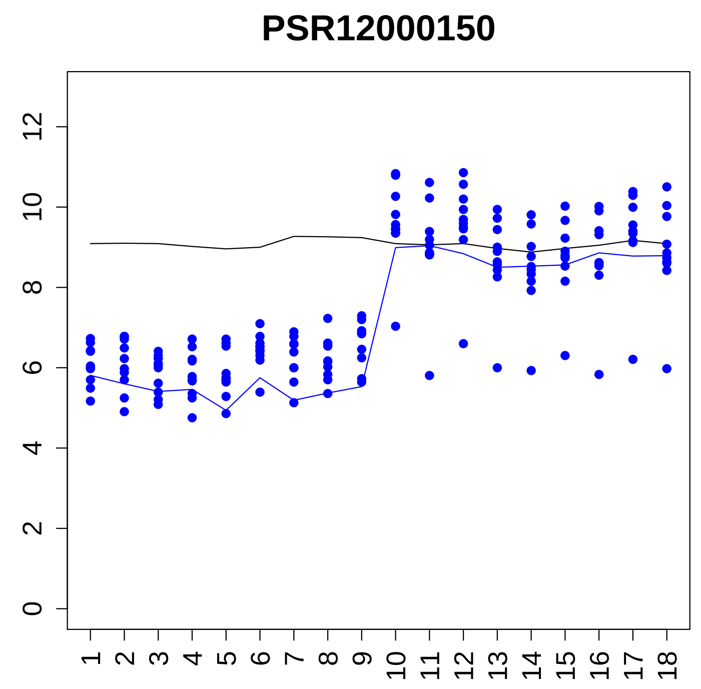

Figure 2: *The observed probe intensities of PSR12000150relative to the summarized gene level values of TC12000010. The black and blue lines indicate the mean profiles of the gene and exon level data respectively. The blue points show the probe level data.*

Transcript isoform composition can be retrieved with TranscriptsPlot.R. We show the composition of "TC12000010" and highlight PSR12000150. Note the subfix of ".hg.1 after the probe set name. The addition is necessary since this is the full probe set name in the Affymetrix based positions and transcriptinfo files.

```
> load("HTAData.RData")
> load("ExonLevel.RData")
> library(data.table)
> transcript.clusters.NetAffx.36<-fread("HTA-2_0.na35.2.hg19.transcript.csv",
+                 skip=19,header=TRUE,sep=",")
> transcript.clusters.NetAffx.36=as.data.frame(transcript.clusters.NetAffx.36)
> transcript.clusters.NetAffx.36=transcript.clusters.NetAffx.36[,c(1,4,8)]
> save(transcript.clusters.NetAffx.36,file="transcript.clusters.NetAffx.36.RData")
> #load("transcript.clusters.NetAffx.36.RData")
>
> probesets.NetAffx.36 <- fread("HTA-2_0.na35.hg19.probeset.csv",
+                 skip=19,header=TRUE,sep=",")
> probesets.NetAffx.36 =as.data.frame(probesets.NetAffx.36)
> positions_36 <- probesets.NetAffx.36[probesets.NetAffx.36[,13] == "main",c(1,2,4,5,6,7)]
> save(positions_36,file="positions_36.RData")
> #load("positions_36.RData")
>
>
> TranscriptsPlot(trans="TC12000010", positions=positions_36,
+                 transcriptinfo=transcript.clusters.NetAffx.36,
+                 display.probesets=TRUE,Data=ExonLevel,groups=list(c(1:3),c(4:6)),
+                 Start=NULL,Stop=NULL,Highlight="PSR12000150")
>
```

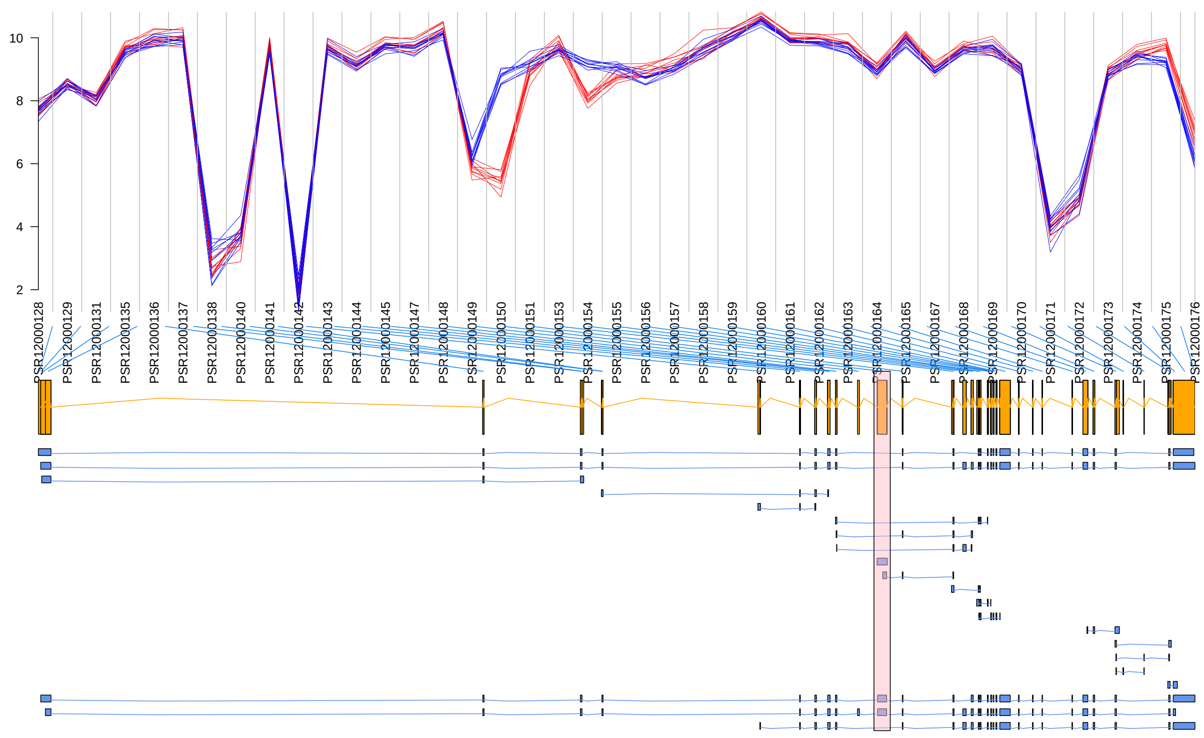

Figure 3: *The isoform composition of transcript cluster TC12000010 (WNK1). Probe set PSR12000150 is highlighted as an alternative last event exon. The red lines represent the probe set expression levels of the SCR samples. The blue lines show the probe set expression levels for the siRNA samples.*

The linking of the probe sets can be visualized with arcdiagrams. The code is retrieved from the R package arcdiagrams available via GitHub (https://github.com/gastonstat/arcdiagram). The plot for this function can be produced with the internal function of the REIDS_JunctionAssesment.R function with the Plot parameter set to TRUE: the JunInfo function.

```
> load("Data/HTAData_1vs2.RData")
> load("Data/HTAData.RData")
> load("Low_GSamples.RData")
> load("Low_AllSamples.RData")
> for(i in 3:ncol(HTAData_RASA)){
+         HTAData_RASA[,i]=as.numeric(as.character(HTAData_RASA[,i]))
+ }
> ExonScoreFilter=HTAData_LiverVSMuscle[HTAData_LiverVSMuscle$X50.>=0.50,]
> ASPSR_PSR=ExonScoreFilter[ExonScoreFilter$adj.p.value<0.05,]
> length(unique(ASPSR_PSR$ExonID))
> #ExonAnnotations
> EI=read.table("LineIndexing_ExonAnnot.txt",header=FALSE,sep="\t",
+                 stringsAsFactors=FALSE)
> #JunctionAnnotations
> JI=read.table("LineIndexing_JAnnot.txt",header=FALSE,sep="\t",
+                 stringsAsFactors=FALSE)
> #Transcript Annotations
> TrI=read.table("LineIndexing_TrAnnot.txt",header=FALSE,sep="\t",
+                 stringsAsFactors=FALSE)
> JunInfo(x="TC12000010",ASPSR=unique(ASPSR_PSR$ExonID),JLines=JI[which(JI[,1]==x),],
+                 TrLines=TrI[which(TrI[,1]==x),],ELines=I[which(EI[,1]==x),],
+                 DataS=HTADataData[which(as.character(HTAData[,1])==x),],
+                 Groups=list(c(1:3),c(4:6)),Low_ALLSamples=Low_ALLSamples,Low_GSamples=Low_GSampl
```

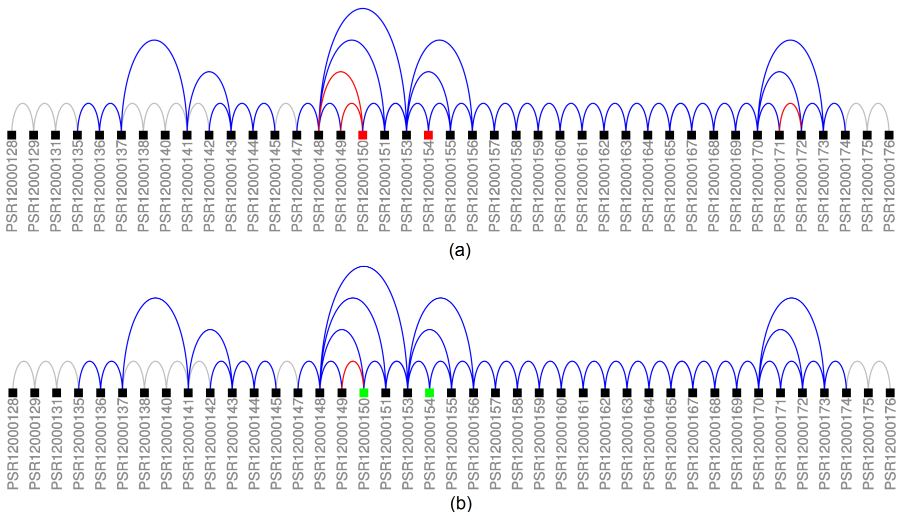

Figure 4: *Illustration of transcript cluster TC12000010 (WNK1). Black probe sets were identified as constituitive while coloured probe sets were identified as alternatively spliced. Green colour indicates an enrichment of a probe set while red colour denotes a depletion. Junctions which are DABG are shown in blue while depleted junctions are again shown in red. Panel (a) shows the probe sets for the SCR samples and panel (b) for the siRNA samples.*

References

Bengtsson, H., Irizarry, R., Carvalho, B., and Speed, T. (2008), "Estimation and assessment of raw copy numbers at the single locus level." *Bioinformatics*, 24, 759–767.

Kasim, A., Lin, D., Van Sanden, S., Clevert, D., Bijnens, L., Goehlmann, H. W. H., Amaratunga, D., Hochreiter, S., Shkedy, Z., and Talloen, W. (2010), "Informative or noninformative calls for gene expression: a latent variable approach." *Stat Appl Genet Mol Biol*, 9, Article4.

Purdom, E., Simpson, K. M., Robinson, M. D., Conboy, J. G., Lapuk, A. V., and Speed, T. P. (2008), "FIRMA: a method for detection of alternative splicing from exon array data," *Bioinformatics*, 24, 1707–1714.

Van Moerbeke, M., Kasim, A., Talloen, W., Reumers, J., , Göhlmann, H. W. H., and Shkedy, Z. (2017), "A Random Effectiveects Model for the Identifcation of Differenterential Splicing (REIDS) Using Exon and HTA Arrays," *BMC Bioinformatics*, 18, 273.

Van Moerbeke, M. and Kasim, A.and Shkedy, Z. (2018), "The Usage of Exon-Exon Splice Junctions for the Detection of Alternative Splicing using the REIDS model," *Unde review in Scientific Reports*, xx, xx.

7 Software used

- R Under development (unstable) (2018-01-28 r74175), `x86_64-w64-mingw32`
- Locale: `LC_COLLATE=C`, `LC_CTYPE=Dutch_Belgium.1252`, `LC_MONETARY=Dutch_Belgium.1252`, `LC_NUMERIC=C`, `LC_TIME=Dutch_Belgium.1252`
- Running under: `Windows 7 x64 (build 7601) Service Pack 1`
- Matrix products: default
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Loaded via a namespace (and not attached): compiler 3.5.0, tools 3.5.0