ExCluster package: Robust detection of differentially spliced genes in RNA-seq data

R. Matthew Tanner, William L. Stanford, and Theodore J. Perkins

2 October 2018

Contents

1 Introduction

This vignette will detail the basic usage of the ExCluster package, both quickly and step-by-step. IMPORTANT NOTE: ExCluster will not function with a Windows operating system, and will only install on linux and MacOSX machines. Unfortunately, this package requires Rsubread, which is not availble for Windows.

Before proteins may be produced from genes in our DNA, those genes are first transcribed into pre-messenger RNA (pre-mRNA). This pre-mRNA contains introns and exons, from which introns must be removed to produce mature mRNA; this process is called mRNA splicing. However, not all exons need be combined in the mature mRNA – some exons may be spliced out in different combinations. This process is known as alternative splicing, and it allows a single gene encode multiple protein coding transcripts, with

roughly 20,000 protein coding genes producing nearly 200,000 protein coding transcripts.

Different cell types within a given organism may express different levels of mRNA transcripts, and may even express entirely different mRNA transcripts. When two different biological conditions, such as two cell types or two tissues, display differential transcript expression patterns, the genes encoding those transcripts are said to be differentially spliced. Indeed, differential regulation of mRNA splicing plays crucial roles in biology, such as regulating organism development, specifying tissue identity, and maintaining cell homeostasis. Aberrant regulation of mRNA splicing has also been observed in many cancers. Thus, having the proper tools to detect differentially spliced genes in RNA-seq datasets is crucial to studying these facets of biology with high-throughput RNA-seq data.

The ExCluster package allows users to detect differentially spliced genes between two conditions of RNA-seq data, and also allows for the visualization of differential exon expression within significantly differentially spliced genes. ExCluster requires at least two biological replicates per condition, and requires exactly two conditions per analysis. ExCluster has been designed to more rebustly detect differentially spliced genes than do previous established differential splicing tools. ExCluster also more accurately tunes statistics, to reduce the number of false positives when using the typical 0.05 FDR cutoff.

2 Quick ExCluster analysis pipeline

This section assumes some familiarity with R and programming in general, and has much less explanation. If you are confused, please read the detailed step-by-step guide for a basic ExCluster analysis. Also, you can refer to the ExCluster manual, which contains details on ExCluster functions and their inputs.

To start, we run `GFF_convert` on our GTF file, saving the GFF file, both of which require file paths to be set to variables:

```
> library(ExCluster)
> GTF_file <- "/path/to/file.gtf"
> GFF_file <- "/path/to/file.ExClust.gff
> GFF <- GFF_convert(GTF.File=GTF_file, GFF.File=GFF_file)
```

We now have our GFF annotation data frame written out, and stored in the 'GFF' variable. We can now count reads per BAM file – but the described process requires only your BAM files of interest in the bamDir and sub-directories. We also name bam files with sampleNames. Then we run `processCounts`:

```
> bamDir <- "/path/to/bamDir/"
> bamFiles <- list.files(bamDir,full.names=TRUE, recursive=TRUE,
```

```
   pattern="*.bam")
> sampleNames <- c("hESC_cond1_rep1", "hESC_cond1_rep2", "hESC_cond1_rep3",
   "iPSC_cond2_rep1", "iPSC_cond2_rep2", "iPSC_cond2_rep3")
> normCounts <- processCounts(bam.Files=bamFiles, sample.Names=sampleNames,
   annot.GFF=GFF, paired.Reads=TRUE)
```

Note: if you are counting paired-end reads, we use paired.Reads=TRUE. If some or all of your samples are single-end, use paired.Reads=FALSE (default if unset is FALSE). `processCounts` counts reads per exon bin from each BAM file and normalizes them for library size differences – we assign this output to the normCounts variable.

Now we can set up to run `ExCluster`, which requires condition numbers, and we should give it an outDir and fileName to write files to. We also want to plot our significant results, so we use plot.Results=TRUE. We do all of this as follows:

```
> condNums <- c(1,1,1,2,2,2)
> outDir <- "/path/to/ExClustResults/"
> fileName <- "hESC_vs_iPSC_ExClust_Results"
> ExClustResults <- ExCluster(exon.Counts=normCounts, cond.Nums=condNums,
   annot.GFF=GFF, out.Dir=outDir, result.Filename=fileName,
   combine.Exons=TRUE, plot.Results=TRUE)
```

This will run for several hours, and will output the results to the ExClustResults variable, as well as write those same results to outDir/hESC_vs_iPSC_ExClust_Results.txt. Additionally, all significantly differentially expressed genes will have their exon bin log2FC means and variances plotted in a sub-folder of outDir.

The only additional note is the combine.Exons=TRUE argument. Some exons are co-expressed always within a given gene, and can therefore be combined into a 'super-exon'. This increases the power of ExCluster and reduces computation time greatly. However, it may miss aberrant splicing events. If you suspect abberant splicing, you can use combine.Exons=FALSE (also FALSE by default if unset), and run the longer multhour ExCluster analysis.

This concludes the quick run explanation.

3 Detailed ExCluster pipeline explanation

This Vignette assumes that you have successfully installed ExCluster in R, using Bioconductor. If you have not yet installed ExCluster with Bioconductor, please try to run the following:

```
> chooseCRANmirror()
> install.packages("BiocManager")
> BiocManager::install("ExCluster")
```

Please note that BiocManager (and ExCluster) will require R 3.5.1 or newer. If you have trouble downloading the BiocManger package, try selecting a different mirror or contact Bioconductor. If only ExCluster has issues, please contact the package maintainer. Please also remember that ExCluster will not install properly on Windows, as it relies on the Rsubread package.

With ExCluster installed, we must now load the R package as follows:

```
> library(ExCluster)
```

The next step in the ExCluster pipeline is to take a GTF genome annotation file, and flatten it into a GFF file, containing non-overlapping exon bins. This is done using the function `GFF_convert`. GTF files may be obtained from sources such as Ensembl, GENCODE, RefSeq, and UCSC. We strongly recommend using GENCODE GTF files, as they were used to test and develop ExCluster. Within the Stanford and Perkins labs, we have generally had more successful bioinformatics analyses with GENCODE annotations.

Before we run `GFF_convert`, we must specify the GTF file location, and we should also specify a GFF file location to write the output to. Saving GFF files is not necessary, as the GFF annotations can be carried through the pipeline if conducted in a single session. However, saving GFF files is good practice, incase data needs to be re-analyzed later on. Once a GFF file has been created, it may be re-used in future analyses.

We therefore set up variables to run the `GFF_convert` function as follows:

```
> GTF_file <- "/path/to/file.gff"
> GFF_file <- "/path/to/file.ExClust.gff
```

Experienced programmers and R users will understand what the above placeholder filepaths mean. If you are confused about the above example, I will use a more specific example: My username on MacOS is 'matanner', and my GFF file is called 'gencode.v23.annotation.gff'. The full path to my GTF file is:

/Users/matanner/Documents/gencode.v23.annotation.gtf

Now that our GTF_file and GFF_file variables are correctly set, we can run `GFF_convert`:

```
> GFF <- GFF_convert(GTF.File=GTF_file, GFF.File=GFF_file)
```

Once this last command is run, it will both assign the GFF annotation data frame to the GFF variable, and write out your GFF data frame to the GFF_file location. If you do not wish to write out GFF files, simply do not specify the GFF.File argument, which by default does not write out data.

We can now move forward and set up our exon bin read counting with the `processCounts` function. For this we need BAM files with two different conditions, and at least two biological replicates per condition. If you are new to analyzing RNA-seq data, and you only have raw fastq files, you will need to align your fastq files to obtain BAM files. This can be done with many programs, such as TopHat2, HISAT2, STAR, and BWA, among others. We recommend HISAT2 as an aligner, as ExCluster was tested with HISAT2 output – although HISAT2 only writes SAM files, which must be converted to BAM files with SAMtools.

Assuming you have your RNA-seq data aligned and in BAM file format, it is helpful if the exact BAM files you want to analyze are in a single folder (different sub-folders are okay). The most important point for this example is to have only your bam files of interest in your bamDir and its sub-folders. Assuming this is the case, we specify our BAM file paths into the 'bamFiles' variable, as follows:

```
> bamDir <- "/path/to/bamDir/"
> bamFiles <- list.files(bamDir, full.names=TRUE, recursive=TRUE,
   pattern="*.bam")
> print(bamFiles)
```

The last command should list your bamFiles if the process worked correctly. You should see exactly your BAM file paths of interest. If you have too many BAM files, or are missing BAM files, you may need to manually specify the paths in your bamFiles array. Please consult section 4 (after this section) for alternate ways to specify the bamFiles variable. If you see no BAM file paths printed out, you need to re-check and ensure you provided the exactly correct path to the folder containing your BAM files (folder and file paths are case sensitive).

With your BAMfiles correctly specified, we must now assign sample names to each bam file, in precise order. Sometimes BAM files will be named 'accepted_hits.bam', such as outputs from TopHat2. For this reason we specify exactly what we want each sample (BAM file) to be called in the exon bin count table. We specify sample names in the sampleNames variable. These names should not contain spaces – instead of spaces, use underscores _ . It also helps to name your samples by condition name and number, and replicate number. For example, if we are comparing human embryonic stem cells (hESCs) to induced pluripotent stem cells (iPSCs), and we have 3 replicates per condition, we can name our samples as follows:

```
> sampleNames <- c("hESC_cond1_rep1", "hESC_cond1_rep2",
   "hESC_cond1_rep3", "iPSC_cond2_rep1", "iPSC_cond2_rep2",
   "iPSC_cond2_rep3")
```

This naming convention is helpful, because it clearly indicates both the name of the condition, and which condition is which number (which determines the direction of exon fold changes). It also unambiguously identifies replicates. Again, we must verify that this order of sample names matches the order of the bamFiles array, or we will have improperly named samples and thus an improper analysis.

With our setup complete, we can now run the `processCounts` function – let us assume we are counting paired-end read data, so we also specify paired.Reads=TRUE:

```
> normCounts <- processCounts(bam.Files=bamFiles,
   sample.Names=sampleNames, annot.GFF=GFF, paired.Reads=TRUE)
```

Again, you will notice that we specified the paired.Reads=TRUE argument, which is important when conducting ExCluster analyses on paired-end data. Unfortunately, ExCluster cannot currently accomodate mixed sample types. In other words, if some of your BAM files are paired-end, and some are single-end, we cannot specify different counting types. If some or all of your samples are single-end, you can set the paired.Reads=FALSE argument. This is OK, as processCounts also normalizes library sizes, which should factor out much of the library size differences between the paired-end and single-end data.

The result of the above R command should give you a library size normalized exon bin count matrix, which is stored in the normCounts variable. To check this data frame, you should use ncols() and nrows() and have a number of columns equal to your number of BAM files, and roughly 500-600 thousand rows for human data. You should also sum your columns with apply(), and see millions of reads per column. These checks can be checked as follows:

```
> ncols(normCounts)
> nrows(normCounts)
> apply(normCounts,2,sum)
```

Assuming your normCounts read count data frame has counted properly, you may now proceed with the main function of this package, ExCluster! As before, we must set up some variables before we run our function. We must assign unique condition identifiers to the condNums variable, which correspond in exact order to your condition 1 and condition 2 samples in normCounts and bamFiles. For example, if your first 3 samples are condition 1, and your last 3 samples are condition 2, you could set up condNums as follows:

```
> condNums <- c(1,1,1,2,2,2)
```

If you have alternating samples in your normCounts data frame, you would use the following command:

```
> condNums <- c(1,2,1,2,1,2)
```

If you have more than two conditions, please only run `ExCluster` with 2 conditions at once. To compare multiple conditions, you will have to run `ExCluster` several times. Once you have ensured the correct columns in normCounts have been given the correct condition numbers in condNums, we must specify a directory and file name to write out our `ExCluster` results. Writing out results is not absolutely necessary, but it is strongly advised. The `ExCluster` function can take hours to run, and therefore failing to save data could result in data loss, and the need to re-run the multi-hour analysis again. We specify an output folder and file name as follows:

```
> outDir <- "/Users/username/path/to/ExClustResults/"
> fileName <- "hESC_vs_iPSC_ExClust_Results"
```

Note, when you specify the fileName variable string, you need not include the file extension. By default, `ExCluster` will add '.txt' as a file extension, and write a tab delimited file. We can now run the `ExCluster` function as follows, assigning the ExCluster results to the ExClustResults variable:

```
> ExClustResults <- ExCluster(exon.Counts=normCounts, cond.Nums=condNums,
   annot.GFF=GFF, out.Dir=outDir, result.Filename=fileName,
   combine.Exons=TRUE, plot.Results=TRUE)
```

You will notice that we specified two extra parameters: combine.Exons and plot.Results. Both arguments must be set as logical TRUE or FALSE values. Anything else will likely throw an error, and ExCluster may fail. If they are not set when running `ExCluster`, both combine.Exons and plot.Resuts default to FALSE.

Very briefly, some exons are always co-expressed together in transcripts. These exons can therefore be combined into 'super-exons', based on GFF transcript definitions. This increases the read depth of these exons and reduces their variance, giving more powerful statistical resolution. It also decreases the number of exons bins analyzed per gene, which greatly cuts down on computation time. HOWEVER, this may cause aberrant splicing events to become missed. If you suspect aberrant splicing may occur in your biological paradigm, MAKE SURE combine.Exons=FALSE! However, if you are conducting a standard analysis where no aberrant splicing should occur, combine.Exons=TRUE will result

in greater power and faster computation completion.

The `plotExonlog2FC` function may be called from within `ExCluster` to save extra lines of commands to the pipeline – this is done by setting plot.Results=TRUE. Here, all genes equal to or less than an FDR cutoff will be called significant, and plotted in a sub-folder of the ExCluster results directory (outDir). By default the FDR.cutoff = 0.05, however you may specify as low as FDR.cutoff=0.01 and as high as FDR.cutoff=0.2 as an extra argument when running `ExCluster`. You may also run `plotExonlog2FC` on its own. See the manual for more details.

This concludes the step-by-step, detailed explanation of the ExCluster pipeline.

4 Custom bamFiles variable assignment

Assuming that you cannot have all of your BAM files exclusively within a given folder, you can manually set your bamFiles array by entering the paths to your BAM files one by one. This is done as follows:

```
> bamFiles <- c("/path/to/cond1_rep1.bam",
   "/path/to/cond1_rep2.bam",
   "/path/to/cond1_rep3.bam",
   "/path/to/cond2_rep1.bam",
   "/path/to/cond2_rep2.bam",
   "/path/to/cond2_rep3.bam")
```

Just as a reminder, if my username is 'matanner' on a Mac OS system, the full path to my first BAM file might be "/Users/matanner/Documents/iPSC_RNAseq/hESC_cond1_rep2.bam

In this way, you must specify the exact filepath of each BAM file, and ensure that file path is valid. Although slow, this method offers greater control over your analysis.

5 Running ExCluster "toy dataset" examples

The ExCluster package contains a small dataset for 3 genes, with 4 BAM files split evenly between two conditions. This toy dataset also includes sub-sampled GTF and GFF files, as well as various ExCluster results objects. If you are having trouble executing ExCluster functions, or simply wish to test the functionality of the package, you may use the following examples to test each function:

First we test the GFF_convert function, which takes a GTF file input and flattens it into a GFF3 file. We run the code as follows:

```
> library(ExCluster)
> # load the sub-sampled GTF file path from the ExCluster package
> GTF_file <- system.file("extdata","sub_gen.v23.gtf", package = "ExCluster")
> # now run GTF\_file without assigning a GFF\_file to write out, assigning the results to the GFF object
> GFF <- GFF_convert(GTF.File=GTF_file)
```

Next we can run the processCounts function on the small BAM file dataset as follows:

```
> # specify the path to the ExCluster package
> ExClust_Path <- system.file(package="ExCluster")
> # now find the bam files within that folder
> bamFiles <- list.files(ExClust_Path,recursive=TRUE,pattern="*.bam",full.names=TRUE)
> # assign sample names (only 2 replicates per condition in this example)
> sampleNames <- c("iPSC_cond1_rep1","iPSC_cond1_rep2","iPSC_cond2_rep1","iPSC_cond2_rep2")
> # now run processCounts, with paired.Reads=TRUE because we are counting paired-end data
> normCounts <- processCounts(bam.Files=bamFiles, sample.Names=sampleNames, annot.GFF=GFF, paired.Reads=TRUE)

        ==========     _____ _    _ ____  _____  ______          _____
        =====         / ____| |  | |  _ \|  __ \|  ____|   /\   |  __ \
          =====      | (___ | |  | | |_) | |__) | |__     /  \  | |  | |
            ====      \___ \| |  | |  _ <|  _  /|  __|   / /\ \ | |  | |
              ====    ____) | |__| | |_) | | \ \| |____ / ____ \| |__| |
        ==========   |_____/ \____/|____/|_|  \_\______/_/    \_\_____/
       Rsubread 2.4.0

//========================== featureCounts setting ===========================\\
||                                                                            ||
||             Input files : 4 BAM files                                      ||
||                                                                            ||
||                           iPSC_batch1_20_cond1_rep1.bam                    ||
||                           iPSC_batch1_20_cond1_rep2.bam                    ||
||                           iPSC_batch1_20_cond2_rep1.bam                    ||
||                           iPSC_batch1_20_cond2_rep2.bam                    ||
||                                                                            ||
||              Paired-end : yes                                              ||
||        Count read pairs : yes                                              ||
||              Annotation : R data.frame                                     ||
||      Dir for temp files : /tmp/Rtmp3paSN7                                  ||
||                 Threads : 1                                                ||
||                   Level : meta-feature level                               ||
||      Multimapping reads : counted                                          ||
|| Multi-overlapping reads : counted                                          ||
||   Min overlapping bases : 1                                                ||
||                                                                            ||
\\============================================================================//

//================================= Running ==================================\\
||                                                                            ||
|| Load annotation file .Rsubread_UserProvidedAnnotation_pid27350 ...         ||
||    Features : 23                                                           ||
||    Meta-features : 23                                                      ||
||    Chromosomes/contigs : 1                                                 ||
||                                                                            ||
|| Process BAM file iPSC_batch1_20_cond1_rep1.bam...                          ||
||    Paired-end reads are included.                                          ||
||    Total alignments : 920                                                  ||
||    Successfully assigned alignments : 542 (58.9%)                          ||
||    Running time : 0.01 minutes                                             ||
||                                                                            ||
```

```
|| Process BAM file iPSC_batch1_20_cond1_rep2.bam...                          ||
||    Paired-end reads are included.                                          ||
||    Total alignments : 619                                                  ||
||    Successfully assigned alignments : 299 (48.3%)                          ||
||    Running time : 0.00 minutes                                             ||
||                                                                            ||
|| Process BAM file iPSC_batch1_20_cond2_rep1.bam...                          ||
||    Paired-end reads are included.                                          ||
||    Total alignments : 960                                                  ||
||    Successfully assigned alignments : 386 (40.2%)                          ||
||    Running time : 0.00 minutes                                             ||
||                                                                            ||
|| Process BAM file iPSC_batch1_20_cond2_rep2.bam...                          ||
||    Paired-end reads are included.                                          ||
||    Total alignments : 920                                                  ||
||    Successfully assigned alignments : 342 (37.2%)                          ||
||    Running time : 0.00 minutes                                             ||
||                                                                            ||
|| Write the final count table.                                               ||
|| Write the read assignment summary.                                         ||
||                                                                            ||
\\============================================================================//
```

The main function of the package, ExCluster, can be tested with the following code:

```
> # assign condition numbers to your samples (we have 4 samples, 2 replicates per condition)
> condNums <- c(1,1,2,2)
> # now we run ExCluster, assigning its output to the ExClustResults variable
> # we are not writing out the ExClustResults table, nor are we plotting exons
> # we also use combine.Exons=FALSE to discover one 'significant' gene for example plot purposes
> ExClust_Results <- ExCluster(exon.Counts=normCounts,cond.Nums=condNums,annot.GFF=GFF, combine.Exons=FALSE)
```

We can test the exon log2FC plotting function plotExonlog2FC as follows:

```
> # now we must specify a directory to write images to
> # here we use tempdir, but you may substitute another folder path if you wish
> outDir <- paste(tempdir(),"/Images/",sep="")
> # now we can run our exon log2FC plotting function
> plotExonlog2FC(results.Data=ExClust_Results, out.Dir=outDir, plot.Type="PNG")
```

6 Formatting ExCluster results as a GRanges object

In addition to the above methods, users may also use GTF files imported by rtracklayer in conjuction with the rtracklayerGTFtoGFF function, to produce GFF flattened files. This function calls the GFF_convert function from within itself, and must be given the results of an rtracklayer import() function.

Users may convert the results ExCluster to a GRanges objects, as said results contain genomic coordinates for each exon bin. This may allow for easier manipulation of these datastructures outside of the standard ExCluster pipeline described in this vignette. Please note that the plotExonlog2FC function does not accept GRanges objects, instead

expecting data in the format previously described in this vignette.

Converting ExCluster results to GRanges format can be done as follows:

```
> GRanges.ExClustResults <- GRangesFromExClustResults(results.ExClust=ExClust_Results)
```

7 Authors

R. Matthew Tanner, William L. Stanford, and Theodore J. Perkins
CMM Department, Faculty of Medicine, University of Ottawa
Ottawa Hospital Research Institute
501 Smyth Box 511, Ottawa ON K1H 8L6
Canada