Identify differential APA usage from RNA-seq alignments

Elena Grassi
Department of Molecular Biotechnologies and Health Sciences,
MBC, University of Turin, Italy

roar version 1.6.1 (Last revision 2014-09-24)

Abstract

This vignette describes how to use the Bioconductor package *roar* to detect preferential usage of shorter isoforms via alternative poly-adenylation from RNA-seq data. The approach is based on Fisher test to detect disequilibriums in the number of reads falling over the 3'UTRs when comparing two biological conditions. The name roar means "ratio of a ratio": counts and fragments lengths are used to calculate the prevalence of the short isoform over the long one in both conditions, therefore the ratio of these ratios represents the relative "shortening" (or lengthening) in one condition with respect to the other. As input, *roar* uses alignments files for the two conditions and coordinates of the 3'UTRs with alternative polyadenylation sites. Here, the method is demonstrated on the data from the package *RNAseqData.HNRNPC.bam.chr14*. To cite this software, please refer to `citation("roar")`.

Contents

1 Introduction

The alternative polyadenylation mechanism at the basis of the existence of short and long isoforms of the same gene is reviewed in Elkon et al [**?**]. For the relevance of this phenomenon to development and cancer see [**?**, **?**, **?**].

2 Input data

A roar analysis starts from some alignments, obtained via standard RNAseq experiments and data processing. It is possible to build a *RoarDataset* object giving its constructor two lists of bam files obtained from samples of the two conditions to be compared or to use another constructor that takes directly two lists of *GAlignments*.

2.1 Alternative polyadenylation annotations

The other information needed to build a *RoarDataset* object are 3'UTRs coordinates (with canonical and alternative polyadenylation sites) for the genes that one wants to analyze - these could be given using a gtf file or a GRanges object.

The gtf file should have an attribute (*metadata column* for the *GRanges* object) called "gene_id" that ends with "_PRE" or "_POST" to address respectively the short and the long isoforms.

An element in the annotation is considered "PRE" (i.e. common to the short and long isoform of the transcript) if its gene_id ends with "_PRE". If it ends with "_POST" it is considered the portion present only in the long isoform. The prefix of gene_id should be a unique identifier for the gene and each identifier has to be associated with only one "_PRE" and one "_POST", leading to two genomic regions associated to each gene_id.

In the package we added a file (examples/apa.gtf) that follows these directives: it is build upon the hg19 human genome release using PolyA_DB ([**?**]) version 2 to obtain coordinates of alternative polyadenylation (from now on addressed as APA) sites. Every gene with at least one APA site is considered; their longest transcript is used to get the "classical" polyadenylation syte (canonical end of the longest transcript stored in UCSC) and the most proximal (with respect to the TSS, that is the farther to the canonical end of the transcript; but preferring sites still in the 3'UTR) reported APA site found in PolyA_DB is used to define the end of the short isoform. Therefore we define the "PRE" portion as the stretch of DNA starting from the beginning of the exon containing (or proximal to) the APA site and ending at the site position, while the POST portion starts there and ends at the canonical transcript end. Using only the nearest exon to the APA site and not the entire transcript to define the short isoform avoids spurious signals deriving from other alternative splicing events.

To summarize the package requirements are: every entry in the gtf should have an attribute called "gene_id" formed by a prefix representing unique gene identifiers and a suffix. The suffix defines if the given coordinates refer to the portion of this gene common between the short and the long isoform ("_PRE") or to the portion pertaining only to the long isoform ("_POST"). Every gene identifier must appear two (and only two) times in the gtf, one with the suffix "_PRE" and one with "_POST". An example of how PRE and POST are defined is reported in Figure **??**.

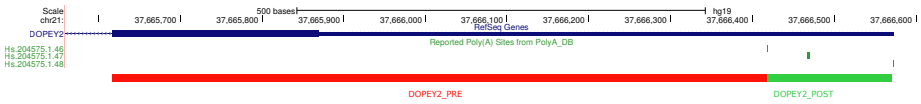

Figure 1: An example of PRE/POST portions definition

3 Analysis steps

The suggestion is to use one of the two wrapper scripts (*roarWrapper* or *roarWrapper_chrBychr*) or at least follow their tracks. The principal steps of the analysis are performed by different methods that receive a *RoarDataset* object as an argument and returns it with the needed step performed. It is also possible to call directly the methods to get the results (eg. *totalResults*), in this way all the preceding steps will be performed automatically.

Loading and handling alignments data require a lot of memory: if you have small datasets (up until 2.5Gb, 58.5 million reads with a length of 50 bp) 4 Gb of RAM will be more than sufficient (peak memory usage, gotten via the ps command with the keyword rss, when comparing two samples of 2.5Gb: 1.3Gb) and you can use the *roarWrapper* script. For bigger datasets (eg. 20Gb, 413 million reads with a length of 101bp) you will have to split your alignments in small chunks: *roarWrapper_chrBychr* works on single chromosomes.

Please note than when working in this way FPKM values returned by *fpkmResults* will be different from those obtained with the analysis performed using a single *RoarDataset* on the whole alignments - in this case you can extract counts with *countResults* and knit together real FPKM values afterwards, if you need them (*roarWrapper_chrBychr* does this).

In the following sections we will show an example of this package usage on the *RNAseqData.HNRNPC.bam.chr14* package data and the annotation given in the package itself.

3.1 Creating a RoarDataset object

The analysis begins by creating a *RoarDataset* object that holds the alignment data (4 HNRNPC ko samples and 3 control samples) and the annotation regarding 3'UTRs coordinates.

```
> library(roar)
> library(rtracklayer)
> library(RNAseqData.HNRNPC.bam.chr14)
> gtf <- system.file("examples", "apa.gtf", package="roar")
> # HNRNPC ko data
> bamTreatment <- RNAseqData.HNRNPC.bam.chr14_BAMFILES[seq(5,8)]
> # control (HeLa wt)
> bamControl <- RNAseqData.HNRNPC.bam.chr14_BAMFILES[seq(1,3)]
> rds <- RoarDatasetFromFiles(bamTreatment, bamControl, gtf)
```

3.2 Obtaining counts

This is the first step in the Roar analysis: it counts reads overlapping with the PRE/POST portions defined in the given gtf/GRanges annotation. Reads of the given bam are accounted for with the following rules:

1. reads that align on only one of the given features are assigned to that feature, even if the overlap is not complete

2. reads that align on both a PRE and a POST feature of the same gene (spanning reads) are assigned to the POST one, considering that they have clearly been obtained from the longer isoform

If the alignments were derived from a strand-aware protocol it is possible to consider strandness when counting reads using the stranded=TRUE argument; in this case we use FALSE as long as we don't want to consider strandness.

```
> rds <- countPrePost(rds, FALSE)
```

3.3 Computing roar

This is the second step in the Roar analysis: it computes the ratio of prevalence of the short and long isoforms for every gene in the treatment and control condition (m/M) and their ratio, roar, that indicates if there is a relative shortening-lengthening in a condition versus the other one (the choice of calling them "treatment" and "control" is simply a convention and reflects the fact that to calculate the roar the m/M of the treatment condition is used as the numerator). For details about the m/M calculations see section **??**.

A roar > 1 for a given gene means that in the treatment condition that gene has an higher ratio of short vs long isoforms with respect to the control condition (and the opposite for roar < 1). Negative or NA m/M or roar occurs in not definite situations, such as counts equal to zero for PRE or POST portions. If for one of the conditions there is more than one sample, like in this example that has four and three samples for the two conditions, then calculations are performed on average counts.

```
> rds <- computeRoars(rds)
```

3.4 Computing p-values

This is the third step in the Roar analyses: it applies a Fisher test comparing counts falling on the PRE and POST portion in the treatment and control conditions for every gene. If there are multiple samples for a condition every combination of comparisons between the samples lists is considered - for example in this case we have four and three samples, therefore it will calculate 12 p-values.

```
> rds <- computePvals(rds)
```

Computing all samples pairing is not the preferrable choice when a paired experimental design exists: in this case only the correct pairs between control and treatment samples should be compared with the Fisher test; then their p-values can be combined following the Fisher method ([**?**]) because we have different independent tests on the same null hypothesis. For these situations

there is an alternative way to obtain the p-values: *computePairedPvals*, which require the user to specify as arguments not only the *RoarDataset* object but also two vectors containining the ordered samples for treatment and control - see the related man page for details.

3.5 Obtaining results

There are various functions aimed at extracting results from a *RoarDataset*; the first one is *totalResults*:

```
> results <- totalResults(rds)
```

It returns a dataframe with gene_id as rownames and several columns: "mM_treatment", "mM_control", "roar", "pval" and a number of columns called "pvalue_X_Y" (when there are multiple samples for at least one condition, X refers to the treatment samples and Y to the control ones). The first two columns have the value of the ratio showing the relative abundance of the short isoform with respect to the long one in the treatment (or control) condition, the third one represents the roar and it's bigger than one when the shorter isoform is relatively more expressed in the treatment condition (and the other way around when it's smaller than one). The pval column stores the results of the Fisher tests when both conditions have a single sample, while the multiplication of all the possible combinations p-values for un-paired multiple samples analysis or the combined p-values following the Fisher method if *computePairedPvals* has been used.

In this case for example there will be columns "pvalue_1_1" up to "pvalue_4_3": the first number represents the number of the considered sample for the treatment condition, while the second for the control one (the samples are considered in the same order given in the lists passed to the *RoarDataset* constructors).

All other functions are simply helper functions: *fpkmResults* adds columns "treatmentValue" and "controlValue" representing the level of expression of the relative gene in the two conditions, while *countResults* puts in that columns the number of read that were counted on the PRE portion of genes.

standardFilter get a parameter with a cutoff for the FPKM value and selects only genes with an expression higher than that and without any NA/Inf value for m/M or roar (deriving from not definite situations with counts equal to zero). *pvalueFilter* further selects only genes with a Fisher test pvalue smaller than the given cutoff in the single sample case, while if more than one sample has been given for one condition it adds a column named "nUnderCutoff" that stores the number of p-values that are smaller than the given cutoff. It is worth noting that all the p-values are nominal with this method, while *pvalueCorrectFilter* applies an user requested multiple correction procedure (after FPKM filtering) for single sample or paired (that is when *computePairedPvals* has been used) analyses and filters with the given cutoff on these corrected value (for un-paired multiple samples design this does exactly the same as *pvalueFilter*).

For example in our case we can see how many genes have a pvalue under 0.05 in every comparisons in this way:

```
> results_filtered <- pvalueFilter(rds, fpkmCutoff=-Inf,
+                                  pvalCutoff=0.05)
> nrow(results_filtered[results_filtered$nUnderCutoff==12,])

[1] 2
```

Note that for FPKM we used -Inf as a cutoff because in this case we didn't want to filter out genes considering their expression values (and because we used only partial alignments deriving from a single chromosome, thus the usual FPKM cutoff are not directly applicable).

Another thing to point out is that the FPKM values derive from the total reads falling over the genes given in the annotation, therefore when doing the analysis "stepwise" (eg. with roarWrapper_chrByChr) their values will be different than those obtained performing the analysis all together. *countResults* could be used in this situations to save single steps counts and then calculate "whole" FPKM values.

4 Appendix

4.1 m/M calculations

To correctly evaluate the quantities ratios of the short and long isoforms counts of the reads falling over the two transcript portions (as previously defined: PRE is the portion common between the two isoforms, while POST is the one present only in the longer transcript) should be accounted for considering the fact that those falling over PRE could have been obtained from both isoforms while the others falling on POST derive exclusively from the longer one; another more trivial question that has to be addressed is that reads fall with larger frequencies on longer stretches of RNA.

We can say that the total number of reads falling over a transcript in its entirety (N) derives from the relative abundance of the two isoforms and their potential of generating reads; that is: $N = \epsilon_M M + \epsilon_m m$ where m is the quantity of the short isoform, M of the long one and ϵ identifies their efficiency in generating reads.

Assuming the equiprobability of reads distribution (that is each nucleotide has the same probability of finding itself in a read) the efficiency in generating reads of the two isoforms is related to their lengths (with the addition of a constant of proportionality k):

$$\epsilon_M = k(l_{PRE} + l_{POST})$$

$$\epsilon_m = k(l_{PRE})$$

Defining l_{PRE} as the length of the PRE portion and l_{POST} as the length of the POST we can now obtain the number of reads falling on the two portions

as:

$$\#r_{PRE} = \epsilon_M M(\frac{l_{PRE}}{l_{PRE}+l_{POST}}) + \epsilon_m m$$

and:

$$\#r_{POST} = \epsilon_M M(\frac{l_{POST}}{l_{PRE}+l_{POST}})$$

These two equations reflect the fact that all the reads deriving from the short isoform ($\epsilon_m m$) fall on the PRE portion while the ones deriving from the long one are distributed over the PRE and POST portions depending on their lengths.

We can now setup a system of equations aimed at obtaining the m/M value in terms of the numbers of reads falling over the two portions and their lengths.

$$\text{We start from: } \begin{cases} \#r_{PRE} = \epsilon_M M(\frac{l_{PRE}}{l_{PRE}+l_{POST}}) + \epsilon_m m \\ \#r_{POST} = \epsilon_M M(\frac{l_{POST}}{l_{PRE}+l_{POST}}) \end{cases}$$

Then with simple algebraic steps the system can be solved yielding the formula to obtain m/M using only read counts and lengths:

$$m/M = \frac{l_{POST}\#r_{PRE}}{l_{PRE}\#r_{POST}} - 1$$

As a simple emblematic case suppose that the PRE and POST portions have the same lengths and that the short and long isoforms are in perfect equilibrium (i.e. they are present in a cell in equal numbers). In this situation we will find on the PRE portion two times the number of reads falling on the POST one because half of them will derive from the long isoforms and the other half from the short ones. In this case the equation correcly gives us an m/M equal to 1.

In the previous discussion we have ignored reads falling across the PRE/POST boundaries. As long as they can derive only from the long isoform it is reasonable to assign them to $\#r_{POST}$.

Portion lengths should be corrected to keep in consideration read lengths and the just cited assignment to POST of the spanning reads, therefore:

$$l'_{PRE} = l_{PRE}$$
$$l'_{POST} = l_{POST} + readLength - 1$$

Normally we should have added readLength to both the lengths but in this case we do not expect reads to fall after the POST portion (that is the end of transcripts) and thus we only have to correct for the spanning reads. We do not subtract the same value from l_{PRE} as long as in theory we could expect reads

to fall at its 5' (i.e. reads falling across that exon and the previous one or those from still unspliced transcripts).
These corrected lengths are those used for the m/M calculations.