An Introduction to *SolaS*

Hugues Richard and Marcel Schulz

July 22, 2009

This document gives a short introduction to the R*SolaS* package provides key functionality for detecting and quantifying alternative splicing events in an RNA-Seq experiment. Solas is designed to infer and quantify alternative splicing based on the read densities observed within coding regions. The analysis is performed througth 3 routines:

- DASI (Differential Alternative Splicing Index) for detecting alternative splicing events differentiating two conditions
- CASI (Cell Alternative Splicing Index) for detecting genes and exons which are part of an alternative splicing event.
- POEM (Proportion Estimation) to quantify the relative proportion of different isoforms.

Application of each of the strategies will be illustrated on a canonical gene in the next section. The remaining section will deal with certain points of detail.

```
> library(Solas)
```

1 Alternative splicing analysis

The analysis proposed in Solas are always performed one gene at a time. In order to apply the methods to a gene, we need to specify two information:

- The description of the gene model (exon coordinates and transcript structures), encoded as a *GeneStructure* object.
- The read counts observed within the exons, given as a *Genecount* object.

We assume that the reads from the RNA-Seq experiment have already been aligned to the genome of interest[1], and the mapping position where intersected with genomic coordinates.

Let's work with an example gene `MyGene` consisting of 5 exons and 4 transcript isoforms. All exons are supposed to have the same length (150 bp) The gene structure is illustrated in the following figure:

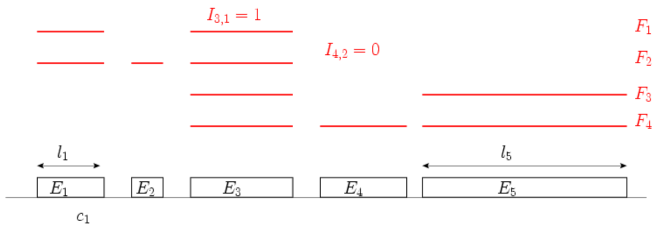

Figure 1: Example of a gene with 5 exons and 4 forms. The encoding of the different isoform structure is done throught the matrix I, which denotes at coordinate (i, j) if the i$^{\text{th}}$ form is using the the j$^{\text{th}}$ exon.

An isoform 1 (with transcript ID "Form1") skips exons num. 2, and so on... This gene model is initialized as a *GeneStructure* object with the following set of commands:

```
> transcript.names = c("Form1", "Form2", "Form3", "Form4")
> gene.info = data.frame(name = "MyGene", chr = "chr0",
+     nexons = 5, direction = "+", nforms = 4)
> I = matrix(c(1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
+     1, 0, 1, 0, 0, 1, 1, 1), ncol = 5, byrow = TRUE)
> colnames(I) = paste("E", 1:5, sep = "")
> trans.structure = data.frame(gene = "MyGene", form = transcript.names,
+     I)
> exons.coord = data.frame(gene = "MyGene", chr = "chr0",
+     subexon = paste("E", 1:5, sep = ""), begin = cumsum(c(1,
+         rep(450 - 1, 4))), end = cumsum(c(150, rep(450 -
+         1, 4))))
> mygene = gene.new(gene.info, trans.structure, exons.coord)
```

Readers interested in more details on *GeneStructure* objects creation are refered to the manual. Note that by default, the length of each exon is

[1]for a mapping program to use, see for instance RazerS at `http://www.seqan.de/projects/razers.html`

computed according to its start and end position. However, due to the short length of the reads some positions are ambiguous in the sense that a single read mapps to more than one position in the genome. In the case where certain positions cannot be mapped without ambiguities, the user can set a `n.uniq.hits` column, which lists the number of reads that can be mapped within an exon. For instance, if the 4th exon possesses 20 positions where a read map is ambiguious to another position in the genome, the `exon.coord` can be updated as follows (in this example experiment, reads have length 27bp, but the read length is an adjustable parameter):

```
> exons.coord$length = exons.coord$end - exons.coord$begin +
+     1
> exons.coord$n.uniq.hits = exons.coord$length - 27 +
+     1
> exons.coord$n.uniq.hits[4] = exons.coord$n.uniq.hits[4] -
+     20
> mygene = gene.new(gene.info, trans.structure, exons.coord)
```

Let's hypothesize we observe the following read counts on each of the exons in the two conditions:

```
> exons.count.hek = genecount.new("mygene", c(32, 10,
+     45, 56, 65))
> exons.count.bcells = genecount.new("mygene", c(23,
+     30, 70, 42, 74))
```

1.1 Prediction of alternative splicing

Without doing hypothesis on the transcript structure, we can test for the exons involved in alternative splicing events and perform a CASI routine on our gene:

```
> CASI.hek = ASI.test(mygene, exons.count.hek)
> CASI.hek

        CASI Chi-squared test for Alternative Splicing

data:
X-squared = 50.5238, df = 4, p-value = 2.807e-10

> which(CASI.hek$z.score <= -2)
```

```
[1] 2

> CASI.bcells = ASI.test(mygene, exons.count.bcells)
> CASI.bcells

        CASI Chi-squared test for Alternative Splicing

data:
X-squared = 42.5811, df = 4, p-value = 1.264e-08

> which(CASI.bcells$z.score <= -2)

integer(0)
```

Here the second exon is detected as part of an alternative splicing event with a zscore lower than -2 in HEK cells. No exons are detected with B cells.

```
> DASI = ASI.test(mygene, exons.count.hek, exons.count.bcells,
+     cell = "two")
> which(abs(DASI$z.score) >= 2)

integer(0)
```

Although the computed pvalue is significant, no exon is detected as differentially alternatively spliced between the two conditions.

1.2 Estimation of relative expression levels of known isoforms

Given transcript structures for a gene, we can estimate the different isoform proportions for the HEK cell line.

```
> poem.hek = POEM.gene(mygene, exons.count.hek, read.length = 27)
> poem.hek$p.estim

     Form1      Form2      Form3      Form4
0.16865305 0.12925760 0.04816192 0.65392743

> poem.hek$counts.estim

[1]  26.705214  30.700766   7.626155 146.967866
```

The function `gene2gff` can then be used to output the proportion together with the transcript structure as a `gff` file. One can verify the quality of the fit with the bootstrap and the goodness of fit methods.
The quality score (Goodness of fit) evaluates the proposed relative expression of each transcript for the gene according to all reads for the gene.

```
> poem.hek$quality.score

        Chi-squared test for given probabilities

data:  counts
X-squared = 23.3589, df = 4, p-value = 0.0001073
```

```
> poem.hek$quality.score$p.value

[1] 0.0001073439
```

A small p value for the quality score is indicative of a very good estimate of the algorithm, which in turn is a good indicator that the used transcript structures fit the data well. When the quality score has a high p value, in most of the cases, the transcript structures are incomplete or alternatively the proposed background distribution is inappropriate.

A useful way of assessing the robustness of the estimate is to use the implemented bootstrapping function `Bootstrap.estimate`.

```
> boot.sample = Bootstrap.estimate(poem.hek, nboot = 40)
> summary(boot.sample)

     Form1            Form2             Form3        
 Min.   :0.0928   Min.   :0.04399   Min.   :0.04445  
 1st Qu.:0.1264   1st Qu.:0.09816   1st Qu.:0.09414  
 Median :0.1733   Median :0.12330   Median :0.13276  
 Mean   :0.1706   Mean   :0.12821   Mean   :0.14957  
 3rd Qu.:0.2014   3rd Qu.:0.15620   3rd Qu.:0.20158  
 Max.   :0.2575   Max.   :0.24713   Max.   :0.32989  
     Form4       
 Min.   :0.4070  
 1st Qu.:0.5151  
 Median :0.5502  
 Mean   :0.5516  
 3rd Qu.:0.5982  
 Max.   :0.6762  
```

The bootstrap analysis shows, that the original estimate of Form3 in `p.estim` likely underestimates the relative expression level of Form3. The bootstrapping shows that Form2 and Form3 should be expressed around the same level.

2 Future improvements and extensions

The future improvement planned for the *SolaS* package are:

1. integration of splice junction counts for the estimation of transcripts proportions.
2. more general set of strategy for POEM estimation, integrating: paired-end reads, allele specific expression...