easyRNASeq, an overview

Nicolas Delhomme, Ismaël Padioleau

October 2, 2013

Contents

1 Before you start **2**

2 Changes to the vignette **2**

3 Introduction **4**

4 easyRNASeq **6**
4.1 usage . . . 7
4.2 description . . . 10
4.3 different input . . . 13
4.4 different annotation . . . 15
4.5 different output . . . 19
4.6 different summarization . . . 20
4.7 optional normalization . . . 22
4.8 Performance . . . 25

5 Advanced usage **26**
5.1 De-multiplexing samples . . . 26

6 Yet to come **31**

7 Use Cases **32**

8 Session Information **42**

9 Final remarks **44**

1 Before you start

If you are completely new to the R/Bioconductor [Gentleman et al., 2004] packages dealing with Next-Generation Sequencing, you might want to start by doing the tutorial available in the `RnaSeqTutorial` vignette in the data package *RnaSeqTutorial*. The same holds true if you're interested in understanding the details behind the content of the present vignette. If neither nor, just go ahead.

Moreover, if you find *easyRNASeq* useful and apply it in the frame of your research for a publication, please cite it:

easyRNASeq: a bioconductor package for processing RNA-Seq data
Nicolas Delhomme; Ismael Padioleau; Eileen E. Furlong; Lars M. Steinmetz
Bioinformatics 2012; doi: 10.1093/bioinformatics/bts477

2 Changes to the vignette

The last changes are reported in red.

1.3.14 A new section: 6 was added describing foreseen changes to the package, see page 31. Additional changes to the vignette use-case where made thanks to Richard Friedman's suggestions, see section 7, page 32.

1.3.10 - 1.3.13 Some vignette discrepancies have been corrected. Thanks to Richard Friedman for spotting them, see paragraph 4.1, page 8. Some additional information about the 'gtf' and 'gff' format has been added after fixing the bug reported by Tomasz Kulinski. See section 4.1, page 7.

1.3.9 Not much change to the vignette, but for the fact that the *easyRNASeq* application note got published in **Bioinformatics** [Delhomme et al., 2012].

1.3.5 - 1.3.8 The vignette has been updated to report function call changes to the `easyRNASeq` function:

- the format argument defaults to bam
- the chr.sizes can be derived from the bam file header
- the addition of the knownOrganisms function
- support for variable length reads
- read files can be processed in parallel

Sections affected are: 4.1 (page 7), 4.1 (page 9), 4.2 (page 10), 4.3 (page 13), 4.8 (page 25),7 (page 32),7 (page 37)

1.3.1 - 1.3.4 The vignette has been updated to:

- first: enhance its readability.
- second: introduce a new **Use Case** section

Sections affected are: 4 (page 6), 4.1 (page 7), 4.2 (page 10), 4.4 (page 15), 5.1 (page 26), 7 (page 32)

3 Introduction

This file describes the use of the *easyRNASeq* package to ease the processing of RNA-seq data in R/Bioconductor. RNA-seq [Mortazavi et al., 2008] was introduced as a new method to perform Gene Expression Analysis, using the advantages of the high throughput of *Next-Generation Sequencing* (NGS) machines.

The goal of this vignette is, first, to show an example of the `easyRNASeq` function that generates a count table for the selected genic features of interest, *i.e.* exons, transcripts, gene models, *etc.* using the read data and the genic and genomic annotations. This overall process is described in figure 1, page 5. Shortly, the genomic and genic annotation will be retrieved from the selected/preferred source and converted into an appropriate `object`. In parallel, the sequenced reads' information (*e.g.* chromosome, position, strand, *etc.*) will be retrieved from the alignment file and, similarly, converted to an appropriate `object`. Then, the reads contained in the `reads` object are summarized per genic annotation contained in the `annotation` object, which give raise to a count table that, finally, can be normalized using additional R packages.

Then, the `easyRNASeq` function arguments will be detailed and additional information given on how to generate normalized data, such as creating *RPKM (Reads Per Kilobase of feature per Million reads in the library)* counts or by using more advanced statistical models implemented in the *DESeq* or *edgeR* packages.

In the second part, we will see how more advanced pre-processing can be performed, such as *de-multiplexing*.

Additional post-processing such as exporting the count table as bed and wig formatted file, to be visualized into the UCSC genome browser or a stand alone genome browser are described in the vignette of the companion data package *RnaSeqTutorial*.

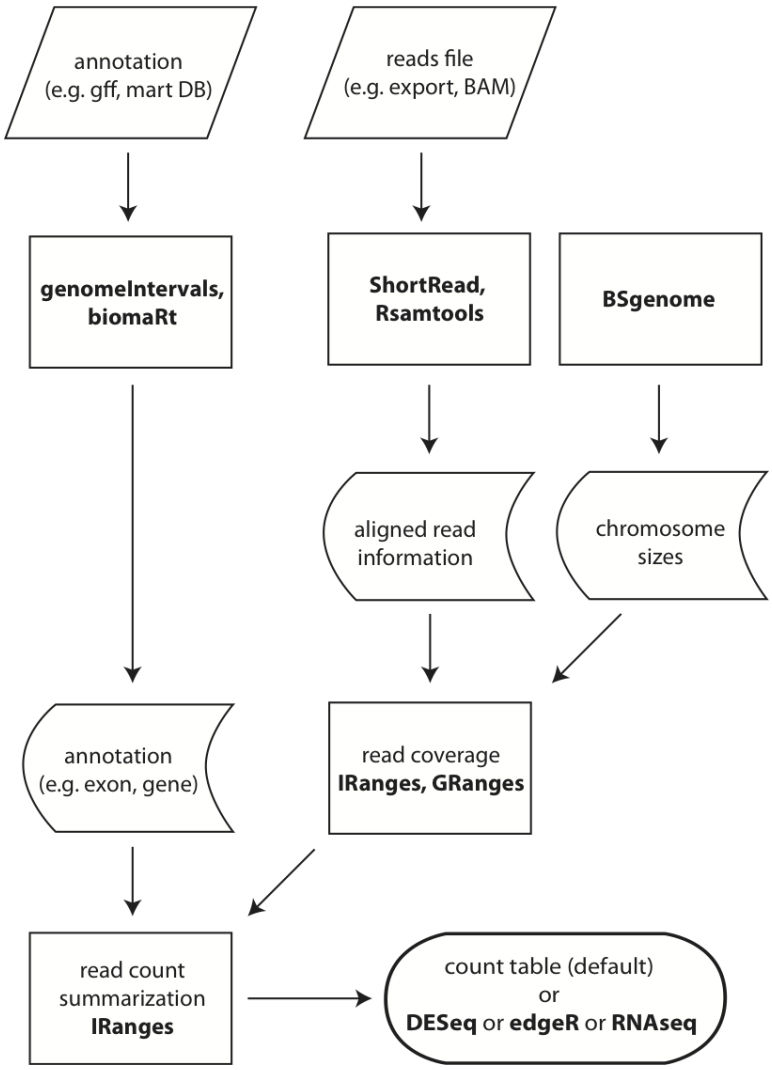


Figure 1: easyRNAseq Overview. The R packages used for the different steps are emphasized in bold face.

4 easyRNASeq

We will use the data present in the *RnaSeqTutorial* data package to perform the examples. Refer to its documentation for more details. Shortly, it contains different *Drosophila melanogaster* RNA-Seq dataset saved in different format as well as the related annotation retrieved from FlyBase [Tweedie et al., 2009].

The main function of the *easyRNASeq* package is `easyRNASeq`. That should be the only processing method you need to know about when using the package. It is essentially a wrapper around other function performing the different tasks described in the following, functions which are all exported too, if you feel you need to have a look at them. The *easyRNASeq* package defines an `RNAseq` class for holding the necessary information used during the processing. The `easyRNASeq` function instantiates an object of that class and although most arguments of the `easyRNASeq` function are optional (as are the corresponding *RNAseq* object slots),it is advised to set them, especially the *readLength* and the *organismName*; to help having a proper documentation of your analysis. The organismName slot is actually mandatory if you want to get genomic annotation using *biomaRt*. In that case, you need to provide the name as specified in the corresponding *BSgenome* package, *i.e.* "Dmelanogaster" for the *BSgenome.Dmelanogaster.UCSC.dm3* package.

Important Note: As there are numerous RNA-Seq protocol, each with its own specificities, the `easyRNASeq` offers many arguments to adapt to these. This number of arguments, with only a few defaults values, might be intimidating. We are in the process of consolidating them to make it easier to handle them. In the meantime, please refer to the documentation and the examples in this vignette and the *RnaSeqTutorial* one to get familiar with them. Do not hesitate to ask on the Bioconductor mailing list if the purpose of some argument was still unclear.

4.1 usage

Initial example In the following, the `easyRNASeq` function is called with an almost minimal set of parameters (some optional ones too...). The sequencing reads' excerpts were obtained from a fruit-fly sample that was subjected to a 36 cycles sequencing on an Illumina GAIIX machine. The annotation were retrieved from FlyBase and converted into an "Rdata" object. We're interested to retrieve the count table of 4 samples for every exon in the genome. As of version 1.3.5, 'bam' is the default format. Hence, the argument 'format' has been removed from the following function call.

```
> library(easyRNASeq)
> library(RnaSeqTutorial)
> library(BSgenome.Dmelanogaster.UCSC.dm3)
> count.table <- easyRNASeq(filesDirectory=system.file(
+                             "extdata",
+                             package="RnaSeqTutorial"),
+                           pattern="[A,C,T,G]{6}\\.bam$",
+                           readLength=30L,
+                           organism="Dmelanogaster",
+                           chr.sizes=seqlengths(Dmelanogaster),
+                           annotationMethod="rda",
+                           annotationFile=system.file(
+                             "data",
+                             "gAnnot.rda",
+                             package="RnaSeqTutorial"),
+                           count="exons"
+                           )

Checking arguments...
Fetching annotations...
Summarizing counts...
Processing ACACTG.bam
Processing ACTAGC.bam
Processing ATGGCT.bam
Processing TTGCGA.bam
Preparing output

> head(count.table)

           ACACTG.bam ACTAGC.bam ATGGCT.bam TTGCGA.bam
CG11023:1          0          0          0          0
CG11023:2          0          0          0          0
CG11023:3          0          0          0          1
CG2671:1           0          0          0          0
```

```
CG2671:2            1           0           0           1
CG2671:3           13           8          11          12

> dim(count.table)

[1] 69306     4
```

In this command, we're reading bam files (`format`), present in the first argument to the function (`filesDirectory`) using a regular expression to match the file names (`pattern`). The `system.file` retrieve the actual path on your file system where the "RnaSeqTutorial" package was installed and in that particular case of its "extdata" sub-directory. The pattern used to retrieve the files is similar to that of the *base* commands `dir` and `grep`. Here we look up files, which name contains 6 times one of the "A", "C", "G" or "T" letter, followed by the ".bam" extension. We additionally define the number of sequencing cycles (`readLength`), and the `organism` that sample originates from. We then provide genomic and genic information. The size of the chromosomes (`chr.sizes`) and the exons' annotation (`annotationFile`) that were already processed in an "RData" file (`annotationMethod`). Finally, we precise that we want the sequenced reads to be summarized by "exons"; *i.e.* we want to get a read count value per exon and per sample.

And that is all. In that one command, you got the exon count table for your 4 samples!

Important Note: It is not required to use any of the *BSgenome* packages to retrieve the chr sizes. These can be provided as a simple named vector. We just use the *BSgenome.Dmelanogaster.UCSC.dm3* package for commodity. Moreover, since version 1.3.5, the 'chr.sizes' can be extracted from the BAM file header; *i.e.* setting 'chr.sizes' to "auto" (the default) would have retrieved the chromosome sizes from the BAM file. Note that this obviously does not work if your format is not 'bam'. See the additional comments in the use case page 34. In addition, since version 1.3.10, the 'filesDirectory' argument defaults to the current directory. This one can be looked up using the `getwd()` function.

Warnings As you could see when running the previous example, warnings were emitted and quite rightly so.

1. about the annotation: The annotation we are using here is redundant and this at two levels. First, some exons overlap. These are alternative exons from different transcript isoforms. Second, the annotation contains the information about all the possible different transcript isoforms. This means that some exons are duplicated. Therefore counting by exons or transcripts using these annotation will result in counting some of the reads several times. There might be reasons one might

want to do that, but as it is probably not what you want when performing an RNA-Seq analysis, the warning is emitted. As this can be a very significant source of error, all the examples here will emit this warning, at the exception of the summarization by "GeneModels" that address that problem (*c.f.* 4.6, page 20). The ideal solution is to provide an annotation object that contains no overlapping features. The `disjoin` function from the *IRanges* package offers a way to achieve this.

2. about potential naming issue in the input file: It is (sadly) very frequent that the sequencing facilities use different naming conventions for the chromosomes they report in the alignment files. It is therefore very frequent that the annotation provided to easyRNASeq uses different chromosome names than the alignment file. These warnings are there to inform you about this issue.

Arguments Going back to the `easyRNASeq` function, more arguments can be set to fine tune it, depending on the input at hand, the desired output, etc. Most will be detailed in the following, but not all, so check "?easyRNASeq" out for more. One such argument is: "filenames", that offers the possibility to give actual filenames rather than using a pattern matching approach to find them. For example, if you're only interested in two samples out of the four available ones in the provided directory, do:

```
> count.table <- easyRNASeq(system.file(
+                                        "extdata",
+                                        package="RnaSeqTutorial"),
+                           organism="Dmelanogaster",
+                           readLength=30L,
+                           annotationMethod="rda",
+                           annotationFile=system.file(
+                             "data",
+                             "gAnnot.rda",
+                             package="RnaSeqTutorial"),
+                           count="exons",
+                           filenames=c("ACACTG.bam", "ACTAGC.bam"))
```

As of version 1.3.5, 'bam' is the default format. Hence, the argument 'format' has been removed from the previous function call. The same is true for the 'chr.sizes' argument. Removing it results in setting it to its default value: 'auto', which has the effect of determining the chromosome sizes using the BAM header.

```
> head(count.table)
> dim(count.table)
```

4.2 description

There are several arguments that influence how the `easyRNASeq` is going to be executed. They concern:

- the input at hand
- how to retrieve the annotations
- how to summarize the counts
- the desired output
- whether normalization should be performed

Input Using the `format` arguments, reads can be loaded from BAM files or any format supported by the *ShortRead* [Morgan et al., 2009] package. The SAM/BAM format [Li et al., 2009] is becoming a *de-facto* standard for storing NGS aligned data. Gapped alignment can be read from BAM files, set the `gapped` argument to "TRUE" for that. As of version 1.3.5, the 'bam' format is the default. If you are using bam files, you do not need to precise that argument anymore. As of version 1.3.6, it is possible to set additional options to locate the read files. All the options from the list.files base function are available, *i.e.* one can search for file recursively. As of version 1.3.7, variable read lengths is supported. The current implementation results in a small, neglictible approximation of the read counts, due to the likelyhood of an integer overflow. This is currently investigated and appropriate warnings will be raised if it were to affect the results.

Annotation The `easyRNASeq` function currently accepts the following `annotationMethods`:

- "biomaRt" use the *biomaRt* [Durinck et al., 2005] package to retrieve the annotation
- "env" use a *RangedData* or *GRangesList* class object present in the environment - see page 36 for a use case.
- "gff" reads in a gff version 3 file
- "gtf" reads in a gtf file as provided by Ensembl [Flicek et al., 2011]
- "rda" load an RData object. The object needs to be named `gAnnot` and of class *RangedData* or *GRangesList*. See the use case page 35 to prepare such an object using the *GenomicFeatures*. ".rda" is the filename extension of serialized R data object (*i.e.* object written to disk using the `save` function). The filename extension for these

files used to be and still is sometimes".RData". The actual filename extension used is of no importance to `easyRNASeq` - see page 35 for a use case.

The structure of the *RangedData* object that has to be provided when using the "rda" or "env" `annotationMethods` is described in the section 4.4 (page 15). For using a *GRangesList*, see the use case page 35. More details on the gff3 and gtf format can be obtained from:

- gff3: http://www.sequenceontology.org/gff3.shtml
- gtf: http://mblab.wustl.edu/GTF22.html

Unlike for the gtf, the gff3 ninth column holding the attributes has no mandatory tags. Some of them are however "predefined". As exemplified further down (see 4.4, 16), we expect these "predefined" attributes to be used and we rely on the "ID", "Parent" and "Name" to identify the relation between a gene and its exons and transcripts.

Summarization The reads can be counted/summarized by:

- exons
- features (any features such as introns, enhancers, *etc.*)
- transcripts
- geneModels: they consists of the set of disjoint "synthetic" exons that fully overlap every known exons and UTRs of a gene; *I.e.* every alternate exon will be split into separate "synthetic" exons and these exons will be grouped into a set that correspond to a single gene.

When you use a the "geneModels" summarization, no reads will be counted twice at the exception of those mapping overlapping exons of different genes, whether on the same strand or not. The `easyRNASeq` does not deal with these situation, but as previously a warning will be emitted if this occurs.

Output The results can be exported in four different formats:

- count table (the default, a n (features) x m (samples) `matrix`).
- a *DESeq* [Anders and Huber, 2010] *CountDataSet* class object. Useful to perform further analyses using the *DESeq* package.
- an *edgeR* [Robinson et al., 2010] *DGEList* class object. Useful to perform further analyses using the *edgeR* package.
- an *RNAseq* class object. Useful for performing additional pre-processing without re-loading the reads and annotations.

Normalization The results can optionally be "normalized" as *Reads per Kilobase of feature per Million reads in the library* (RPKM, Mortazavi et al. [2008]). In addition, the normalization step is available when the results are exported in the *DESeq* or *edgeR* formats. This will as well generate plots to assess

the validity of the normalization and help decide how to proceed with any downstream analysis.

In the following paragraphs, the usage of these different arguments will be presented.

4.3 different input

The following show how to load an 'export' file. This was the default Illumina output before Illumina released CASAVA v1.8 in spring 2011. The alignment are done with ELAND. The `format` argument has to be set to the "aln" value and an additional arguments: `type` has to be provided to precise the input file format. See the *ShortRead* `readAligned` help page for all the available formats. Note as well the `filter` argument that is essential to filter the usable reads from the export file. Indeed, the export file does contain every read, including those not aligning and those not passing the original quality filter performed during the Illumina Base Calling (Bustard) a.k.a. the chastity filter. For more details, see the section "Filtering the Data" in the *RnaSeqTutorial* vignette.

```
> count.table <- easyRNASeq(system.file(
+                                       "extdata",
+                                       package="RnaSeqTutorial"),
+                           organism="Dmelanogaster",
+                           chr.sizes=seqlengths(Dmelanogaster),
+                           readLength=36L,
+                           annotationMethod="rda",
+                           annotationFile=system.file(
+                             "data",
+                             "gAnnot.rda",
+                             package="RnaSeqTutorial"),
+                           format="aln",
+                           type="SolexaExport",
+                           count="exons",
+                           pattern="subset_export",
+                           filter=compose(
+                             chromosomeFilter(regex="chr"),
+                             chastityFilter()))
```

Note that here the chr.sizes argument has to be specified. Indeed, we are using the 'aln' format and consequently cannot determine the chromosome sizes from the header of the BAM files.

In the next example, we read a BAM file containing gapped alignments. In that particular case, TopHat [Trapnell et al., 2009] was used to look for splicing events. Note the use of the `gapped` boolean argument. As of version 1.3.5, the format argument default to bam and has therefore been removed from the following function call.

```
> count.table <- easyRNASeq(system.file(
+                                       "extdata",
+                                       package="RnaSeqTutorial"),
```

```
+                         organism="Dmelanogaster",
+                         chr.sizes="auto",
+                         readLength=36L,
+                         annotationMethod="rda",
+                         annotationFile=system.file(
+                           "data",
+                           "gAnnot.rda",
+                           package="RnaSeqTutorial"),
+                         gapped=TRUE,
+                         count="exons",
+                         filenames="gapped.bam")
```

Note that setting the 'chr.sizes' argument to 'auto' is the same as removing it from the function call.

4.4 different annotation

The following paragraphs present how to use different annotation sources to retrieve the genic information. It describes as the well the expected format of these annotations.

Important Note: There are many derivative of the GFF3 and GTF file formats, some of which do not strictly follow the convention these files should have. Using incorrectly formatted file will fail and most of the time a relevant error message will be reported. GTF files as produced by Ensembl and GFF3 files as produced by Flybase are correct implementation of these formats and are processed smoothly by the `easyRNASeq` function. If you encounter problem while using such an annotation file, please try to correct them. For the GFF format, the example file present in the *RnaSeqTutorial* package can be used as a guideline. If you have installed this package, suggested by the *easyRNASeq* package, the following will give you the path of that file on your machine:

```
> system.file(
+             "extdata",
+             "annot.gff",
+             package="RnaSeqTutorial")
```

If you do not manage to identify the issue, please contact the Bioconductor mailing list for help.

biomaRt The following show how to use biomaRt to retrieve the annotations.

```
> count.table <- easyRNASeq(system.file(
+                                       "extdata",
+                                       package="RnaSeqTutorial"),
+                           organism="Dmelanogaster",
+                           readLength=36L,
+                           annotationMethod="biomaRt",
+                           gapped=TRUE,
+                           count="exons",
+                           filenames="gapped.bam")
```

Once the annotation has been fetched, one can retrieve them from an `RNAseq` object. This implies that the "outputFormat" argument will have been set to "RNAseq", so that the complete object is returned and not only the count table (*c.f.* 4.5, page 19). Using the "genomicAnnotation" accessor on that object, one can get the annotation as an *RangedData* object. This object can be saved in an "rda" file under the name "gAnnot" for a faster processing using the `easyRNASeq` function with the argu-

ments"annotationMethod" and "annotationFile" set to "rda" and the path where you would have saved the rda file.

genomeIntervals In the next example is shown how to use a gff file for the same purpose. Clearly, every gff formatted file will have different gff attributes (its last column). This column contains "key=value" pairs separated by semi-colons. We depend on specific key to be present as we have been using Flybase gff files to develop that functionality; hence we expect attributes as defined by Flybase. First, we're only considering feature of type 'exon' in the third column. Then we expect the following keys: *ID*, *Parent*, and *Name* in the attributes. The *ID* key should have the format: *geneID:exonNumber*; both the exon and gene are derived from it. The *Parent* key should contain the transcript ID. If one exon is part of several transcripts, these can be concatenated using commas without space. Finally, the *Name* key should contain the gene name, but is actually optional. As we are using the `readGff3` function from the *genomeIntervals* package, it is as well essential that the pairs key=value are separated by semi-colons **whitout** space. See the "annot.gff" file used in the next example.

```
> count.table <- easyRNASeq(system.file(
+                                        "extdata",
+                                        package="RnaSeqTutorial"),
+                           organism="Dmelanogaster",
+                           readLength=36L,
+                           annotationMethod="gff",
+                           annotationFile=system.file(
+                             "extdata",
+                             "annot.gff",
+                             package="RnaSeqTutorial"),
+                           gapped=TRUE,
+                           count="exons",
+                           filenames="gapped.bam")
```

RangedData Internally, an `easyRNAseq` function call with the `annotationMethod` "gff", "gtf", or "biomaRt" will convert the annotation into an object of the *RangedData* class. Using the `annotationMethod` "env" or "rda" requires that you provide such an object. The following example shows the expected structure of such an object.

```
> gAnnot <- RangedData(
+                       IRanges(
+                               start=c(10,30,100),
+                               end=c(21,53,123)),
+                           space=c("chr01","chr01","chr02"),
```

```
+                           strand=c("+","+","-"),
+                           transcript=c("trA1","trA2","trB"),
+                           gene=c("gA","gA","gB"),
+                           exon=c("e1","e2","e3"),
+                           universe = "Hs19"
+                           )
> gAnnot

RangedData with 3 rows and 4 value columns across 2 spaces
     space     ranges |      strand  transcript        gene        exon
  <factor>  <IRanges> | <character> <character> <character> <character>
1    chr01 [ 10,  21] |           +        trA1          gA          e1
2    chr01 [ 30,  53] |           +        trA2          gA          e2
3    chr02 [100, 123] |           -         trB          gB          e3
```

Here we are describing the genomic location of three exons (chromosome, position and strand) as well as their transcript and gene membership. Nothing else is actually required. Note that the names' spelling is **essential** for the `easyRNAseq` function to work properly. Indeed, the `count` argument will be used to lookup the proper values into the annotation, *e.g.* for summarizing by "exons", an "exon" column in the *RangedData* object is mandatory; a "feature" one for counting "features", a "transcript" one for "transcripts", *etc.* To have a look at the *RangedData* class `gAnnot` object used in this tutorial, you can do the following:

```
> data(gAnnot)
> gAnnot
```

The *easyRNASeq* package support as well annotation provided as a *GRangesList* class object (from the *GenomicRanges* package). Converting a *RangedData* class object into a *GRangesList* class object is pretty straightforward.

```
> grngs <- as(gAnnot,"GRanges")
> grngsList<-split(grngs,seqnames(grngs))
> grngsList

GRangesList of length 2:
$chr01
GRanges with 2 ranges and 3 metadata columns:
      seqnames    ranges strand |  transcript        gene        exon
         <Rle> <IRanges>  <Rle> | <character> <character> <character>
  [1]    chr01  [10, 21]      + |        trA1          gA          e1
  [2]    chr01  [30, 53]      + |        trA2          gA          e2
```

```
$chr02
GRanges with 1 range and 3 metadata columns:
      seqnames     ranges strand | transcript gene exon
  [1]    chr02 [100, 123]      - |        trB   gB   e3

---
seqlengths:
 chr01 chr02
    NA    NA
```

The advantage of doing so is that the *RangedData* class might get deprecated in the future. The next advantage is that the *GRangesList* class is strand-aware. Therefore, support for strand-specific data is going to be supported as soon as I put my hand on such a dataset. Any suggestion, or data excerpt is welcome.

Remember Generating the proper annotation is probably the most important step in processing your RNA-Seq sample. Mind the warnings produced by the `easyRNASeq` function, they might be annoying, but there are there for a good reason: to help.

4.5 different output

For example, to generate from the same input files and annotation an object of class *RNAseq* , use the `outputFormat` argument as follow:

```
> rnaSeq <- easyRNASeq(system.file(
+                                    "extdata",
+                                    package="RnaSeqTutorial"),
+                      organism="Dmelanogaster",
+                      readLength=30L,
+                      annotationMethod="rda",
+                      annotationFile=system.file(
+                        "data",
+                        "gAnnot.rda",
+                        package="RnaSeqTutorial"),
+                      count="exons",
+                      pattern="[A,C,T,G]{6}\\.bam$",
+                      outputFormat="RNAseq")

Checking arguments...
Fetching annotations...
Summarizing counts...
Processing ACACTG.bam
Processing ACTAGC.bam
Processing ATGGCT.bam
Processing TTGCGA.bam
Preparing output
```

More details on how to generate *CountDataSet* (*DESeq*) or *DGEList* (*edgeR*) will be presented further down, see section 4.7, page 22.

4.6 different summarization

easyRNASeq offers two possibilities to apply different reads' summarization. First, as we have seen previously, by using one of the four possible `count` argument: "exons", "features", "transcripts" or "genes". However, if you want to perform different summarization on the same data, *e.g.* by "exons" and "transcripts", calling the `easyRNASeq` twice is inefficient as you will be processing the read files and fetching the annotation twice. To prevent this, one can provide the "RNAseq" value to the `outputFormat` argument, in which case, the entire object is returned. Using specific functions on that object is the second way to apply different summarization on the same data. These functions are:

- `exonCounts` function to calculate the exon counts
- `featureCounts` function to calculate the exon counts
- `transcriptCounts` function to calculate the transcript counts
- `geneCounts` function to calculate the gene counts. It takes an additional parameter defining the kind of gene summarization, either `bestExons` or `geneModels`. The `bestExons` summarization will return per gene, the highest exon count. The `geneModels` summarization first calculate a gene model and then return the read count for it.
- `readCounts` function to access the different count tables stored in the `RNAseq` object.

For example, to summarize the reads per transcripts, we will use the `transcriptCounts` function on the previously generated `rnaSeq` object and then use the `readCounts` function to access the transcript count table.

```
> rnaSeq <- transcriptCounts(rnaSeq)
> head(readCounts(rnaSeq,'transcripts'))
```

```
             ACACTG.bam ACTAGC.bam ATGGCT.bam TTGCGA.bam
FBtr0005009          0          0          0          0
FBtr0005088         56         29         44         52
FBtr0005673          2          0          1          2
FBtr0005674          7          3          9          6
FBtr0006151          0          0          0          1
FBtr0070000          0          0          0          0
```

Summarizing by transcript introduces some complexity in the data analysis, *i.e.* exons part of different isoforms introduce a bias in the counts. For that reason, it might be better to have a first look at the data, summarized by genes. This, however, requires to combine all the alternative exons and

UTRs present for every gene into a "gene model"; *i.e.* overlapping exons are merged into "synthetic" ones. This is what is performed when the arguments "count" and "summarization" are set to "genes" and "geneModels", respectively. A caveat not addressed by this procedure are genes overlapping on the same or opposite strands. If this occurs a warning will be emitted. If the reads were summarized by "geneModels" and the "outputFormat" argument was set to "RNAseq", one can use the "geneModel" accessor on the obtained object to access the computed gene models. They are stored in an *RangedData* object and can be modified to address the caveat previously mentioned. To be strict, one would remove every overlapping loci and conserve only the other ones. Such a modified annotation can then be saved and used for the next `easyRNASeq` run.
It is not possible yet to summarize by "geneModels" using the `geneCounts` function. A meaningful error message is thrown if the `geneCounts` is used for that purpose.

```
> try(geneCounts(rnaSeq,summarization='geneModels'))
```

This behavior will be corrected in the next release. At the moment, this has to be done using the `easyRNASeq` function directly.

```
> rnaSeq2 <- easyRNASeq(system.file(
+                                    "extdata",
+                                    package="RnaSeqTutorial"),
+                       organism="Dmelanogaster",
+                       readLength=30L,
+                       annotationMethod="rda",
+                       annotationFile=system.file(
+                         "data",
+                         "gAnnot.rda",
+                         package="RnaSeqTutorial"),
+                       count="genes",
+                       summarization="geneModels",
+                       pattern="[A,C,T,G]{6}\\.bam$",
+                       outputFormat="RNAseq")
> head(readCounts(rnaSeq2,'genes','geneModels'))
```

4.7 optional normalization

In this section, the different "normalization" available will be described. First the RPKM, even though this cannot be considered as a proper normalization based on sound statistical models, but is rather a kind of common sense "correction" that one can apply to the data. This implies normalizing the read counts depending on the genic feature size (exon, transcript, gene model,...) and on the total number of reads sequenced for that library. I would not recommend to use it for doing any kind of differential expression analysis, but it's definitely sufficient to create tracks to be displayed in a genome browser in order to get a feel when comparing different sample visually.

```
> count.table <- easyRNASeq(system.file(
+                                        "extdata",
+                                        package="RnaSeqTutorial"),
+                           organism="Dmelanogaster",
+                           readLength=30L,
+                           annotationMethod="rda",
+                           annotationFile=system.file(
+                             "data",
+                             "gAnnot.rda",
+                             package="RnaSeqTutorial"),
+                           count="exons",
+                           filenames=c("ACACTG.bam", "ACTAGC.bam",
+                             "ATGGCT.bam", "TTGCGA.bam"),
+                           normalize=TRUE
+                           )
```

In addition, `easyRNASeq` count tables can be easily transformed into RPKM, by using the `RPKM` method:

```
> feature.size = width(genomicAnnotation(rnaSeq))
> names(feature.size) = genomicAnnotation(rnaSeq)$exon
> lib.size=c(
+   "ACACTG.bam"=56643,
+   "ACTAGC.bam"=42698,
+   "ATGGCT.bam"=55414,
+   "TTGCGA.bam"=60740)
> head(RPKM(readCounts(rnaSeq,summarization="exons")$exons,
+           NULL,
+           lib.size=lib.size,
+           feature.size=feature.size))

           ACACTG.bam ACTAGC.bam ATGGCT.bam TTGCGA.bam
CG11023:1     0.0000     0.0000     0.0000    0.00000
```

```
CG11023:2     0.0000     0.0000     0.0000    0.00000
CG11023:3     0.0000     0.0000     0.0000   19.98012
CG2671:1      0.0000     0.0000     0.0000    0.00000
CG2671:2    129.8120     0.0000     0.0000  121.05600
CG2671:3    294.6183   240.5166   254.8213  253.61153
```

The same can be directly done on object of the *RNAseq* class.

```
> head(RPKM(rnaSeq,from="transcripts"))
```

```
            ACACTG.bam ACTAGC.bam ATGGCT.bam TTGCGA.bam
FBtr0005009         NA         NA         NA         NA
FBtr0005088         NA         NA         NA         NA
FBtr0005673         NA         NA         NA         NA
FBtr0005674         NA         NA         NA         NA
FBtr0006151         NA         NA         NA         NA
FBtr0070000         NA         NA         NA         NA
```

```
> head(RPKM(rnaSeq2,from="geneModels"))
```

A better way of normalizing the data is to use either the *edgeR* or *DESeq* packages, provided you have got enough replicates.

DESeq To be able to normalize the data using *DESeq* (or *edgeR* for that matter), one needs to define the samples' "conditions", *e.g.* "disease" vs. "healthy". To ensure traceability, the *easyRNASeq* package require the conditions to be a *named vector* where the names are the raw data filenames.

```
> conditions=c("A","A","B","B")
> names(conditions) <- c("ACACTG.bam", "ACTAGC.bam",
+                        "ATGGCT.bam", "TTGCGA.bam")
```

Once the conditions have been defined, one can call the `easyRNASeq` function with the `outputFormat` argument set to DESeq. The `normalize` argument precise whether or not to perform the normalization implemented in *DESeq*. If either case, a `CountDataSet` object is returned containing or not normalized values. Here, in addition, we precise the kind of fit that needs to be performed by the `estimateDispersion` function of the *DESeq* package. Since we have so few data, the default fit would fail and we would get an error telling us to change the `fitType` argument. As a consequence, we set that argument to *local*.

```
> countDataSet <- easyRNASeq(system.file(
+                                        "extdata",
+                                        package="RnaSeqTutorial"),
```

```
+                             organism="Dmelanogaster",
+                             readLength=30L,
+                             annotationMethod="rda",
+                             annotationFile=system.file(
+                               "data",
+                               "gAnnot.rda",
+                               package="RnaSeqTutorial"),
+                             count="exons",
+                             filenames=c("ACACTG.bam", "ACTAGC.bam",
+                               "ATGGCT.bam", "TTGCGA.bam"),
+                             normalize=TRUE,
+                             outputFormat="DESeq",
+                             conditions=conditions,
+                             fitType="local"
+                             )
```

Note that the plot produced here are irrelevant since the dataset is too small. You can turn the plotting off by setting the `plot` argument to *FALSE*. Have a look at the *DESeq* vignette (essential if you plan to use *DESeq* anyway!) for proper explanation about these.

edgeR Here we perform the same procedure as that described for the *DESeq* in the paragraph above, using the *edgeR* instead. We simply use "edgeR" as the value to the "outputFormat" argument.

```
> dgeList <- easyRNASeq(system.file(
+                                     "extdata",
+                                     package="RnaSeqTutorial"),
+                         organism="Dmelanogaster",
+                         readLength=30L,
+                         annotationMethod="rda",
+                         annotationFile=system.file(
+                           "data",
+                           "gAnnot.rda",
+                           package="RnaSeqTutorial"),
+                         count="exons",
+                         filenames=c("ACACTG.bam", "ACTAGC.bam",
+                           "ATGGCT.bam", "TTGCGA.bam"),
+                         normalize=TRUE,
+                         outputFormat="edgeR",
+                         conditions=conditions
+                         )
```

As mentioned in the former paragraph about *DESeq* (4.7, page 23, the plots generated here are only semi-relevant. Check out the *edgeR* package

vignette for more details about these plots. See in particular chapter 10 (*edgeR* v2.3.24, `R` v2.14.0). Note that producing the plots is rather slow.

Next steps At this stage you are done with the normalization and what's ahead of you: calling differential expression, exporting track files for visualization, etc. is not the scope of the *easyRNASeq* package. This one has a few more functionalities, the most important of which will be described in the next section. To proceed with your data analysis, check the relevant package vignettes (*DESeq*, *edgeR*) for differential expression calling and the *RnaSeqTutorial* for examples of track files generation using the *rtracklayer* package.

4.8 Performance

As of version 1.3.8, it is possible to parallelize some additional steps in the workflow. So far, only the generation of the geneModel could be parallelized. As of now, the actual read counting and summarization can be parallelized too. To achieve this, the 'nbCore' argument simply has to be set to the number of CPU cores, one wants to use. There are a number of word of caution though:

1. CPU cores means CPU cores, *i.e.* located on the same physical machine. Do not expect it to work across machines.
2. there's no load nor memory management, so watch out yourself for it. Memory will scale linearly with the number and size of read libraries (e.g. bam files) you process in parallel.
3. It's using the R 'parallel' package, so it's supported on all three main OS. However, some cosmetic reporting might get lost.

That's it for the cautionary part. And despite these, try it, you'll get a time boost and that's worth monitoring CPU and RAM for a while.

5 Advanced usage

In this section we will discuss about more advanced RNA-Seq pre-processing, such as de-multiplexing, or the *de novo* identification of expressed regions. The same can be directly done on object of the *RNAseq* class.

5.1 De-multiplexing samples

Nowadays, NGS machines produces such huge quantity of"raw" reads (40M, 100M and 160M per Illumina GAIIx, ABI SOLiD4 or Illumina HiSeq lanes (or equivalent), respectively), that the coverage obtained per lane for the transcriptome of "small" genome-sized organisms, is for a single sample essentially a waste of resource. *E.g.* one lane of HiSeq results in an approximate 2,800X coverage if the *Saccharomyces cerevisiae* genome which is 12Mb large. Therefore, techniques to have several samples running as a *single* library have been created [Lefrançois et al., 2009, Smith et al., 2010], using 4-6bp barcodes to uniquely identify the sample. This is called **multiplexing** and one can today with an average Illumina GenomeAnalyzer GAIIx average run (105bp read, single end), multiplex 48 yeast samples in a single lane at an almost 30X coverage. This approach is very advantageous for researchers, especially in term of costs, but it adds an additional layer of pre-processing that is not as trivial to process as one would think. Extracting the barcodes would be fairly straightforward, but for the average 0.1 percent sequencing error rate that introduces a lot of multiplicity in the actual barcodes present in the samples. A proper design of the barcodes, maximizing the Hamming distance[Hamming, 1950, Pilcher et al., 2008] is an essential step for a proper de-multiplexing.
There are two kinds of barcoding, the one described in Lefrançois et al. [2009] where the barcode is part of the read sequence and the one developed by Illumina, where the barcode is read in a separate sequencing reaction after the first mate sequencing.
The data used in the following example has been sequenced using the Illumina protocol. We'll look at the specificities that this introduce.

Important Note: This pre-processing procedure has to be applied on the raw reads before any alignment is performed. Most often, one would use the "fastq" formatted file as input. In the particular case of the Illumina protocol, the barcodes can be retrieved from the fastq ID lines or from the Illumina export format. The export format is normally the result of aligning the reads with ELAND, the Illumina aligner, but it can be generated as well without aligning the reads, *i.e.* the export file may not contain any alignment information. As the ShortRead package offers a quick functionality to access the barcode field of an export file, we use this example to introduce it.

```
> alns <- readAligned(
+                     system.file(
```

```
+                                    "extdata",
+                                    package="RnaSeqTutorial"),
+                        pattern="multiplex_export",
+                        filter=compose(
+                          chastityFilter(),
+                          nFilter(2),
+                          chromosomeFilter(regex="chr")),
+                        type="SolexaExport",
+                        withAll=TRUE)
```

Note the use of the `withAll` argument. It is essential to get the barcode, since this data was multiplexed using the Illumina protocol. For the Illumina protocol, the barcode is read in a separate sequencing reaction and its sequence is reported as a field of the *export* file. This field is not parsed by default by the `readAligned` to save time and memory. It becomes available when the data is loaded using the`withAll` argument and is afterwards accessible in the metadata of the returned object. It can be accessed using the following command:

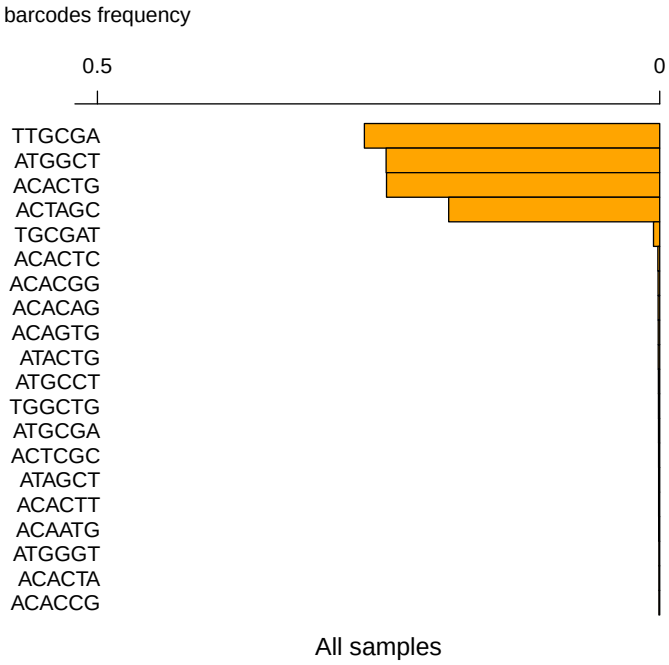
```
> alignData(alns)$multiplexIndex
```

where `alns` is the object of the *ShortRead AlignedRead* class.

In the following, we will look at the other kind of barcoding, where the barcode is part of the sequenced sequence. First, we will create some plots to evaluate the barcoding efficiency.

```
> barcodes=c("ACACTG","ACTAGC","ATGGCT","TTGCGA")
```

```
> barcodePlot(alns,
+             barcodes=barcodes,
+             type="within",
+             barcode.length=6,
+             show.barcode=20,
+             main="All samples",
+             xlim=c(0,0.5))
```



All the barcodes seem to be almost equally distributed. Every one has a proportion close to 25%. Just the "ACTAGG" seems to have been either less amplified during the library preparation or simply had a lower concentration than the other or generated less clusters. Overall, only a low percentage of them cannot be surely assigned. Once this has been verified, the sample can be *demultiplexed*.

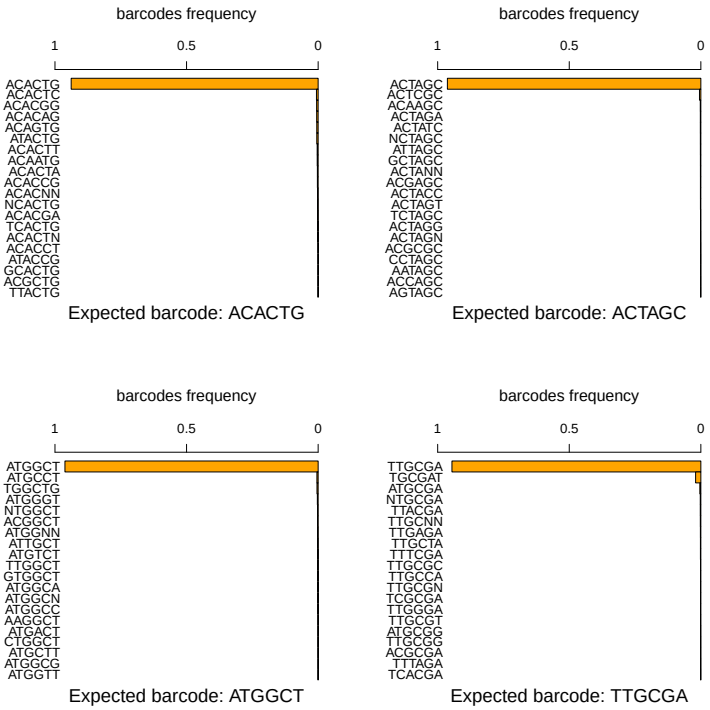
```
> dem.alns <- demultiplex(alns,
+                         barcodes=barcodes,
+                         edition.dist=2,
+                         barcodes.qty=4,
+                         type="within")

> dem.alns$reads[[1]]
> dem.alns$barcodes[[1]]
```

Remains to write the extracted data to file, or proceed it within R. Again performing some validation plot is important to ensure that the *demultiplexing* process succeeded.

```
> par(mfrow=c(2,2))
> barcode.frequencies <- lapply(
+                               names(dem.alns$barcodes),
```

```
+                                    function(barcode,alns){
+                                      barcodePlot(
+                                                  alns$barcodes[[barcode]],
+                                                  barcodes=barcode,
+                                                  type="within",barcode.length=6,
+                                                  show.barcode=20,
+                                                  main=paste(
+                                                    "Expected barcode:",
+                                                    barcode))
+                                    },dem.alns)
```



Here we see the demultiplexed barcodes. It's important to assess if within a given sample, there was a barcode bias. Generally, visually assessing the data is important to make sure that the raw data you are looking at agrees with the experimental design that was performed. This is discussed in the `RnaSeqTutorial` vignette of the *RnaSeqTutorial*.
Finally, we can save the demultiplexed files to disk.

```
> status <- lapply(
+                  names(dem.alns$barcodes),
+                  function(barcode,alns){
+                    writeFastq(
+                               alns$reads[[barcode]],
```

```
+                                   file=paste(
+                                     barcode,
+                                     "fastq",sep="."))
+                     },dem.alns)
```

The fastq files are now ready to be aligned with your preferred aligner and the resulting bam files processed with the `easyRNASeq` function as described in the first part of the present vignette! Enjoy.

6 Yet to come

The `easyRNASeq` can now return a *SummarizedExperiment* object. The aim is to have it be the only output in the next development version of the *easyRNASeq* package (version 1.5.*x*). This in order to consolidate the kind of objects used for Next-Generation Sequencing in the Bioconductor repository (rather ambitious, yes). For this, as of version 1.3.14, a new function: `count` has been introduced that supersedes the `easyRNASeq` function. An example follows, however the parameters -currently alike those of the `easyRNASeq` - will be refined in order to consolidate and structure them, so do not rely on them for writing production code.

```
> ## creating a SummarizedExperiment from 4 bam files
> sumExp <- count(filesDirectory=system.file(
+                   "extdata",
+                   package="RnaSeqTutorial"),
+                 pattern="[A,C,T,G]{6}\\.bam$",
+                 readLength=30L,
+                 organism="Dmelanogaster",
+                 chr.sizes=seqlengths(Dmelanogaster),
+                 annotationMethod="rda",
+                 annotationFile=system.file(
+                   "data",
+                   "gAnnot.rda",
+                   package="RnaSeqTutorial"),
+                 count="exons"
+                 )
> ## the counts
> assays(sumExp)
> ## the sample info
> colData(sumExp)
> ## the 'features' info
> rowData(sumExp)
>
```

See the *GenomicRange* package *SummarizedExperiment* class for more details on last three accessors used in the example.

7 Use Cases

The following use cases have been created to answer user request from the Bioconductor mailing list. We want to thank Francesco Lescai, Wade Davis and Richard Friedman for asking pertinent questions that helped us make this vignette better.
The first use case examplify how to use the `easyRNASeq` to get *DESeq* normalized data from two human samples. It introduces as well how to use the *GenomicFeatures* to retrieve annotations. Note that the data for this example is not readily available. If you want to reproduce this example, you will need to get 2 (at least) fastq files from either the SRA or ENA websites and align them against the human genome using your aligner of choice and convert the alignment into the BAM format. An example of how to achieve this in R is described in the beginning of the use case.
The second use case describe how to combine different annotation (chromosomic and genic), when for example the chromosome names in the aligned file(s) are different from the annotation retrieved using *(*biomaRt. In both use case, we will assume the *easyRNASeq* library as already been loaded as in:

```
> library(easyRNASeq)
```

Processing a set of human samples

Getting the data If you already have a set of bam files resulting of short read alignments against the human genome, you can just proceed to the next paragraph. If not here is an example on how to retrieve and align data in R. Note that the *Rsubread* is only supported on the linux platform. You'll need to use your aligner of choice and generate BAM files if you're using another platform. However you should still be able to download the SRA/ENA files. The files listed in this example are from the Jääger et al. [2012] study, but where simply selected for their small size. Still downloading and creating all the necessary file requires times and a computer with a sufficient amount of memory to index the human genome.

```
> library(BSgenome.Hsapiens.UCSC.hg19)
> library(GEOquery)
> library(SRAdb)
> library(Rsamtools)
> library(Rsubread)
> ## create a temp dir
> dir.create(file.path(getwd(),"tmp1234"))
> ## change the working directory
> setwd(file.path(getwd(),"tmp1234"))
```

```
> ## init SRA
> sqlfile <- getSRAdbFile()
> ## init a connection
> sra_con <- dbConnect(SQLite(),sqlfile)
> ## list the files
> acc <- c("SRR490224","SRR490225")
> getFASTQinfo( in_acc = acc, srcType = 'ftp' )
> ## get the read files
> getSRAfile( in_acc=acc, sra_con, destDir = getwd(),
+            fileType = 'fastq', srcType = 'ftp' )
> ## close the connection
> dbDisconnect( sra_con )
> ## write the human genome sequences
> writeXStringSet(Reduce(append,
+                        lapply(seqnames(Hsapiens),
+                               function(nam){
+                                 dss <- DNAStringSet(unmasked(Hsapiens[[nam]]))
+                                 names(dss) <- nam
+                                 dss
+                               })),
+                 file="hg19.fa")
> ## create the indexes
> dir.create("indexes")
> buildindex(basename=file.path("indexes","hg19"),
+            reference="hg19.fa")
> ## align the reads
> sapply(dir(pattern="*\\.gz$"),function(fil){
+   ## decompress the files
+   gunzip(fil)
+
+   ## align
+   align(index=file.path("indexes","hg19"),
+         readfile1=sub("\\.gz$","",fil),
+         nsubreads=2,TH1=1,
+         output_file=sub("\\.fastq\\.gz$","\\.sam",fil)
+         )
+
+   ## create bam files
+   asBam(file=sub("\\.fastq\\.gz$","\\.sam",fil),
+         destination=sub("\\.fastq\\.gz$","",fil),
+         indexDestination=TRUE)
+ })
>
```

Note that this has generated a number of files that you should clean up afterwards, i.e. delete the "tmp1234" folder, once you are done with the use case.

Processing the data First, we start by retrieveing the size of the chromosomes. This is an important information for calculating any feature count. Actually, neither the BSgenome nor any related packages are required for *easyRNASeq* to run. As they are the easiest way to access genomic information such as chromosome lengths within the R/Bioconductor framework, they are made available to the `easyRNASeq` for that purpose. However, providing a simple named vector is sufficient and therefore is easyRNASeq not limited to existing *BSgenome* organisms. The chromosome size is essential for one reason: to provide a complete representation of the data. When counting reads per features, one get counts for these features that have at least one read aligning to them, i.e. every feature having no reads will be missed. One could then either return only those features having counts or returning a value of 0 counts for those that do not. We do not find these solutions satisfying and to ensure that we provide coherent data, we return the counts for every feature present on the chromosomes. For that purpose, the chromosome size is essential as it allows us to define those features on a chromosome that are located between the last feature having a number a count bigger or equal to one and the end of the chromosome - features, which would otherwise be ignored. It is as well a mean to monitor that the provided annotation is pertinent. As of version 1.3.5, when using the 'bam' format, it is even easier. Setting the 'chr.sizes' argument to **auto** will result in the chromosome sizes to be retrieved from the BAM header. However, the purpose of this use case is to, at least partly, demonstrate the importance of using appropriate annotations and therefore the use of the 'chr.sizes' argument is demonstrated.

```
> library(BSgenome.Hsapiens.UCSC.hg19)
> chr.sizes=seqlengths(Hsapiens)
```

Then, we list the BAM files.

```
> bamfiles=dir(getwd(),pattern="*\\.bam$")
```

We can now run the `easyRNASeq` function, fetching the annotation using *biomaRt*. As this is time consuming (about 10 minutes from an average network) and since we might want to work on these annotation to avoid double-counting, we first ask the function to return an instance of the *RNAseq* class. Then, using the genomicAnnotation accessor, we extract the retrieved annotation. Note: As of version 1.3.5, 'bam' is the default format. Hence, the argument 'format' has been removed from the following function call.

```
> rnaSeq <- easyRNASeq(filesDirectory=getwd(),
+                      organism="Hsapiens",
+                      chr.sizes=chr.sizes,
+                      readLength=58L,
+                      annotationMethod="biomaRt",
+                      count="exons",
+                      filenames=bamfiles[1],
+                      outputFormat="RNAseq"
+                      )
> gAnnot <- genomicAnnotation(rnaSeq)
```

As one can see by looking at this object, it contains 434 "chromosomes", most of which are of no interest to us. For that reason, we first filter it and then save it to the disk for later re-use. The ".rda" extension is a synonym of the ".RData" one and identifies a serialized R data file.

```
> gAnnot <- gAnnot[space(gAnnot) %in% paste("chr",c(1:22,"X","Y","M"),sep=""),]
> save(gAnnot,file="gAnnot.rda")
```

Now using the modified, saved annotation, we can get a count table as follows:

```
> countTable <- easyRNASeq(filesDirectory=getwd(),
+                          organism="Hsapiens",
+                          chr.sizes=chr.sizes,
+                          readLength=58L,
+                          annotationMethod="rda",
+                          annotationFile="gAnnot.rda",
+                          count="exons",
+                          filenames=bamfiles[1]
+                          )
```

Another way to retrieve the annotation is to use the *GenomicFeatures* library. *easyRNASeq* does not yet supports that library automatically, but it can take `GRangesList` object as input; objects that are derivatives of the ones returned by the functions of the *GenomicFeatures* package. A few changes needs to be done to the obtained object so that it can be used by the `easyRNASeq`: first, a metadata element needs to be updated and then the object needs to be converted into a `GRangesList`.

```
> library(GenomicFeatures)
> hg19.tx <- makeTranscriptDbFromUCSC(
+                                     genome="hg19",
+                                     tablename="refGene")
> gAnnot <- exons(hg19.tx)
```

```
> colnames(elementMetadata(gAnnot)) <- "exon"
> gAnnot <- split(gAnnot,seqnames(gAnnot))
```

As previously, this annotation could be saved to disk or can as well be used directly, using the `annotationMethod` "env" and the `annotationObject` arguments. Additionaly, one can select chromosome of interest using the `chr.sel` argument. It accepts a vector of chromosome names. In the following, we subset for the chromosome "chr1" only.

```
> countTable <- easyRNASeq(filesDirectory=getwd(),
+                          organism="Hsapiens",
+                          readLength=58L,
+                          annotationMethod="env",
+                          annotationObject=gAnnot,
+                          count="exons",
+                          filenames=bamfiles[1],
+                          chr.sel="chr1"
+                          )
```

Note that in the previous call, we removed the 'chr.sizes' argument, just to demonstrate that the chr.sizes can be retrieved from the bam files. Removing the argument is the same as setting it to its default value: 'auto'

Alternatively, one can use the `count` function to get a *SummarizedExperiment* object back. Mind the comments in section 6, page 31 though.

```
> sumExp <- count(filesDirectory=getwd(),
+                      organism="Hsapiens",
+                      readLength=58L,
+                      annotationMethod="env",
+                      annotationObject=gAnnot,
+                      count="exons",
+                      filenames=bamfiles[1],
+                      chr.sel="chr1"
+                      )
```

Finally, instead of returning a count table, we can get a `CountDataSet` instance from the *DESeq* package. This will perform the normalization of the data and generate some Quality Assessment plots. In the present example, it would not yield very sensitive results as we have no replicates (biological). These are important to *DESeq* to accurately model the technical and biological variance. With no replicates for every condition, the dispersion will be based on a "pooled" estimate making the differential expression call lose sensitivity. In addition, *DESeq* is in such cases using a conservative approach (which is good) so you'd get even less significant results. Here, as we have no replicates, we need to pass additional arguments (the last two)

that will be tunnelled to the *DESeq* `estimateDispersions` function. See the *DESeq* vignette for more details on these.

```
> conditions <- c("A","B")
> names(conditions) <- bamfiles
> countDataSet <- easyRNASeq(filesDirectory=getwd(),
+                            organism="Hsapiens",
+                            chr.sizes=chr.sizes,
+                            annotationMethod="env",
+                            annotationObject=gAnnot,
+                            count="exons",
+                            filenames=bamfiles,
+                            chr.sel="chr1",
+                            outputFormat="DESeq",
+                            normalize=TRUE,
+                            conditions=conditions,
+                            fitType="local",
+                            method="blind"
+                           )
```

Note that as the read length differs between the two files: 58 and 76, the `readLength` argument was removed from the previous function call. The read size is then identified automatically.

Dealing with annotation inconsistencies This use case shows how to deal with inconsistent annotations, *e.g.* when the chromosome names present in the aligned file are different from those that can be retrieved from an annotation source such as *biomaRt*.
First, we have a look at the data, in this case some Illumina export files. Reading in the data using the *ShortRead* package is quite resource demanding as the whole sequences are loaded in memory. Then we look at the chromosome names. These differs from what we expect - UCSC standards - as they have an additional ".fa" extension.

```
> aln <- readAligned("data",type="SolexaExport",pattern="*.txt.gz")
> gc()
> levels(chromosome(aln))

[1] "chr1.fa"  "chr10.fa" "..."      "chrY.fa"
```

They are different from what *biomaRt* will return as well: *i.e.* 1:19, X, Y and MT plus others. This triple inconsistency will be a problem for *easyRNASeq*. If there were only two sets of names, using the "custom" chromosome name map by-pass (see the Details section of the "?easyRNASeq" help page) would solve the issue. However, in the present particular case, as

we are retrieving the annotation from *biomaRt*, we need to precise the name of the organism, which circumvent the chromosome name mapping by-pass. The solution is to first fetch the annotation, modify it and save it as an R data file. Some of the retrieved annotation are "NT" contigs. There are only a few of them, so instead of filtering them out, we just ignore them.

```
> obj <- fetchAnnotation(new('RNAseq',
+                            organismName="Mmusculus"
+                            ),
+                        method="biomaRt")
> gAnnot <- genomicAnnotation(obj)
> length(grep("NT_",space(gAnnot)))
>
[1] 1181
> names(gAnnot) <- paste("chr",names(gAnnot),".fa",sep="")
```

As described previously, see page 35 we can save the annotation to a file or use it directly, using the `AnnotationMethod` "env" argument. In that later case, since we did not process the annotation, numerous warnings concerning possible multiple countings of reads will be raised. Double counting reads is not what ones want, that is why the better way is to rework the obtained annotation to remove/clarify these cases and save the obtained annotation as an rda file on your file system.
Note that *easyRNASeq* does support some chromosome names conversion by default. The list of organism for which this is possible can be listed using the knownOrganisms function.
Before going on, we do some cleanup as some of the objects we have generated take large amounts of memory.

```
> rm(aln,obj)
> gc()
```

As on page 34 we get the chromosome sizes. Again, note that the use of the BSgenome is not mandatory. It's just easy as they are available in Bioconductor. Typing in your own chromosome sizes named vector is as valid.

```
> library(BSgenome.Mmusculus.UCSC.mm9)
> chr.sizes<- seqlengths(Mmusculus)
```

Note that now, providing the chromosome sizes as a list is deprecated. It has been changed to a named vector, which is more intuitive. The list was an historical remnant of the RangedData API. Note as well that the chr.sizes will be necessary here as we are using 'aln' files and not 'bam' files.
We can now create the chromosome name mapping.

```
> chr.map <- data.frame(
+                       from=paste("chr",c(1:19,"X","Y"),".fa",sep=""),
+                       to=paste("chr",c(1:19,"X","Y"),sep=""))
```

Using this chromosome map, we can now summarize the reads per feature of interest. Here we want to look for gene models, so we set the `count` and `summarization` argument. Note again that the `summarization` argument will be deprecated in the near future and its values merged with the `count` ones.
Asking to get back an *RNAseq* instance allows us to look at the gene models defined by *easyRNASeq*. This offers the possibility to clean them to avoid multiple counting. An additional use case describing how to do this will be introduced soon.
As we are provided with Illumina export data, we need to define a set of Filter to ensure that the data is read properly. Indeed, the export file contains all the reads, so the one that do not pass the chastity filter have to be removed. In addition, some of the other reads are for internal QC, and they have no position. For that reason, we need to filter those too out.
Reading in export data is more resource exhausting that reading in bam files, as we are loading in the sequence and quality information as well, whereas we do not need them.
We will now look through three different set of parameters that will stepwise reduce the number of warnings emitted. These warnings are there to help you understand the different pitfalls that the `easyRNASeq` helps you avoiding when analysing your RNA-Seq data. The first approach generates a lot of warnings, because of the differing annotations, since we are using the entire set of annotation we got. As we do not want to generate all these warnings, these code lines are not evaluated. To get a feel about these warnings, they will look as the follows:

```
Warning messages:
1: In .convertToUCSC(names(genomicAnnotation(obj)), organismName(obj),  :
Your custom map does not define a mapping for the following
chromosome names: chrMT.fa
2: In easyRNASeq(filesDirectory = headDir, organism = "custom", chr.map = chr.map,
There are 6096 synthetic exons as determined from your annotation that overlap!
This implies that some reads will be counted more than once!
Is that really what you want?
```

Let us start with the full set of annotation.

```
> rnaSeq <- easyRNASeq(filesDirectory="data",
+                      organism="custom",
+                      chr.map=chr.map,
+                      chr.sizes=chr.sizes,
```

```
+                     filter=compose(
+                       naPositionFilter(),
+                       chastityFilter()),
+                     readLength=50L,
+                     annotationMethod="env",
+                     annotationObject=gAnnot,
+                     format="aln",
+                     count="genes",
+                     summarization= "geneModels",
+                     filenames="1-Feb_ATCACG_L003_R1_001_export.txt.gz",
+                     outputFormat="RNAseq",
+                     nbCore=2
+                     )
```

To reduce the number of warnings emitted, we can select for the chromosome we are interested in as previously done on page 36.

```
> rnaSeq <- easyRNASeq(filesDirectory="data",
+                     organism="custom",
+                     chr.map=chr.map,
+                     chr.sizes=chr.sizes,
+                     chr.sel=chr.map$from,
+                     filter=compose(
+                       naPositionFilter(),
+                       chastityFilter()),
+                     readLength=50L,
+                     annotationMethod="env",
+                     annotationObject=gAnnot,
+                     format="aln",
+                     count="genes",
+                     summarization= "geneModels",
+                     filenames="1-Feb_ATCACG_L003_R1_001_export.txt.gz",
+                     outputFormat="RNAseq",
+                     nbCore=2
+                     )
```

Finally, to further reduce the warnings, we can manipulate the `RangedData` object to remove the unnecessary annotation.

```
> sel <- grep("NT_",names(gAnnot))
> gAnnot <- RangedData(ranges=ranges(gAnnot)[-sel,],values=values(gAnnot)[-sel,])
> colnames(gAnnot) <- gsub("values\\.","",colnames(gAnnot))
```

This last call will only generate two warnings, one that could be easily dealt with (a complaint about the chrMT). The other one is about double counting and it requires to adapt the annotation.

```
> rnaSeq <- easyRNASeq(filesDirectory="data",
+                      organism="custom",
+                      chr.map=chr.map,
+                      chr.sizes=chr.sizes,
+                      chr.sel=chr.map$from,
+                      filter=compose(
+                        naPositionFilter(),
+                        chastityFilter()),
+                      readLength=50L,
+                      annotationMethod="env",
+                      annotationObject=gAnnot,
+                      format="aln",
+                      count="genes",
+                      summarization= "geneModels",
+                      filenames="1-Feb_ATCACG_L003_R1_001_export.txt.gz",
+                      outputFormat="RNAseq",
+                      nbCore=2
+                      )
```

8 Session Information

The version number of R[R Development Core Team, 2009] and Bioconductor [Gentleman et al., 2004] packages loaded for generating the vignette were:

```
R version 3.0.1 (2013-05-16)
Platform: i386-w64-mingw32/i386 (32-bit)

locale:
[1] LC_COLLATE=C
[2] LC_CTYPE=English_United States.1252
[3] LC_MONETARY=English_United States.1252
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.1252

attached base packages:
[1] parallel  stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] BSgenome.Dmelanogaster.UCSC.dm3_1.3.19
 [2] RnaSeqTutorial_0.0.12
 [3] easyRNASeq_1.6.4
 [4] ShortRead_1.18.0
 [5] latticeExtra_0.6-26
 [6] Rsamtools_1.12.4
 [7] LSD_2.5
 [8] ellipse_0.3-8
 [9] schoolmath_0.4
[10] colorRamps_2.3
[11] RColorBrewer_1.0-5
[12] gtools_3.1.0
[13] MASS_7.3-29
[14] DESeq_1.12.1
[15] lattice_0.20-23
[16] locfit_1.5-9.1
[17] BSgenome_1.28.0
[18] GenomicRanges_1.12.5
[19] Biostrings_2.28.0
[20] IRanges_1.18.4
[21] edgeR_3.2.4
[22] limma_3.16.8
[23] biomaRt_2.16.0
```

```
[24] Biobase_2.20.1
[25] genomeIntervals_1.16.0
[26] BiocGenerics_0.6.0
[27] intervals_0.14.0

loaded via a namespace (and not attached):
 [1] AnnotationDbi_1.22.6 DBI_0.2-7            RCurl_1.95-4.1
 [4] RSQLite_0.11.4       XML_3.98-1.1         annotate_1.38.0
 [7] bitops_1.0-6         genefilter_1.42.0    geneplotter_1.38.0
[10] grid_3.0.1           hwriter_1.3          splines_3.0.1
[13] stats4_3.0.1         survival_2.37-4      tools_3.0.1
[16] xtable_1.7-1         zlibbioc_1.6.0
```

9 Final remarks

RNA-seq is still maturating and a lot of new developments are to be expected. If you have any questions, comments, feel free to contact me: delhomme *at* embl *dot* de.

References

Simon Anders and Wolfgang Huber. Differential expression analysis for sequence count data. *Genome Biology 2010 11:202*, 11(10):R106, Oct 2010.

Nicolas Delhomme, Ismaël Padioleau, Eileen E Furlong, and Larsm Steinmetz. easyrnaseq: a bioconductor package for processing rna-seq data. *Bioinformatics*, Jul 2012. doi: 10.1093/bioinformatics/bts477.

Steffen Durinck et al. Biomart and bioconductor: a powerful link between biological databases and microarray data analysis. *Bioinformatics*, 21(16): 3439–40, Aug 2005.

Paul Flicek et al. Ensembl 2011. *Nucleic Acids Research*, 39(Database issue): D800–6, Jan 2011.

Robert C Gentleman et al. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology 2010 11:202*, 5(10):R80, Jan 2004.

RW Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

Kersti Jääger, Saiful Islam, Pawel Zajac, Sten Linnarsson, and Toomas Neuman. Rna-seq analysis reveals different dynamics of differentiation of human dermis- and adipose-derived stromal stem cells. *PLoS ONE*, 7(6): e38833, Jan 2012.

Philippe Lefrançois et al. Efficient yeast chip-seq using multiplex short-read dna sequencing. *BMC Genomics*, 10:37, Jan 2009.

Heng Li et al. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–9, Aug 2009.

Martin Morgan et al. Shortread: a bioconductor package for input, quality assessment and exploration of high-throughput sequence data. *Bioinformatics*, 25(19):2607–8, Oct 2009.

Ali Mortazavi et al. Mapping and quantifying mammalian transcriptomes by rna-seq. *Nature Methods*, 5(7):621–8, Jul 2008.

CD Pilcher et al. Inferring hiv transmission dynamics from phylogenetic sequence relationships. *PLoS Med*, 5(3):e69, 2008.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

Mark D Robinson et al. edger: a bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26(1): 139–40, Jan 2010.

Andrew M Smith, Lawrence E Heisler, Robert P St Onge, Eveline Farias-Hesson, Iain M Wallace, John Bodeau, Adam N Harris, Kathleen M Perry, Guri Giaever, Nader Pourmand, and Corey Nislow. Highly-multiplexed barcode sequencing: an efficient method for parallel analysis of pooled samples. *Nucleic Acids Research*, May 2010.

C Trapnell et al. Tophat: discovering splice junctions with rna-seq. *Bioinformatics*, 25(9):1105–1111, May 2009.

Susan Tweedie et al. Flybase: enhancing drosophila gene ontology annotations. *Nucleic Acids Research*, 37(Database issue):D555–9, Jan 2009.