Event-level prediction and quantification of transcript isoforms from RNA-seq data

Leonard Goldstein

Department of Bioinformatics and Computational Biology, Genentech Inc.

November 18, 2014

1 Background

RNA-seq data are informative for the analysis of known and novel transcript isoforms. While the short length of RNA-seq reads limits the ability to predict and quantify full-length transcripts, short read data are well suited for the analysis of individual alternative transcript events (e.g. inclusion or skipping of a cassette exon). Available event-centric methods typically rely on annotated transcripts and only consider a subset of all possible events. We developed a novel approach for the identification and quantification of alternative transcript events from RNA-seq data, implemented in the *SGSeq* package.

2 Overview

SGSeq predicts splice junctions and exons from genomic RNA-seq read alignments in BAM format. The discrete transcript features are assembled into a genome-wide splice graph [1]. Splice junctions and disjoint exon bins form the edges of the graph, while nodes correspond to transcript starts and ends, and splice donor and acceptor sites. Alternative transcript events are regions with two or more transcript variants. In the context of the splice graph, they are defined by a start and an end node connected by two or more alternative paths and no intervening nodes with all paths intersecting. *SGSeq* identifies alternative transcript events recursively from the splice graph, and quantifies transcript variants locally, based on counts of reads spanning event boundaries.

3 Preliminaries

This vignette illustrates an analysis of paired-end RNA-seq data obtained from four colorectal tumors and four normal colorectal samples, which are part of a data set published in [2]. For the purpose of this vignette, we created BAM files including alignments overlapping a single gene of interest (*FBXO31*).

```
library(SGSeq)
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
```

In the following, we use a *data.frame* `si` with sample information, and a *GRanges* object `gr` with genomic coordinates of the *FBXO31* gene.

The *data.frame* with sample information contains alignment information, including paired-end status, median read length, median insert size and the total number of alignments. These were obtained from the original BAM files using function `getBamInfo`. We set the correct BAM file paths in the sample information.

```
dir <- system.file("extdata", package = "SGSeq")
si$file_bam <- file.path(dir, "bams", si$file_bam)
```

We obtain transcript annotation from the UCSC knownGene table, available as a *Bioconductor* annotation package *TxDb.Hsapiens.UCSC.hg19.knownGene*. We retain transcripts on chromosome 16, where the *FBXO31* gene is located, and change chromosome names in the annotation to match chromosome names in the BAM files.

```
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
txdb <- keepSeqlevels(txdb, "chr16")
seqlevelsStyle(txdb) <- "NCBI"
```

SGSeq makes extensive use of the *Bioconductor* infrastructure for genomic ranges [3]. To store genomic coordinates for both exons and splice junctions, we created a new class *TxFeatures*, which extends the *GRanges* class with additional columns. Column `type` takes values in `J` (splice junction), `I` (internal exon), `F` (first/$5'$ terminal exon), `L` (last/$3'$ terminal exon) and `U` (unspliced).

In addition to *TxFeatures*, we designed the *SGFeatures* class to store splice graph features. Similar to *TxFeatures*, the *SGFeatures* class extends the *GRanges* class with additional columns. Column `type` in an *SGFeatures* object takes values in `J` (splice junction), `E` (disjoint exon bin), `D` (splice donor) and `A` (splice acceptor).

For both *TxFeatures* and *SGFeatures*, additional column data can be accessed using functions that are named after the columns they access (e.g. use function `type` to obtain feature type). Transcript features or splice graph features can be exported to BED files using function `exportFeatures`.

To work with annotated transcripts in the *SGSeq* framework, we extract transcript features from the *TxDb* object and store them as a *TxFeatures* object. We only retain features overlapping the *FBXO31* gene locus.

```
txf_annotated <- convertToTxFeatures(txdb)
txf_annotated <- txf_annotated[txf_annotated %over% gr]
```

If transcript annotation is not available as a *TxDb* object, function `convertToTxFeatures` can construct *TxFeatures* from a *GRangesList* of exons grouped by transcript (which can be obtained from other formats such as GFF/GTF).

4 Analysis based on annotated transcript features

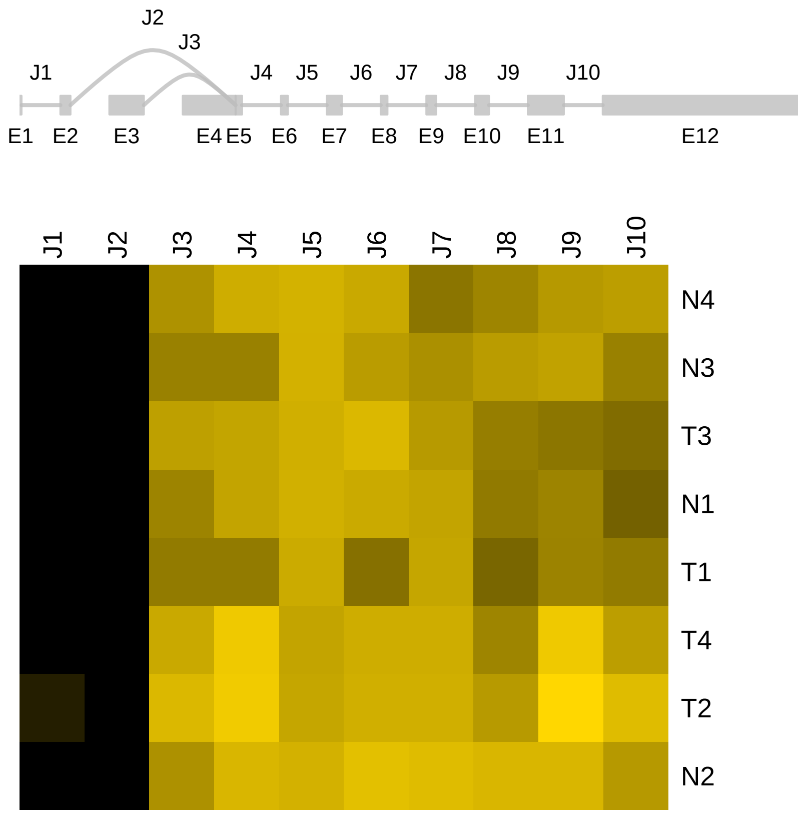
Initially, we perform an analysis for annotated transcript features. The following example converts the transcript features into splice graph features and obtains compatible counts for each feature and each sample.

```
sgfc <- analyzeFeatures(si, features = txf_annotated)

## Process features...
## Obtain counts...
```

`analyzeFeatures` returns an *SGFeatureCounts* object, which extends the *SummarizedExperiment* class from the *GenomicRanges* package. *SGFeatureCounts* contains sample information as `colData`, splice graph features as `rowData` and assays `counts` and `FPKM`, which store compatible counts and FPKMs for each splice graph feature and sample, respectively. Accessor functions `colData`, `rowData`, `counts` and `FPKM` can be used to access the data.

Compatible FPKMs for splice graph features can be visualized with function `plotFeatures`.

```
plotFeatures(sgfc, geneID = 1)
```



5 Analysis based on predicted transcript features

Instead of relying on existing annotation, we can predict transcript features from BAM files directly.

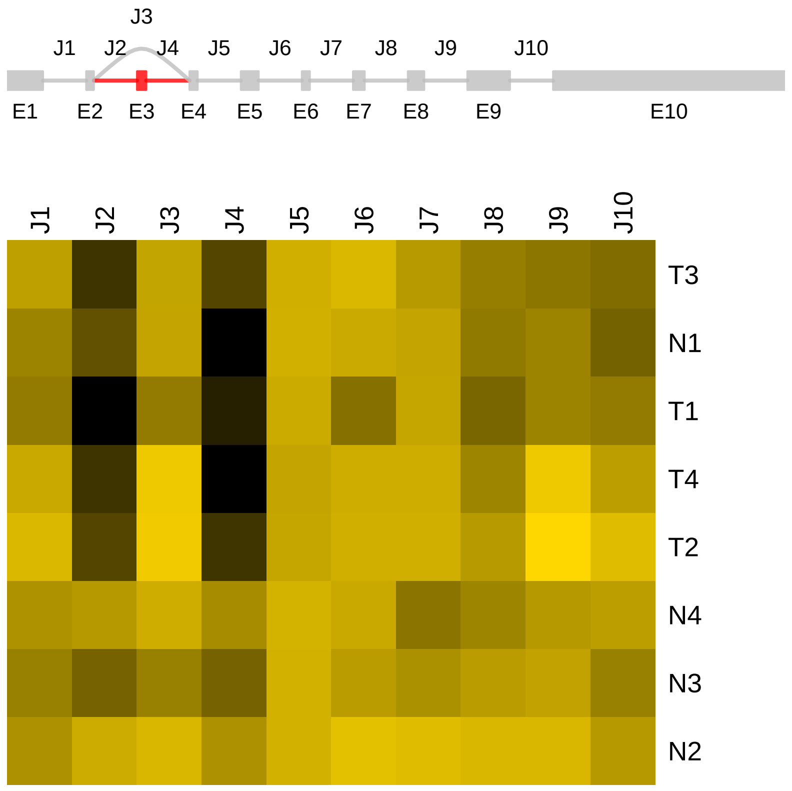
```
sgfc <- analyzeFeatures(si, which = gr)

## Predict features...
## Process features...
## Obtain counts...
```

For interpretability, we annotate predicted features with respect to known transcript features.

```
sgfc <- annotate(sgfc, txf_annotated)
```

The predicted splice graph features and compatible FPKMs can be visualized as previously. By default, splice graph features with missing annotation are highlighted in red.

```
plotFeatures(sgfc, geneID = 1)
```



Note that, in contrast to the previous figure, the predicted gene model does not include parts of the splice graph that are not expressed. Also, an unannotated exon was discovered from the RNA-seq data, which is expressed in three of the four normal colorectal samples.

6 Analysis of transcript variants

We can focus our analysis on alternative transcript events. The following example identifies transcript variants from the splice graph and obtains representative counts for each transcript variant. Estimates for variant frequencies are obtained based on representative counts.

```
txvc <- analyzeVariants(sgfc)

## Find segments...
## Find variants...
## Annotate variants...
```

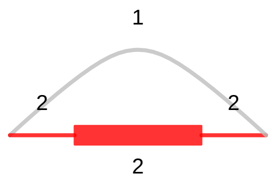
analyzeVariants returns a *TxVariantCounts* object. Similar to *SGFeatureCounts*, *TxVariantCounts* extends the *SummarizedExperiment* class from the *GenomicRanges* package. *TxVariantCounts* contains sample information as colData and transcript variants as rowData. Assay variantFreq stores frequency estimates for each transcript variant and sample. Accessor functions colData, rowData and variantFreq can be used to access the data.

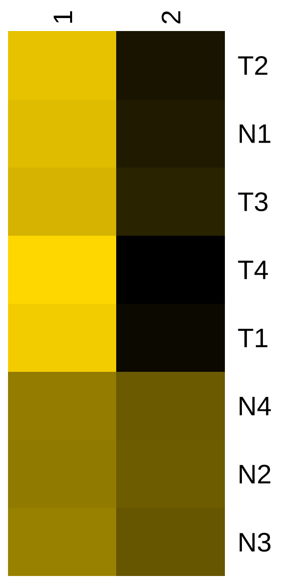
Each transcript variant consists of one or more splice graph features. Information on transcript variants is stored as elementMetadata (or mcols) in the *TxVariants* object and can be accessed as follows.

```
mcols(txvc)

## DataFrame with 2 rows and 16 columns
##               from               to        type   featureID   segmentID  closed3p  closed5p
##        <character>      <character> <character> <character> <character> <logical> <logical>
## 1 D:16:87393901:- A:16:87380856:-           J          28           4      TRUE      TRUE
## 2 D:16:87393901:- A:16:87380856:-         JEJ    32,30,27           2      TRUE      TRUE
##      geneID   eventID variantID   featureID5p   featureID3p
##   <integer> <integer> <integer> <IntegerList> <IntegerList>
## 1         1         1         1            28            28
## 2         1         1         2            32            27
##                              txName        geneName     variantType    variantName
##                      <CharacterList> <CharacterList> <CharacterList>    <character>
## 1 uc002fjv.3,uc002fjw.3,uc010vot.2           79791            SE:S 79791_1_1/2_SE
## 2                                            79791            SE:I 79791_1_2/2_SE
```

Transcript variants and estimates of variant frequencies can be visualized with function `plotVariants`.

```
plotVariants(txvc, eventID = 1)
```





7 Advanced use

Functions `analyzeFeatures` and `analyzeVariants` wrap multiple analysis steps for convenience. Alternatively, the functions performing individual steps can be called directly. The previous analysis based on predicted transcript features can be performed as follows.

```
txf <- predictTxFeatures(si, gr)
sgf <- convertToSGFeatures(txf)
sgf <- annotate(sgf, txf_annotated)
sgfc <- getSGFeatureCounts(si, sgf)
txv <- findTxVariants(sgf)

## Find segments...
## Find variants...
## Annotate variants...

txvc <- getTxVariantCounts(sgfc, txv)
```

Feature prediction and counting (with `predictTxFeatures` and `getSGFeatureCounts`, respectively) can be performed for individual samples, and results can be combined at a later point in time (e.g. to distribute samples across a high-performance computing cluster).

Note that `predictTxFeatures` predicts features for each sample, merges features across samples and finally performs filtering and processing of predicted terminal exons. When using `predictTxFeatures` for individual samples, with predictions intended to be merged later, run `predictTxFeatures` with argument `min_overhang = NULL` to suppress processing of terminal exons. Then predictions can subsequently be merged and processed with functions `mergeTxFeatures` and `processTerminalExons`, respectively.

8 Performance

When performing genome-wide analyses or working with large data sets, parallelization is highly recommended. For functions `analyzeFeatures`, `predictTxFeatures` and `getSGFeatureCounts`, parallelization across samples is controlled with argument `BPPARAM`. It defaults to `MulticoreParam(workers = 1)` (no parallelization). For analyses run on a single node with multiple cores, the number of samples processed in parallel can be specified with argument `workers`. For more details, please see the documentation for the *BiocParallel* package. The number of cores used per samples, enabling parallel processing of multiple chromosomes/strands, is specified with argument `cores_per_sample`. Processing a single BAM file with ~ 50 million paired-end reads using 4 cores typically takes $\sim 2-3$ hours for prediction and $\sim 1-2$ hours for counting. Processing times can be strongly affected by individual genes or genomic regions with many alignments. For some data sets, it may be beneficial to exclude regions with high coverage (e.g. the mitochondrial chromosome). Identification of transcript variants from the splice graph is performed on a per-gene basis and can be parallelized by setting argument `cores` when using `analyzeVariants` or `findTxVariants`.

9 Session information

- R version 3.1.2 (2014-10-31), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.28.1, Biobase 2.26.0, BiocGenerics 0.12.1, BiocParallel 1.0.0, GenomeInfoDb 1.2.3, GenomicFeatures 1.18.2, GenomicRanges 1.18.3, IRanges 2.0.0, S4Vectors 0.4.0, SGSeq 1.0.6, TxDb.Hsapiens.UCSC.hg19.knownGene 3.0.0, XVector 0.6.0, knitr 1.8
- Loaded via a namespace (and not attached): BBmisc 1.8, BatchJobs 1.5, BiocStyle 1.4.1, Biostrings 2.34.0, DBI 0.3.1, GenomicAlignments 1.2.1, RCurl 1.95-4.3, RSQLite 1.0.0, Rsamtools 1.18.2, XML 3.98-1.1, base64enc 0.1-2, biomaRt 2.22.0, bitops 1.0-6, brew 1.0-6, checkmate 1.5.0, codetools 0.2-9, digest 0.6.4, evaluate 0.5.5, fail 1.2, foreach 1.4.2, formatR 1.0, highr 0.4, igraph 0.7.1, iterators 1.0.7, rtracklayer 1.26.2, sendmailR 1.2-1, stringr 0.6.2, tools 3.1.2, zlibbioc 1.12.0

References

[1] Steffen Heber, Max Alekseyev, Sing-Hoi Sze, Haixu Tang, and Pavel A Pevzner. Splicing graphs and EST assembly problem. *Bioinformatics (Oxford, England)*, 18 Suppl 1:S181–8, 2002.

[2] Somasekar Seshagiri, Eric W Stawiski, Steffen Durinck, Zora Modrusan, Elaine E Storm, Caitlin B Conboy, Subhra Chaudhuri, Yinghui Guan, Vasantharajan Janakiraman, Bijay S Jaiswal, Joseph Guillory, Connie Ha, Gerrit J P Dijkgraaf, Jeremy Stinson, Florian Gnad, Melanie A Huntley, Jeremiah D Degenhardt, Peter M Haverty, Richard Bourgon, Weiru Wang, Hartmut Koeppen, Robert Gentleman, Timothy K Starr, Zemin Zhang, David A Largaespada, Thomas D Wu, and Frederic J de Sauvage. Recurrent R-spondin fusions in colon cancer. *Nature*, pages 1–8, August 2012.

[3] Michael Lawrence, Wolfgang Huber, Hervé Pagès, Patrick Aboyoun, Marc Carlson, Robert Gentleman, Martin T Morgan, and Vincent J Carey. Software for Computing and Annotating Genomic Ranges. *PLoS Computational Biology*, 9(8):e1003118, August 2013.