# Splice event detection and quantification from RNA-seq data

Leonard D Goldstein

Department of Bioinformatics and Computational Biology, Genentech Inc.

June 11, 2015

## 1 Background

RNA-seq data can be used for transcript isoform discovery and transcript-level expression studies. Most computational tools for genome-guided transcript isoform analysis (methods that work with RNA-seq data aligned against a reference genome) fall into one of two categories: Methods for the analysis of splice events, which often rely on transcript annotation, and methods for reconstructing and quantifying full-length transcripts. *SGSeq* implements a method for the annotation-free detection and quantification of splice events from RNA-seq data.

## 2 Overview

*SGSeq* predicts splice junctions and exons from RNA-seq reads aligned against a reference genome. Splice junctions and disjoint exon bins are quantified using counts or FPKMs based on structurally compatible reads. Input data must be in BAM format. It is essential that BAM files are obtained using a splice-aware alignment program that generates the custom tag 'XS' indicating the direction of transcription for spliced reads. Splice junctions and disjoint exon bins are assembled into a genome-wide splice graph [1]. In the *SGSeq* framework, the splice graph is a directed acyclic graph with nodes corresponding to transcript starts, ends and splice sites, and edges corresponding to disjoint exon bins and splice junctions, directed from $5'$ to the $3'$ end. A splice event is defined by a start node and an end node connected by two or more paths and no intervening nodes with all paths intersecting. *SGSeq* identifies splice events recursively from the graph, and estimates relative usage of splice variants based on compatible reads spanning the 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 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* `si` contains library 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()`. When analyzing a new data set, library information must be obtained once initially and can then be used for all subsequent analyses. The following code block sets the correct BAM file paths in the sample information for this vignette.

```
path <- system.file("extdata", package = "SGSeq")
si$file_bam <- file.path(path, "bams", si$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, *SGSeq* implements the *TxFeatures* class, which extends the *GRanges* class with additional columns. Column `type` takes values `J` (splice junction), `I` (internal exon), `F` (first/$5'$ terminal exon), `L` (last/$3'$ terminal exon) or `U` (unspliced).

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

For both *TxFeatures* and *SGFeatures*, class-specific columns can be accessed using functions 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 *TxFeatures*. We only retain features overlapping the *FBXO31* gene locus.

```
txf_annotated <- convertToTxFeatures(txdb)

## Warning in convertToTxFeatures(txdb):  Merged adjacent exons

txf_annotated <- txf_annotated[txf_annotated %over% gr]
```

If transcript annotation is not available as a *TxDb* object, `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

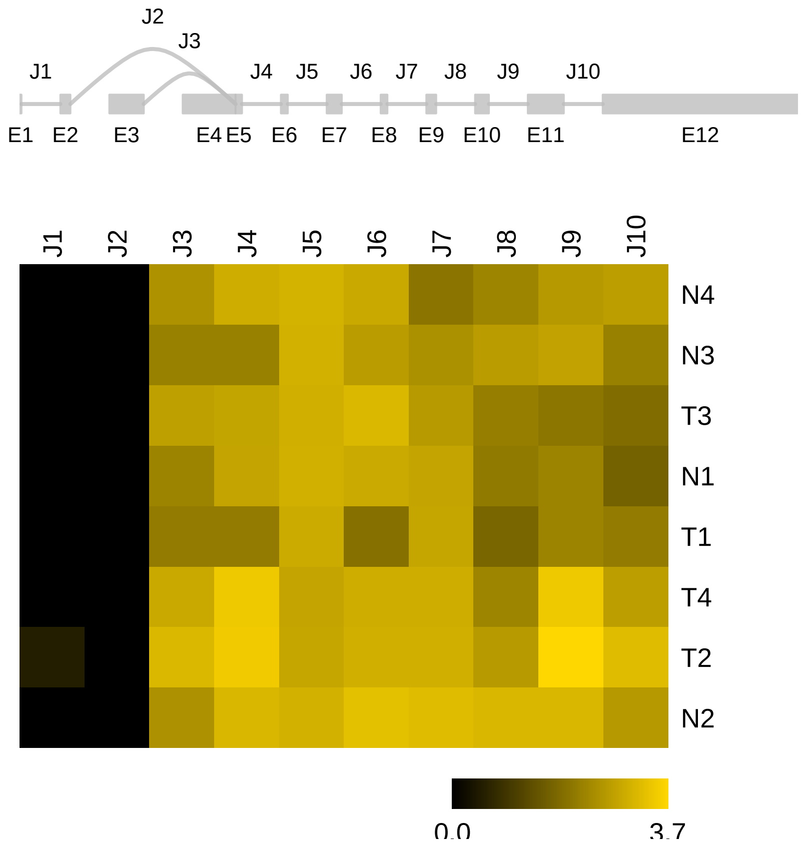
We first perform an analysis based on annotated transcript features. The following example converts the transcript features into splice graph features and obtains compatible counts for each feature and each sample.

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

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

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

Compatible FPKMs for splice graph features can be visualized with `plotFeatures()`. The plotting function invisibly returns a `data.frame` with information on splice graph features, including genomic coordinates.

```
df <- plotFeatures(sgfc, geneID = 1)
df

##     id                        name featureID
## 1   E1 E:16:87425689-87425708:-        42
## 2   E2 E:16:87423343-87423454:-        38
## 3   E3 E:16:87417011-87417394:-        35
## 4   E4 E:16:87393973-87394561:-        33
## 5   E5 E:16:87393901-87393972:-        29
## 6   E6 E:16:87380780-87380856:-        25
## 7   E7 E:16:87377204-87377371:-        21
## 8   E8 E:16:87376483-87376557:-        17
## 9   E9 E:16:87369761-87369870:-        13
## 10 E10 E:16:87368910-87369063:-         9
## 11 E11 E:16:87367492-87367892:-         5
## 12 E12 E:16:87362942-87365116:-         1
## 13  J1 J:16:87423454-87425689:-        40
## 14  J2 J:16:87393972-87423343:-        32
## 15  J3 J:16:87393972-87417011:-        31
## 16  J4 J:16:87380856-87393901:-        27
## 17  J5 J:16:87377371-87380780:-        23
## 18  J6 J:16:87376557-87377204:-        19
## 19  J7 J:16:87369870-87376483:-        15
## 20  J8 J:16:87369063-87369761:-        11
## 21  J9 J:16:87367892-87368910:-         7
## 22 J10 J:16:87365116-87367492:-         3
```



## 5 Analysis based on predicted transcript features

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

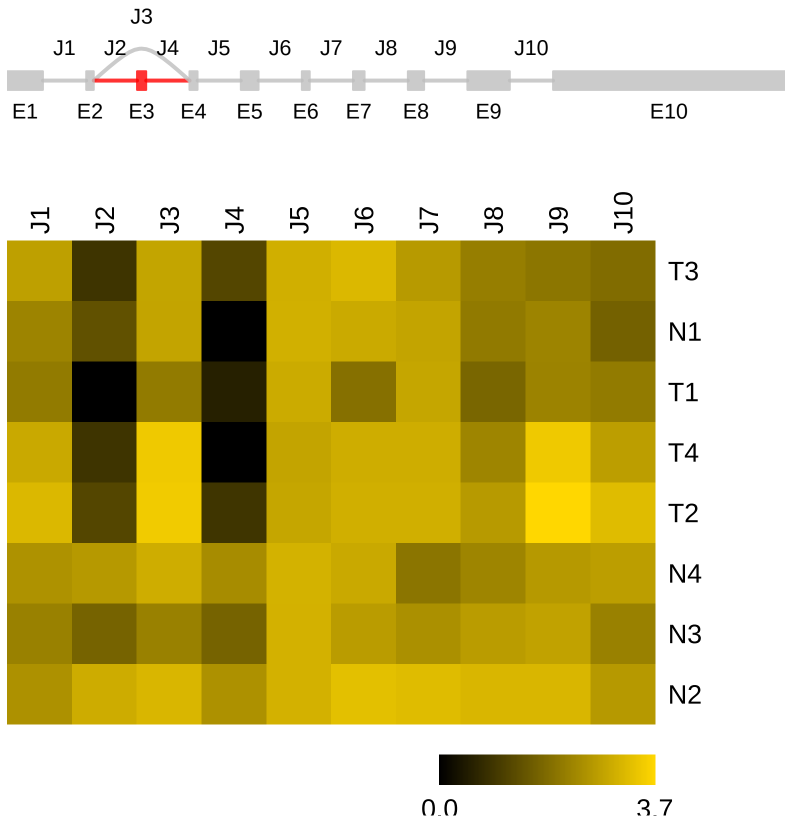
```
sgfc <- analyzeFeatures(si, which = gr)

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

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

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

Predicted splice graph features and compatible FPKMs can be visualized as previously. Splice graph features with missing annotation can be highlighted using argument `color_novel`.

```
df <- plotFeatures(sgfc, geneID = 1, color_novel = "red")
df

##      id                     name featureID
## 1    E1 E:16:87417011-87417348:-        38
## 2    E2 E:16:87393901-87393972:-        34
## 3    E3 E:16:87392017-87392103:-        30
## 4    E4 E:16:87380780-87380856:-        25
## 5    E5 E:16:87377204-87377371:-        21
## 6    E6 E:16:87376483-87376557:-        17
## 7    E7 E:16:87369761-87369870:-        13
## 8    E8 E:16:87368910-87369063:-         9
## 9    E9 E:16:87367492-87367892:-         5
## 10 E10 E:16:87362930-87365116:-         1
## 11   J1 J:16:87393972-87417011:-        36
## 12   J2 J:16:87392103-87393901:-        32
## 13   J3 J:16:87380856-87393901:-        28
## 14   J4 J:16:87380856-87392017:-        27
## 15   J5 J:16:87377371-87380780:-        23
## 16   J6 J:16:87376557-87377204:-        19
## 17   J7 J:16:87369870-87376483:-        15
## 18   J8 J:16:87369063-87369761:-        11
## 19   J9 J:16:87367892-87368910:-         7
## 20  J10 J:16:87365116-87367492:-         3
```



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 splice variants

Instead of considering the complete splice graph of a gene, we can focus our analysis on individual splice events. The following example identifies splice events from the splice graph and obtains representative counts for each splice variant.

```
sgvc <- analyzeVariants(sgfc)

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

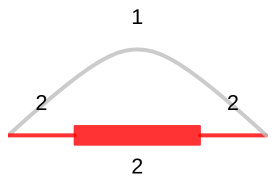
analyzeVariants() returns an *SGVariantCounts* object. Similar to *SGFeatureCounts*, *SGVariantCounts* extends the *SummarizedExperiment* class. *SGVariantCounts* contains sample information as colData and splice variants as rowRanges. Assay variantFreq stores estimates of relative usage for each splice variant and sample. Accessor functions colData(), rowRanges() and variantFreq() can be used to access the data.

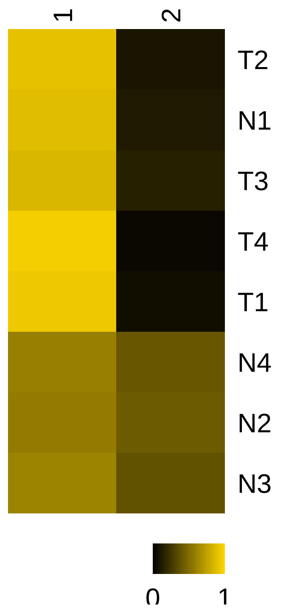
Each splice variant consists of one or more splice graph features. Information on splice variants is stored as elementMetadata (or mcols) in the *SGVariants* object and can be accessed as follows.

```
mcols(sgvc)

## 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
```

Splice variants and estimates of relative usage can be visualized with function `plotVariants()`.

```
plotVariants(sgvc, eventID = 1, color_novel = "red")
```





## 7 Advanced use

Functions `analyzeFeatures()` and `analyzeVariants()` wrap multiple analysis steps for convenience. Alternatively, the functions performing individual steps can be called directly. For example, 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)
```

```r
sgfc <- getSGFeatureCounts(si, sgf)
sgv <- findSGVariants(sgf)

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

sgvc <- getSGVariantCounts(sgv, sgfc)
```

`predictTxFeatures()` and `getSGFeatureCounts()` can be run on individual samples (e.g. for distribution across a high-performance computing cluster). `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 at a later point in time, 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. Functions `getBamInfo()`, `predictTxFeatures()` and `getSGFeatureCounts()` support parallel processing of multiple samples. `predictTxFeatures()` and `getSGFeatureCounts()` also support parallel processing of multiple chromosomes/strands for a given sample. Parallization is controlled with argument `cores`. Running a single BAM file with $\sim 50$ 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 affected by individual genes or genomic regions with many read alignments (e.g. the mitochondrial chromosome). Thus it can be benefical to exclude high coverage regions by filtering BAM files prior to analysis with *SGSeq*. For prediction with `predictTxFeatures()`, genomic regions with high splice complexity likely due to spurious alignments can be skipped automatically to speed up processing. Skipping is controlled with argument `max_complexity`. Identification of splice events is performed on a per-gene basis and can be parallelized using argument `cores` for `analyzeVariants()` and `findSGVariants()`.

## 9 Session information

- R version 3.2.1 beta (2015-06-08 r68489), `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 1.30.1, Biobase 2.28.0, BiocGenerics 0.14.0, GenomeInfoDb 1.4.0, GenomicFeatures 1.20.1, GenomicRanges 1.20.5, IRanges 2.2.4, S4Vectors 0.6.0, SGSeq 1.2.2, TxDb.Hsapiens.UCSC.hg19.knownGene 3.1.2, XVector 0.8.0, knitr 1.10.5
- Loaded via a namespace (and not attached): BiocParallel 1.2.3, BiocStyle 1.6.0, Biostrings 2.36.1, DBI 0.3.1, GenomicAlignments 1.4.1, RCurl 1.95-4.6, RSQLite 1.0.0, Rsamtools 1.20.4, XML 3.98-1.2, biomaRt 2.24.0, bitops 1.0-6, evaluate 0.7, formatR 1.2, futile.logger 1.4.1, futile.options 1.0.0, highr 0.5, igraph 0.7.1, lambda.r 1.1.7, magrittr 1.5, rtracklayer 1.28.4, stringi 0.4-1, stringr 1.0.0, tools 3.2.1, zlibbioc 1.14.0

## 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.