Counting with `summarizeOverlaps`

Valerie Obenchain[1]
Fred Hutchinson Cancer Research Institute, Seattle, WA

December 13-14, 2012

[1] vobencha@fhcrc.org

Topics

- Counting Modes
- BamFileList-method
- Results
- Counting Paired-End and Singleton Reads
- Splice sites

Counting Modes

Counting modes are patterned after those in HTSeq [2] and avoid double counting. The modes can be thought of as ways to resolve multi-hit reads.

- Read hits 0 features $\rightarrow$ discard
- Read hits 1 feature $\rightarrow$ count
- Read hits $>$ 1 feature $\rightarrow$ use a 'mode' to resolve the count

[2] http://www-huber.embl.de/users/anders/HTSeq/doc/overview.html

Counting Modes

Union

- Count if read hits 1 feature, else drop

IntersectionStrict

- Count if read falls completely 'within' one of the features, else drop

IntersectionNotEmpty

- Count if read falls in a unique disjoint region of one of the features, else drop

Set up ...

```
> library(Rsamtools)  ## BamFileList-method
> library(pasillaBamSubset) ## untreated1_chr4, untreated4_chr4
> library(TxDb.Dmelanogaster.UCSC.dm3.ensGene) ## annotation
```

BamFileList-method

The BamFileList-method uses `mclapply` under the hood. `yieldSize` enables streaming over the file.

```
> bamdir <- system.file(package="EMBO2012", "bigdata",
+                "bam", mustWork=TRUE)
> fls <- BamFileList(dir(bamdir, ".bam$", full=TRUE),
+           yieldSize=2000000)
> names(fls) <- basename(names(fls))
> countBam(fls)$records
```

```
[1] 2381906 1532899
```

Adjust annotation seqlevels to match the bam files and count:

```
> txdb <- TxDb.Dmelanogaster.UCSC.dm3.ensGene
> exbygene <- exonsBy(txdb, "gene")
> seqlevels(exbygene) <- sub("chr", "", seqlevels(exbygene))
> ine_counts <- summarizeOverlaps(exbygene, fls,
+                "IntersectionNotEmpty", ignore.strand=TRUE)
```

Results

Results are parsed into a `SummarizedExperiment`.

- Counts are accessed with `assays()`:

```
> head(assays(ine_counts)$counts, 3)
```

```
           SRR074431_subset.bam SRR074461_subset.bam
FBgn0000003                   19                   34
FBgn0000008                    1                    5
FBgn0000014                    0                    0
```

```
> colSums(assays(ine_counts)$counts)
```

```
SRR074431_subset.bam SRR074461_subset.bam
              591580               662661
```

- The `rowData` holds the annotation used for counting.

```
> summary(rowData(ine_counts))
```

```
   Length       Class        Mode
    14869 GRangesList          S4
```

Count Modes: user defined

`mode` can be any user defined function that has the same signature as the existing `modes` and returns a vector of counts the same length as `features`. This example wraps `countOverlaps`.

```
> myco <- function(reads, features, ignore.strand=FALSE, ...) {
+     countOverlaps(features, reads,
+                   ignore.strand=ignore.strand)
+ }
> co_counts <- summarizeOverlaps(exbygene, fls, mode=myco,
+                                ignore.strand=TRUE)
> head(assays(co_counts)$counts, 3)

           SRR074431_subset.bam SRR074461_subset.bam
FBgn0000003                   19                   34
FBgn0000008                    2                    5
FBgn0000014                    0                    0
```

Counting Paired-End and Singleton Reads

paired-end

- When counting paired-end reads set `SingleEnd` to FALSE.

```
> exbygene <- exonsBy(txdb, "gene")
> paired <- summarizeOverlaps(exbygene,
+                 BamFileList(untreated3_chr4()),
+                 SingleEnd=FALSE, ignore.strand=TRUE)
```

singleton

- Count singletons by setting `SingleEnd` to TRUE. Use the `ScanBamParam` to request paired reads with unmapped mates.

```
> singleton <- summarizeOverlaps(exbygene,
+                   BamFileList(untreated3_chr4()),
+                   param=ScanBamParam(flag=scanBamFlag(
+                           isPaired=TRUE,
+                           hasUnmappedMate=TRUE)),
+                   SingleEnd=TRUE, ignore.strand=TRUE)
```

Counting Paired-End and Singleton Reads

Summarize percent singleton reads

```
> ## count summary
> ct <- data.frame(paired=assays(paired)$counts[,1],
+                  singleton=assays(singleton)$counts[,1],
+                  row.names=rownames(singleton))
> ct$ratio <- round(ct$singleton/rowSums(ct), 5)
> head(ct[ct$singleton > 0,])
```

	paired	singleton	ratio
FBgn0002521	409	26	0.05977
FBgn0004607	12	2	0.14286
FBgn0004624	816	69	0.07797
FBgn0004859	199	10	0.04785
FBgn0005558	22	4	0.15385
FBgn0005561	11	1	0.08333

Splice Sites: `locateVariants`

Find gene-centric locations for a set of ranges.

```
> library(VariantAnnotation)
> gr <- as(readGappedAlignments(untreated1_chr4()),
+          "GRanges")
> loc <- locateVariants(gr, txdb, SpliceSiteVariants())
```

```
> loc[1]
GRanges with 1 range and 7 metadata columns:
      seqnames          ranges strand |   LOCATION   QUERYID
         <Rle>       <IRanges>  <Rle> |   <factor> <integer>
  [1]     chr4 [27087, 48388]      - | spliceSite      6381
          TXID     CDSID        GENEID   PRECEDEID    FOLLOWID
     <integer> <integer> <character> <character> <character>
  [1]     18906      <NA> FBgn0052011        <NA>        <NA>
```

Splice Sites: Overlap Encodings (Hervé Pagès)

Overlap encodings describe how the ranges in 'query' are qualitatively positioned with respect to the 'subject'. This information can detect complicated overlaps. In this example we look at reads that meet two general criteria,

- compatible with transcript splicing
- compatible with exon skips

Splice Sites: Overlap Encodings

Compatible with Transcript Splicing:

The read overlaps the transcript in a way that is compatible with the splicing of the transcript.

```
read (no gap):              oooooooo
transcript:       ...  >>>>>>>>>>>>>>  ...


read (1 gap):               ooooo---ooo
transcript:       ...  >>>>>>>>>   >>>>>>>>>  ...


read (2 gaps):                oo---ooooo---o
transcript:       ...  >>>>>>>>   >>>>>   >>>>>>>  ...
```

Splice Sites: Overlap Encodings

```
> flag0 <- scanBamFlag(isDuplicate=FALSE,
+             isNotPassingQualityControls=FALSE)
> gal <- readGappedAlignments(untreated1_chr4(),
+           use.names=TRUE, param=ScanBamParam(flag=flag0))
```

The high-level function `countCompatibleOverlaps` provides the number of compatible transcripts per alignment in 'gal'

```
> exbytx <- exonsBy(txdb, by="tx", use.names=TRUE)
> ncomptx <- countCompatibleOverlaps(gal, exbytx)
> table(ncomptx)
```

```
ncomptx
    0     1     2     3     4     5     6     7     8     9    10
53514 43731 16616 50092 10949  5404 13088  2502  6688  1723    48
```

Splice Sites: Overlap Encodings

Exon skips

The read overlaps the transcript in a way that would be "compatible" if 1 or more exons were removed from the transcript.

```
read (1 gap):          ooooo----------ooo
transcript:      ... >>>>>>>   >>>>   >>>>>>>> ...
```

```
read (1 gap):          ooooo-----------------ooo
transcript:      ... >>>>>>>   >>>>   >>>>>   >>>>>>>> ...
```

```
read (2 gaps):            oo---oooo-----------oo
transcript:      ... >>>>>>>   >>>>   >>>>>   >>>>>>>> ...
```

Splice Sites: Overlap Encodings

`isCompatibleWithSkippedExons` is a low-level function that operates directly on the overlap encodings.

```
> fo <- findOverlaps(gal, exbytx, ignore.strand=TRUE)
> enc <- encodeOverlaps(grglist(gal, order.as.in.query=TRUE),
+           exbytx, hits=fo, flip.query.if.wrong.strand=TRUE)
> compWithSkipped <- isCompatibleWithSkippedExons(enc)
> table(compWithSkipped)

compWithSkipped
 FALSE   TRUE
495625    860
```