Counting Alignment Overlaps with `countGenomicOverlaps`

Valerie Obenchain Martin Morgan

Edited: 28 April 2011; Compiled: August 18, 2011

1 Introduction

This vignette illustrates how to count reads aligned to a reference genome with `countGenomicOverlaps`. This function builds on the properties of `findOverlaps` to provide a method that is strand-aware and offers options to resolve multi-hit reads.

2 Data

For our test cases, we create a *GRangesList* of regions of interest (`subj`) and reads (`query`), e.g., using short helper functions `rng1` and `rng2`.

```
> library(GenomicFeatures)
> rng1 <- function(s, w)
+ GRanges(seq="chr1", IRanges(s, width=w), strand="+")
> rng2 <- function(s, w)
+ GRanges(seq="chr2", IRanges(s, width=w), strand="+")
> subj <- GRangesList(G1=rng1(1000, 500),
+                     G2=rng2(2000, 900),
+                     G3=rng1(c(3000, 3600), c(500, 300)),
+                     G4=rng2(c(7000, 7500), c(600, 300)),
+                     G5=rng2(c(9000, 9000), c(300, 600)),
+                     G6=rng1(4000, 500),
+                     G7=rng1(c(4300, 4500), c(400, 400)),
+                     G8=rng2(3000, 500),
+                     G9=rng1(c(5000, 5600), c(500, 300)),
+                     G10=rng1(6000, 500),
+                     G11=rng1(6600, 400))
> query <- GRangesList(read1=rng1(1400, 500),
+                      read2=rng2(2700, 100),
+                      read3=rng1(3400, 300),
+                      read4=rng2(7100, 600),
```

```
+                       read5=rng2(9000, 200),
+                       read6=rng1(4200, 500),
+                       read7=rng2(c(3100, 3300), 50),
+                       read8=rng1(c(5400, 5600), 50),
+                       read9=rng1(c(6400, 6600), 50))
```

One might also create `subj` with `makeTranscriptDbFromUCSC`, `import` (for GFF-like) or other *rtracklayer* functionality, or from queries to *biomaRt* or using the *AnnotationDbi*, **org*, and *BSgenome** Bioconductor annotation packages. `query` might normally come from BAM files (e.g., `readGappedAlignements` in *GenomicRanges* or `scanBam` in *Rsamtools*) or other aligned reads (using base R functionality for pure text files, or perhaps `readAligned` from *ShortRead*, if the alignments do not contain gaps). These operations are described in the vignettes of the corresponding packages.

The `subj` and `query` objects contain the genes and reads shown in Figure 1. The external boxes labeled with "G" represent the genes. The internal boxes labeled with "F" represent a feature which may be either exons or transcripts. The unlabeled boxes are the reads. The `subj` contains 11 genes where G1, G2, G6, G8, G10 and G11 have a single feature. Genes G3, G4, G5, G7 and G9 have multiple features, some of which are overlapping.

Our `query` contains a total of nine reads, where reads 7, 8 and 9 are split reads. BAM files may be read into a `GappedAlignments` object with `readGappedAlignments`. When a `GappedAlignments` object is coerced to a `GRangesList`, reads with a gap in their CIGAR string are stored as multiple ranges in the `GRangesList`. Hence, reads 7, 8 and 9 appear as multiple ranges within the list element.

We will use these gene-feature combinations to demonstrate how `countGenomicOverlaps` counts 'hits' in various circumstances.

3 `countGenomicOverlaps` Decision Logic

`countGenomicOverlaps` first identifies overlaps of a specific `type` and reads that hit a single subject are recorded. Reads that hit multiple subjects are resolved with one of `resolve` methods.

The logic behind `countGenomicOverlaps` can be understood through the `type` and `resolution` arguments. The `type` argument is passed to `findOverlaps` and has the same options described in the `findOverlaps` documentation (i.e., "any", "start", "end", "within", "equal"). The `resolution` argument indicates how multi-hit reads should be resolved; the options are "none", "divide", and "uniqueDisjoint". Option "none" ignores multi-hit reads. "divide" simply divides the hit equally among all features hit. For example, three 3 overlapping features will all get 1/3 of a hit. The "uniqueDisjoint" option first identifies the disjoint regions of all features hit. Regions that are common to multiple features (i.e., overlapping) are removed and only unique (i.e., non-overlapping) disjoint regions are left. If the read overlaps any of the remaining regions, the hit is assigned to the feature that the disjoint region originated from.

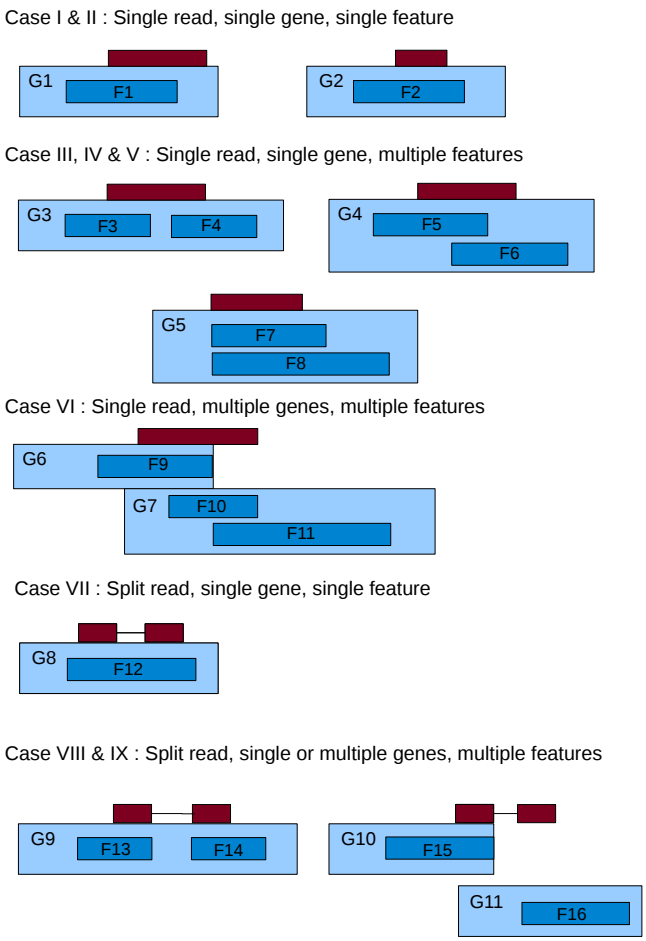

Figure 1: Overlap Cases

The primary difference in the handling of split reads vs simple reads (i.e., no gap in the CIGAR) is the portion of the read hit each split read fragment has to contribute. All reads, whether simple or split, have an overall value of 1 to contribute to a subject they hit. In the case of the split reads, this value is further divided by the number of fragments in the read. For example, if a split read has 3 fragments (i.e., two gaps in the CIGAR) each fragment has a value of 1/3 to contribute to the subject they hit. As with the simple reads, depending upon the `resolution` chosen the value may be divided, fully assigned

or discarded.

The decision process for `countGenomicOverlaps` is shown graphically in Figure 2.

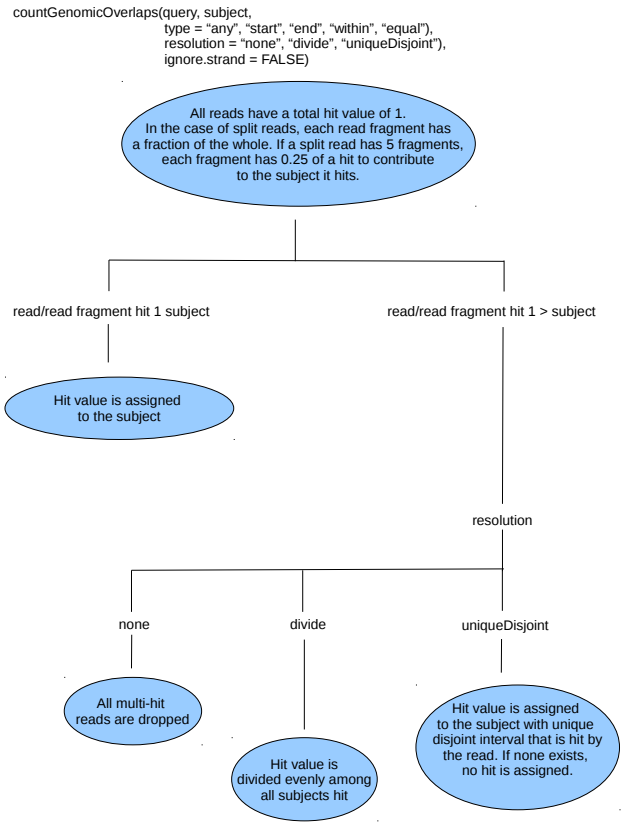

Figure 2: `countGenomicOverlaps` Decision Logic

4 Examples

Below we investigate the counting of reads in the situations depicted in Figure 1 using `type` “any” and “within”.

4.1 `type` = "any"

When the `resolution` is "none", both the multi-hit reads are In this case we have simple reads hitting F1 and F2 and split reads hitting F12 through F16. F12 is hit by two fragments of a split read, each with 0.5 of a hit to contribute which results in a total of 1.0.

When `resolution` "divide" is used, we see all features in all genes are hit. This is a quality of the "divide" option; if a read hits a feature that feature receives either a whole hit or some shared piece of a hit. "divide" does not choose between features or drop reads it cannot resolve as is done with "uniqueDisjoint". Genes G3 and G4 each have two features hit by a single read, thus F3, F4 and F5, F6 and F7, F8 are assigned 0.5 hit each. F9, F10 and F11 in genes G6 and G7 are hit by the same read and are assigned 0.33 hit each. Features F13, F14, F15 and F16 are hit by split reads each having two fragments. F12 is hit by both fragments and thus gets the full hit value of 1. F13, F14, F15 and F16 are each hit by a single fragment and get a hit value of 0.5.

The decision making of `resolution` "uniqueDisjoint" can be explored by comparing G6 to G3, G4 and G5. "uniqueDisjoint" is able to resolve multi-hits in cases such as G6 but not in G3, G4 or G5. For "uniqueDisjoint" to resolve a multi-hit read, there must exist a unique disjoint region (i.e., a region not shared with the other features) that is hit by the read for only one of the features. In G6 but there exists such a region for F9 and thus the hit is assigned to it. This is not the case in G3, G4 or G5. In G3 and G4 unique disjoint regions exist for all (both) of the features in the gene so "uniqueDisjoint" is unable to make a decision and the read is discarded. Alternatively, in G5 no unique disjoint regions exist for either feature so the read is discarded. There are certainly other examples of feature overlap not depicted in the graphic in Figure 2. However the same concepts apply in that "uniqueDisjoint" is able to resolve multi-hit reads if there exists a unique disjoint region of one of the features hit by the read. Multiple such regions or none at all are cases where "uniqueDisjoint" is unable to resolve the hit.

```
> none <- countGenomicOverlaps(query, subj, type="any",
+                                    resolution="none")
> divide <- countGenomicOverlaps(query, subj, type="any",
+                                   resolution="divide")
> uniqueDisjoint <- countGenomicOverlaps(query, subj, type="any",
+                                          resolution="uniqueDisjoint")
> res_any <- data.frame(
+                none = values(unlist(none))[["hits"]],
+                divide = round(values(unlist(divide))[["hits"]], 2),
+                uniqueDisjoint = values(unlist(uniqueDisjoint))[["hits"]])
> rownames(res_any) <- paste("F", seq_len(16), sep="")
> res_any

    none divide uniqueDisjoint
F1   1.0   1.00            1.0
```

```
F2   1.0   1.00           1.0
F3   0.0   0.50           0.0
F4   0.0   0.50           0.0
F5   0.0   0.50           0.0
F6   0.0   0.50           0.0
F7   0.0   0.50           0.0
F8   0.0   0.50           0.0
F9   0.0   0.33           1.0
F10  0.0   0.33           0.0
F11  0.0   0.33           0.0
F12  1.0   1.00           1.0
F13  0.5   0.50           0.5
F14  0.5   0.50           0.5
F15  0.5   0.50           0.5
F16  0.5   0.50           0.5
```

4.2 type = "within"

When `type` = "within" is chosen, the `resolution` option "uniqueDisjoint" is not available. "uniqueDisjoint" resolves multi-hits by creating disjoint regions of all overlapping features and removing those regions that are common (i.e., where features overlap). If a read was identified as a multi-hit read with `type` = "within" then by definition it falls in the common region that `resolution` "uniqueDisjoint" discards. Therefore it does not make sense to resolve `type` "within" with `resolution` "uniqueDisjoint".

```
> none <- countGenomicOverlaps(query, subj, type="within", resolution="none")
> divide <- countGenomicOverlaps(query, subj, type="within", resolution="divide")
> res_within <- data.frame(none = values(unlist(none))[["hits"]],
+                          divide = values(unlist(divide))[["hits"]])
> rownames(res_within) <- paste("F", seq_len(16), sep="")
> res_within
```

```
    none divide
F1   0.0    0.0
F2   1.0    1.0
F3   0.0    0.0
F4   0.0    0.0
F5   0.0    0.0
F6   0.0    0.0
F7   0.0    0.5
F8   0.0    0.5
F9   0.0    0.0
F10  0.0    0.0
F11  0.0    0.0
F12  1.0    1.0
```

```
F13  0.5    0.5
F14  0.5    0.5
F15  0.5    0.5
F16  0.5    0.5
```

Comparing our results with those of the `type` "any" we see that F1 is now not hit. When `resolution` is "divide" we hit F7 and F8 in gene G5 but not the features in genes G3 and G4. Results for the split reads are the same as for `type` "any" because all read fragments fall within the feature they hit.