Ranges, sequences and alignments

Michael Lawrence

July 25, 2014

Outline

Software for genomic ranges

Isoform-specific expression

Counting RNA-seq junctions

Genomic data visualization

Variant calling

Summary

Outline

Software for genomic ranges

Isoform-specific expression

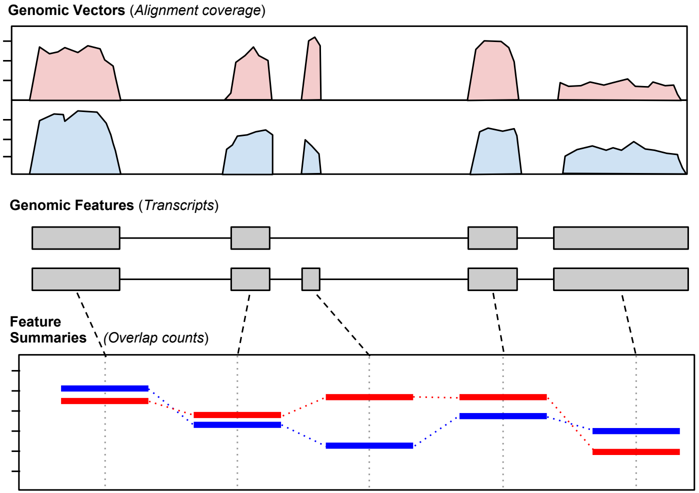
Counting RNA-seq junctions

Genomic data visualization

Variant calling

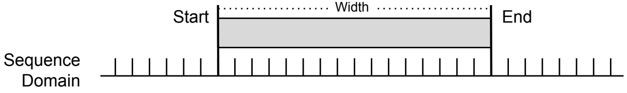
Summary

Genomic data falls into three types


Genomic Vectors (*Alignment coverage*)
Genomic Features (*Transcripts*)
Feature Summaries (*Overlap counts*)

The range: grand unifier of genomic data

- We define the genomic range by:
  - Sequence domain (e.g., chromosome, contig)
  - Start and end
  - Strand
  - Annotations (e.g., score, or name)

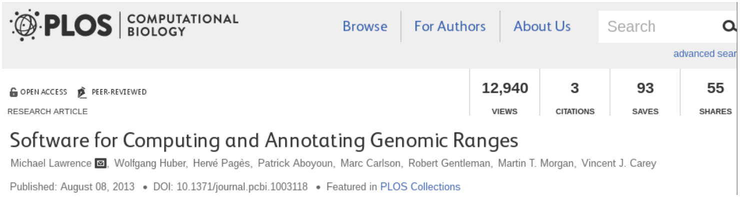


- The genomic range
  - Represents genomic features, like genes and alignments
  - Indexes into genomic vectors, like sequence and coverage
  - Links summaries, like RPKMs, to genomic locations
- The genome acts as a scaffold for data integration
- Ranges have a specialized structure and algebra, requiring specialized data types and algorithms

The IRanges and GenomicRanges packages

Collaborative effort with Bioconductor

- Define core classes for representing ranges, like:
  - *GRanges* for simple ranges (exons)
  - *GRangesList* for compound ranges (multi-exon transcripts)
- Algorithms for transforming, comparing, summarizing ranges.
- Run-length encoding of genome-length vectors: *Rle*
- Encapsulation of feature-level experimental summaries and metadata: *SummarizedExperiment*.



OPEN ACCESS PEER-REVIEWED

RESEARCH ARTICLE

12,940	3	93	55
VIEWS	CITATIONS	SAVES	SHARES

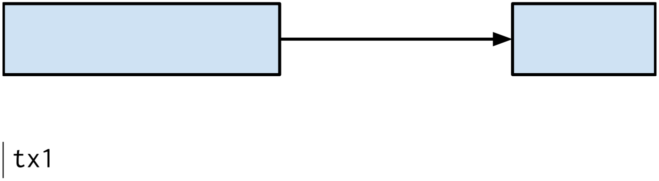
Software for Computing and Annotating Genomic Ranges

Michael Lawrence, Wolfgang Huber, Hervé Pagès, Patrick Aboyoun, Marc Carlson, Robert Gentleman, Martin T. Morgan, Vincent J. Carey

Published: August 08, 2013 • DOI: 10.1371/journal.pcbi.1003118 • Featured in PLOS Collections

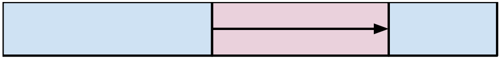
Representing a transcript with *GRanges*

We can represent any type of genomic range with *GRanges*, including the exons of a transcript



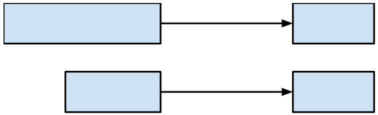
Finding the unspliced transcript using range()

```
unspliced <- range(tx1)
```



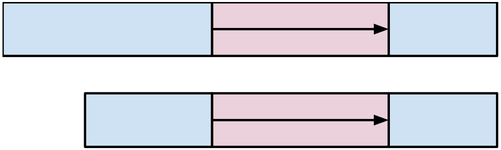
Combining multiple transcripts in a *GRangesList*

```
txList <- GRangesList(tx1, tx2)
```



Finding both unspliced transcripts using range()

```
unspliced <- range(txList)
```



range() returns the appropriate result given the type of the input.

Classes are important for complex data

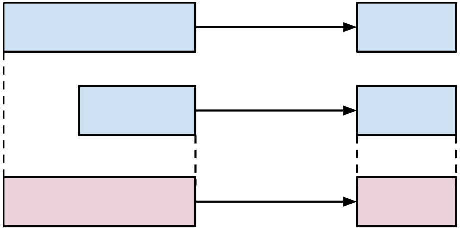
- Ensure the integrity/validity of data (strong typing)
- Hide implementation and enable code to express algorithms in an abstract way (polymorphism)
- Support analysis by better representing the semantics of the biological entity compared to an ordinary *data.frame*
- Science defies rigidity: we need hybrid objects that combine strongly typed fields with arbitrary user-level metadata

Ranges algebra

Arithmetic	`shift`, `resize`, `restrict`, `flank`
Set operations	`intersect`, `union`, `setdiff`, `gaps`
Summaries	`coverage`, `reduce`, `disjoin`
Comparison	`findOverlaps`, `findMatches`, `nearest`, `order`

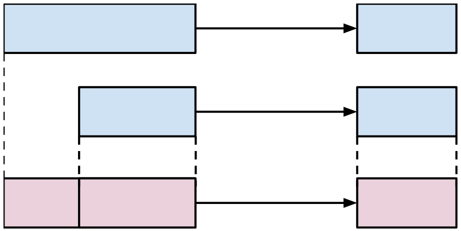
Finding "gene" regions using reduce()

```
exon.bins <- reduce(unlist(txList))
```



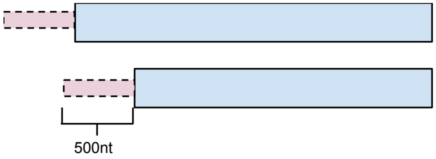
Generating DEXseq counting bins using `disjoin()`

```r
exon.bins <- disjoin(unlist(txList))
```



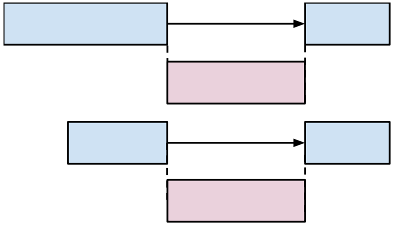
Finding promoters using `flank()`

```r
promoters <- flank(unspliced, 500)
```



Finding the introns using psetdiff()

```r
introns <- psetdiff(unspliced, txList)
```



Outline

Software for genomic ranges

Isoform-specific expression

Counting RNA-seq junctions

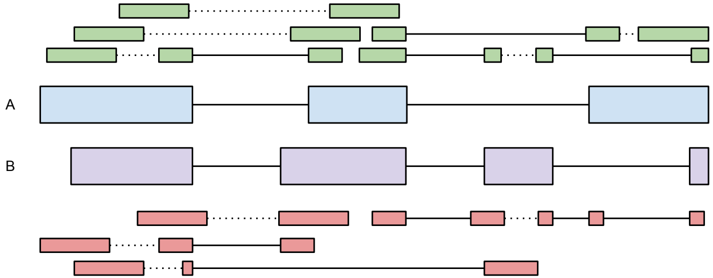
Genomic data visualization

Variant calling

Summary

Counting compatible alignments

- The `findSpliceOverlaps()` function in GenomicAlignments finds *compatible* overlaps between transcripts and RNA-seq read alignments.
- To be *compatible* a read must align completely within the exons and the read gaps should exactly match the introns over the read extent

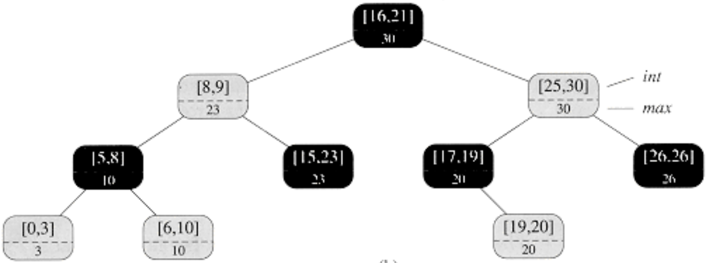


The `findSpliceOverlaps()` algorithm

1. Match read alignments to transcripts by any overlap.
2. For each match, check that the alignment segments and exons are identical over the range of the alignment.

Overlap detection algorithm

- Fast overlap detection based on a textbook interval tree algorithm.
- Extended algorithm for common case of sorted queries (does not need to restart search for each query).
- Index is represented as an *IntervalTree*, which acts like any other *Ranges* object (abstraction).

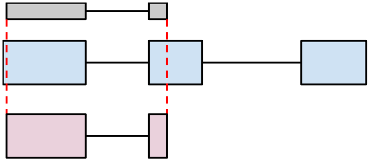


(b)

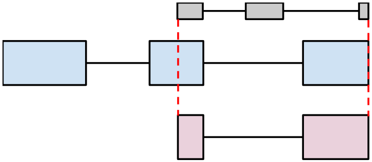
Restrict the problem to range of alignment

```r
subtx <- restrict(tx, start(alignments),
                      end(alignments))
```

Hit A



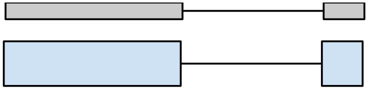
Hit B



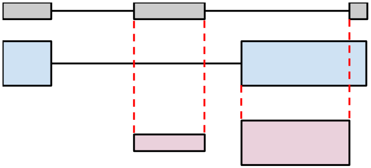
Check that alignments and sub-transcripts are equal

```
sum(width(psetdiff(alignments, subtx))) == 0L &
sum(width(psetdiff(subtx, alignments))) == 0L
```

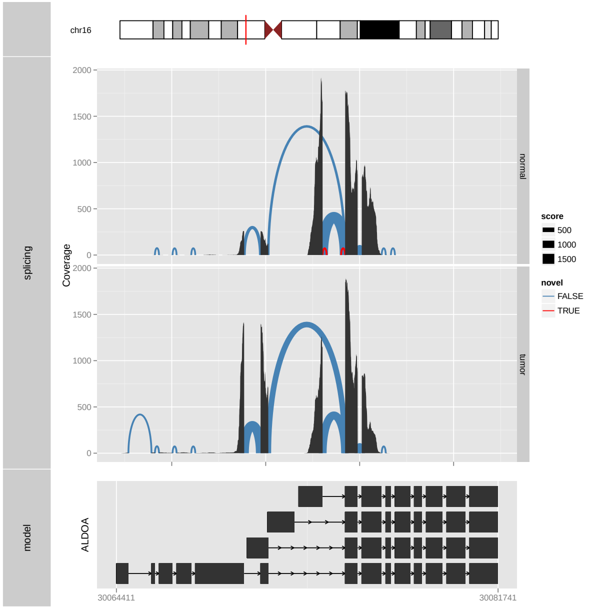
Hit A: Compatible



Hit B: Incompatible



Summary plot with ggbio



Outline

Software for genomic ranges

Isoform-specific expression

Counting RNA-seq junctions

Genomic data visualization

Variant calling

Summary

Example junction counting workflow

Steps

1. Load alignments from BAM
2. Tabulate junctions in alignments
3. Retrieve splice site sequences from reference assembly
4. Store intron locations, counts and annotations in a single object that represents our summarized dataset
5. Obtain splice site sequences and annotate known splices

Assumption

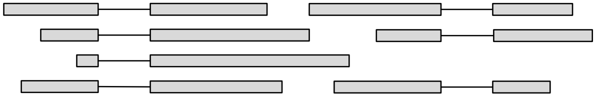
The sequences were generated by a strand-specific protocol.

Existing tools

When doing this for real, see `junctions()` in GenomicAlignments, which is much fancier and can infer the strand based on canonical splice site motifs.

Loading alignments from a BAM file

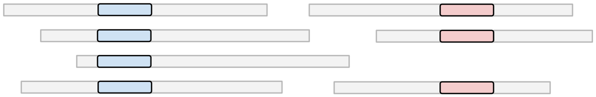
```r
ga <- readGAlignments("my.bam")
reads <- grglist(ga)
```



Tabulating junctions

Find the unique junctions

```
read.junctions <- psetdiff(range(reads), reads)
unique.junctions <- unique(read.junctions)
```



Count matches to unique junctions

```
counts <- countMatches(unique.junctions, read.junctions)
```

Storing summarized counts: *SummarizedExperiment*

The *SummarizedExperiment* object enables integration of feature by sample measurements with feature and sample annotations.

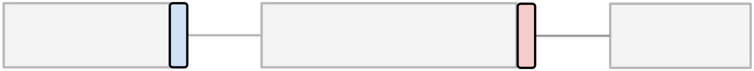
```
assays <- list(junction_count=cbind(A=count))
se <- SummarizedExperiment(assays, unique.junctions)
se
```

```
class: SummarizedExperiment
dim: 20024 1
exptData(0):
assays(1): 'junction_count'
rownames: NULL
colnames(1): A
colData names(0):
```

Retrieving splice site sequences

Finding the 5' splice sites

```
splice.sites <- resize(rowData(se), 2)
```



Getting and recording the sequences

```
library(BSgenome.Hsapiens.UCSC.hg19)
rowData(se)$splice.seqs <- getSeq(Hsapiens, splice.sites)
```

Example of storing arbitrary annotations on the rows/features, a feature supported by most GenomicRanges containers.

Annotate for known splices

- Reference transcript annotations are stored as *TranscriptDb* objects and distributed in individual packages.
- We can load the transcript structures as ranges and compare their introns to those derived from the reads.

Deriving the known junctions

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
tx <- exonsBy(TxDb.Hsapiens.UCSC.hg19.knownGene)
known.junctions <- psetdiff(range(tx), tx)
```

Annotating junctions for matches to reference set

```
rowData(se)$known <- se %in% known.junctions
```

Outline

Software for genomic ranges

Isoform-specific expression

Counting RNA-seq junctions

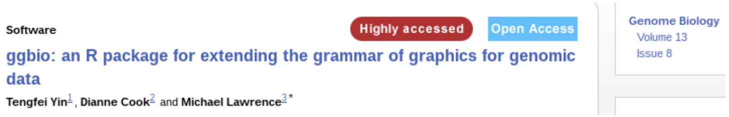
Genomic data visualization

Variant calling

Summary

The ggbio package

Written by intern Tengfei Yin



Software

Highly accessed Open Access

ggbio: an R package for extending the grammar of graphics for genomic data

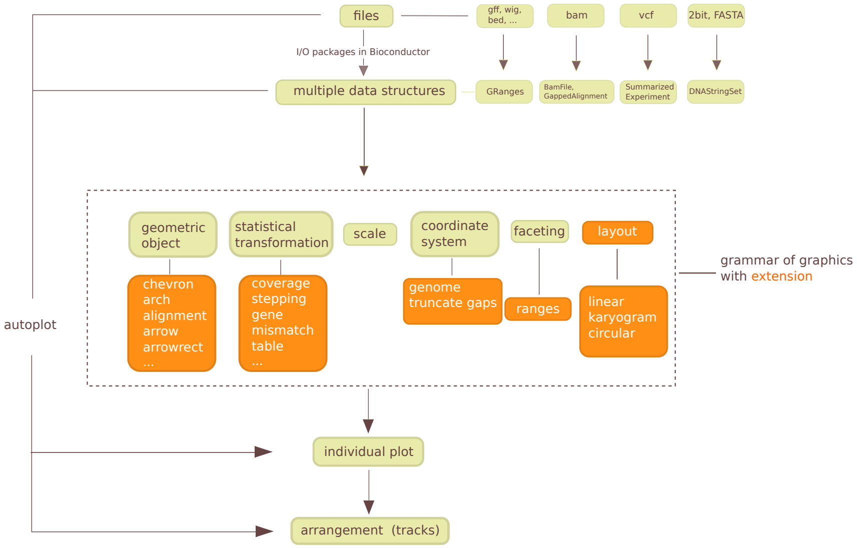
Tengfei Yin[1], Dianne Cook[2] and Michael Lawrence[3]*

Genome Biology Volume 13 Issue 8

- An R/Bioconductor package that extends the Wilkinson/Wickham grammar for applications in genomics
- Integrated with *IRanges* and friends
  - Operates on *GenomicRanges* data structures
  - Leverages efficient range-based algorithms from *IRanges*
  - Relies on file input routines for direct plotting, like those from *rtracklayer* and *Rsamtools*
- Programming interface has two levels of abstraction:

`autoplot`	Maps Bioconductor data structures to plots
grammar	Mix and match to create custom plots

Architecture of ggbio



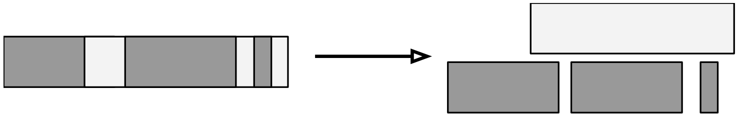
Automatic plotting of Bioc data structures

```
ir
```

```
autoplot(ir) + theme_bw()
```

Computing Y layout with IRanges

```
y <- disjointBins(ir)
```



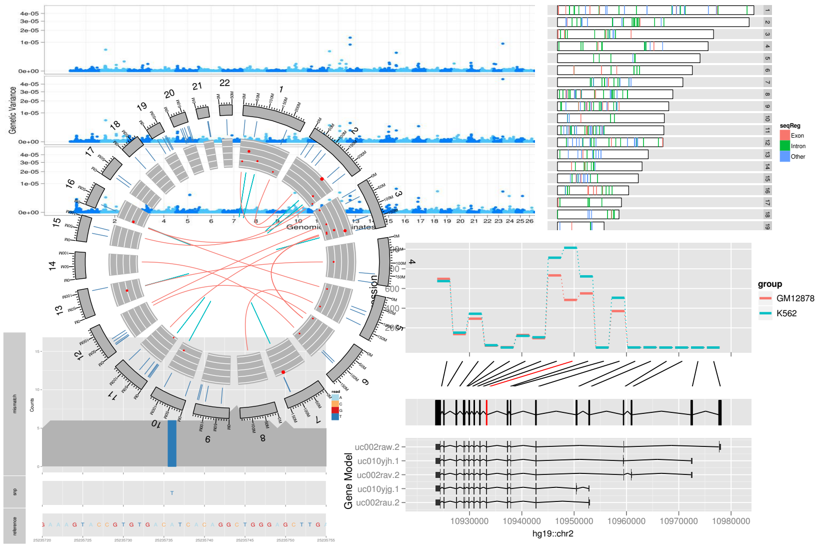
Deep integration with Bioconductor

```
class(bam)
```

```
class(p53)
```

```
tracks(bam, p53) + theme_bw()
```

Pretty pictures



Outline

Software for genomic ranges

Isoform-specific expression

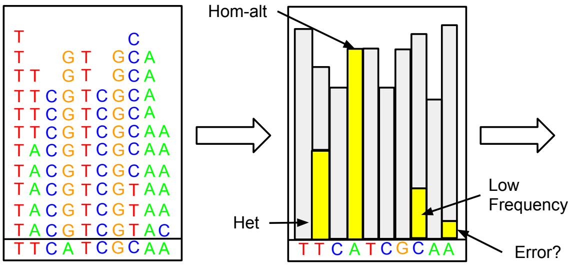
Counting RNA-seq junctions

Genomic data visualization

Variant calling

Summary

Variant calling



POS	REF	ALT
2	T	A
4	A	G
8	C	T

Variant calling use cases

DNA: variants

- Genetic associations with disease
- Mutations in cancer
- Characterizing heterogeneous cell populations

RNA: allele-specific expression

- Allelic imbalance, often differential
- Association with isoform usage (splicing QTLs)
- RNA editing (allele absent from genome)

VariantTools package

- Convenient interface for tallying mismatches and indels
- Provides several built-in variant filters
- Integrates:
  - *VRanges* data structure from `VariantAnnotation`
  - Tallying with `bam_tally` via `gmapR`
  - *FilterRules* framework from `IRanges`
- By default, `callVariants` executes a simple algorithm for finding general variants

VRanges

- The tally results are stored in a *VRanges* object
- One element/row per position + alt combination
- *GRanges* extension with fixed columns describing variants

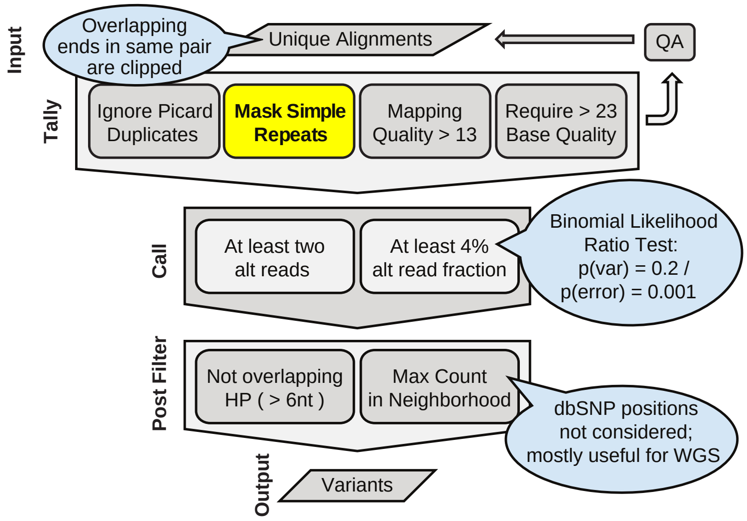
`ref`	ref allele
`alt`	alt allele
`totalDepth`	total read depth
`refDepth`	ref allele read depth
`altDepth`	alt allele read depth
`sampleNames`	sample identifiers
`softFilterMatrix`	*FilterMatrix* of filter results
`hardFilters`	*FilterRules* used to subset object

- Inherits implementation of range algebra and overlap detection
- Tracks filter provenance

Pipeline overview

./fig/fig2A.pdf

Masking simple repeats


Input
Overlapping ends in same pair are clipped
Unique Alignments
QA
Tally
Ignore Picard Duplicates
Mask Simple Repeats
Mapping Quality > 13
Require > 23 Base Quality
Call
At least two alt reads
At least 4% alt read fraction
Binomial Likelihood Ratio Test: p(var) = 0.2 / p(error) = 0.001
Post Filter
Not overlapping HP (> 6nt)
Max Count in Neighborhood
dbSNP positions not considered; mostly useful for WGS
Output
Variants

Masking simple repeats

Load the repeats

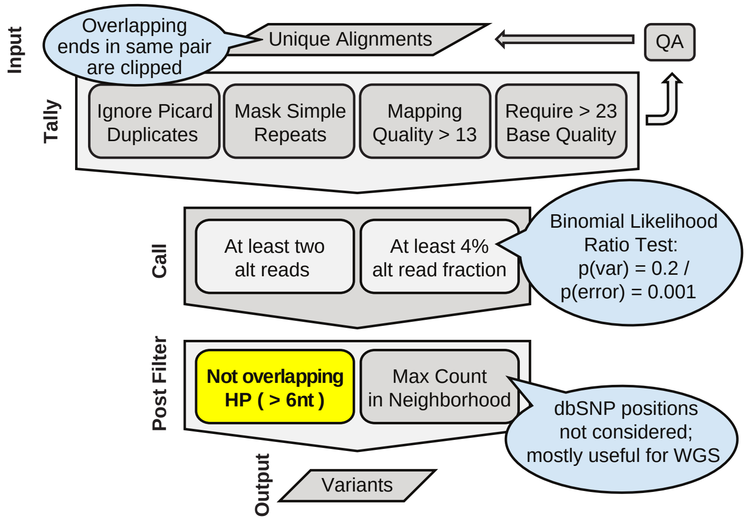
```
repeats <- rtracklayer::import("repeats.bed")
simple.classes <- c("Low_complexity", "Simple_repeat")
repeats <- subset(repeats, repClass %in% simple.classes)
```

```
GRanges with 15055 ranges and 1 metadata column:
     seqnames            ranges strand   |       repClass
        <Rle>         <IRanges>  <Rle>   |       <factor>
  [1]   chr20  [64533, 64556]       +   | Low_complexity
  [2]   chr20  [67648, 67680]       +   |  Simple_repeat
  [3]   chr20  [69506, 69535]       +   |  Simple_repeat
```

Excluding variants over repeats

```
v <- v[!overlapsAny(v, repeats, ignore.strand=TRUE)]
```

Excluding variants in homopolymers


Input
Overlapping ends in same pair are clipped
Unique Alignments
QA
Tally
Ignore Picard Duplicates
Mask Simple Repeats
Mapping Quality > 13
Require > 23 Base Quality
Call
At least two alt reads
At least 4% alt read fraction
Binomial Likelihood Ratio Test: p(var) = 0.2 / p(error) = 0.001
Post Filter
Not overlapping HP (> 6nt)
Max Count in Neighborhood
dbSNP positions not considered; mostly useful for WGS
Output
Variants

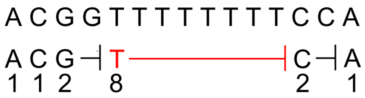
Excluding variants in homopolymers

Load the GMAP genome with gmapR

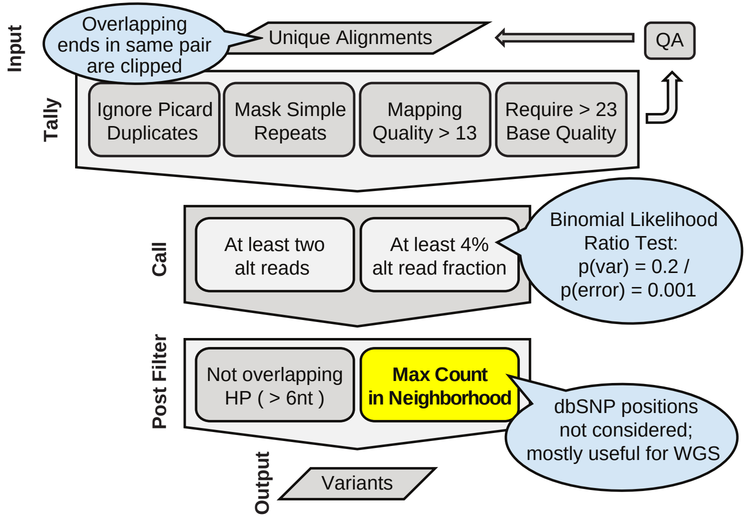
```
genome.sequence <- getSeq(genome)
```

Compute homopolymers (> 6nt)

```
chr1.rle <- Rle(charToRaw(genome.sequence[[1L]]))
chr1.hp <- subset(ranges(chr1.rle), width > 6L)
```



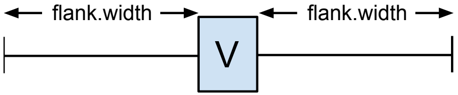
Computing variant neighborhoods


Input
Overlapping ends in same pair are clipped
Unique Alignments
QA
Tally
Ignore Picard Duplicates
Mask Simple Repeats
Mapping Quality > 13
Require > 23 Base Quality
Call
At least two alt reads
At least 4% alt read fraction
Binomial Likelihood Ratio Test: p(var) = 0.2 / p(error) = 0.001
Post Filter
Not overlapping HP (> 6nt)
Max Count in Neighborhood
dbSNP positions not considered; mostly useful for WGS
Output
Variants

Computing variant neighborhoods

Form neighborhoods from variants

```
neighborhoods <- v + flank.width
```

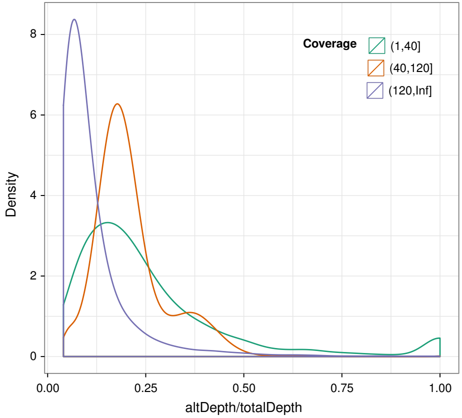


Assign variants to neighborhoods

```
hits <- findOverlaps(v, neighborhoods)
```

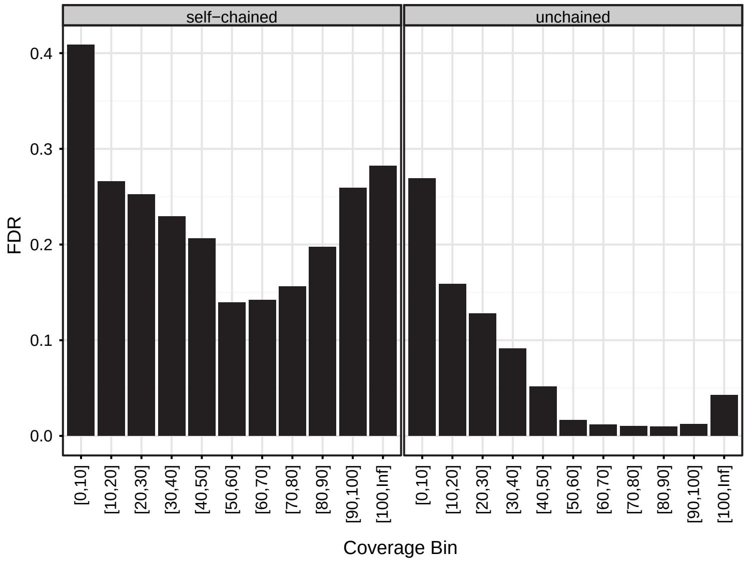
Extreme coverage predicts aberrant frequencies

- Coverage in the expected range (40-120) shows expected variant frequencies
- High coverage (>120) shows much lower frequencies than expected; mapping error?
- Low coverage (<40) also shows aberrant frequencies



FDR associated with coverage extremes

`findOverlaps(variants, self.chains)`



Summary

- Ranges are a fundamental, integrative data type requiring special data structures and algorithms.
- IRanges and friends provide R with an object-oriented framework for representing and computing ranges.
- These packages support over 100 Bioc and CRAN packages, including *HTSeqGenie*, our sequencing pipeline
- They are being applied beyond genomics, e.g., time series

Acknowledgements

Bioconductor

- Herve Pages
- Patrick Aboyoun
- Valerie Oberchain
- Martin Morgan
- Bioconductor community

ggbio

- Tengfei Yin
- Di Cook

isoseq

- Jinfeng Liu

Group

- Robert Gentleman
- Melanie Huntley
- Leonard Goldstein
- Yi Cao
- Jeremiah Degenhardt
- Gabe Becker

Outline

Software for genomic ranges

Isoform-specific expression

Counting RNA-seq junctions

Genomic data visualization

Variant calling

Summary

Summary

- The range integrates the different types of genomic data.
- IRanges and GenomicRanges define the fundamental abstractions, data types and utilities for representing, manipulating, comparing, and summarizing ranges.
- The data structures support storage of arbitrary metadata, and are well integrated with reference annotation sources and visualization packages.
- We applied these tools to the analysis of transcript expression and junction counting in the context of RNA-seq data.
- Broader applications include: variant calling, ChIP-seq, proteomics, and even general fields like time series analysis.

Your turn

- IRanges, GenomicRanges and friends are infrastructure and thus primarily designed for use by software developers.
- The hope is that as use cases emerge, third party developers (like you) create high-level, specialized packages that hide most of the complexity of the underlying framework.
- Examples: ChIPpeakAnno, easyRnaSeq, VariantFiltering, . . . more are welcome.

Acknowledgements

- Herve Pages
- Patrick Aboyoun
- Valerie Oberchain
- Martin Morgan
- Robert Gentleman