Analysis of VAR-Seq Data with R/Bioconductor

...

Neerja Katiyar

December 7, 2014

Overview
- Workflow
- Software Resources
- Data Formats

VAR-Seq Analysis
- Aligning Short Reads
- Variant Calling
- Annotating Variants
  - Prerequisites for Annotating Variants
  - Working VCF Objects
  - Adding Genomic Context to Variants

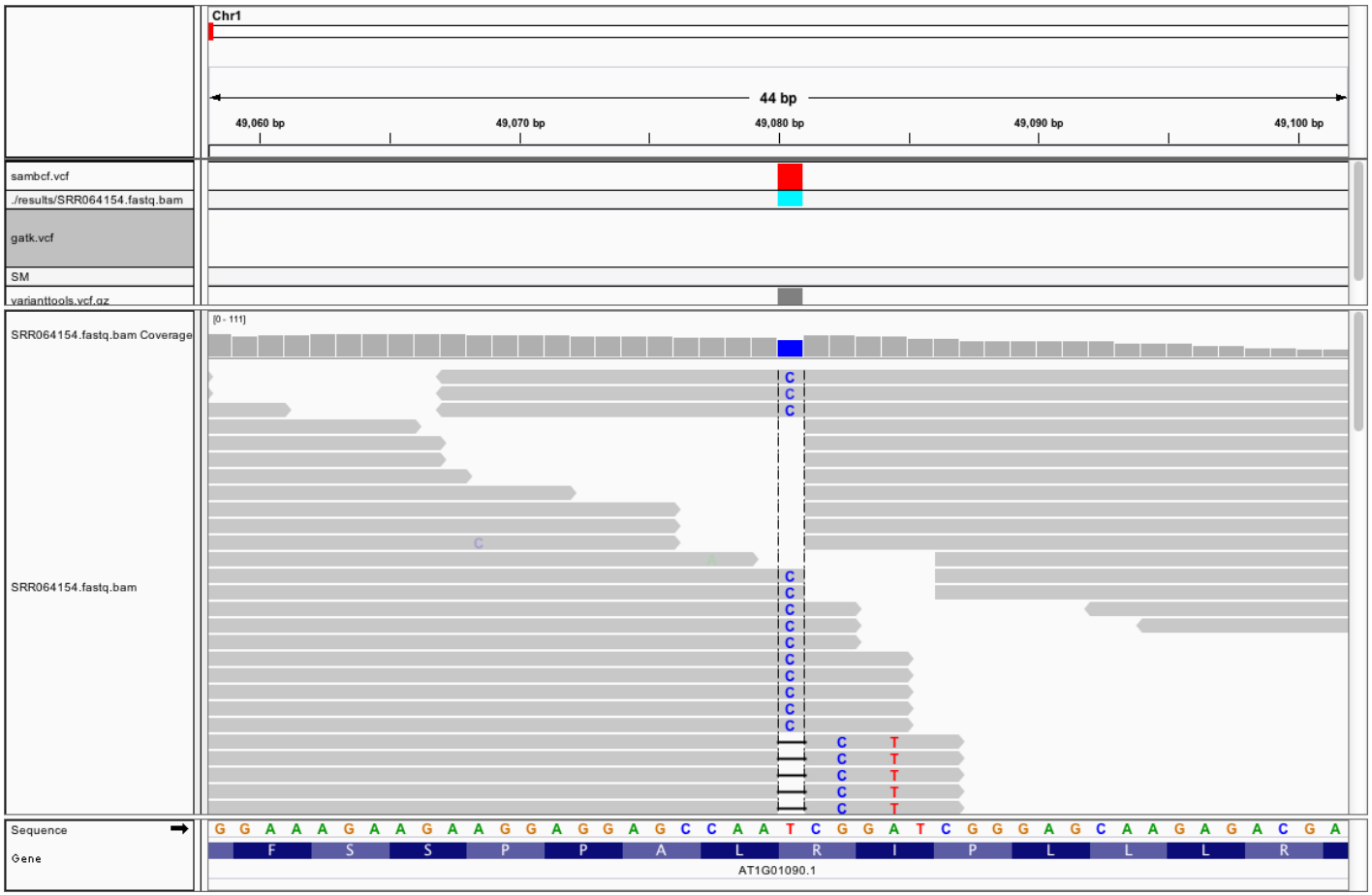
Outline

Overview
- Workflow
- Software Resources
- Data Formats

VAR-Seq Analysis
- Aligning Short Reads
- Variant Calling
- Annotating Variants
  - Prerequisites for Annotating Variants
  - Working VCF Objects
  - Adding Genomic Context to Variants

Objectives and Requirements

- Determine sequence differences (*e.g.* SNPs) of a sample in comparison to a reference genome
- Usually, sample and reference need to share high sequence similarity



Outline

Overview

- Workflow
- Software Resources
- Data Formats

VAR-Seq Analysis

- Aligning Short Reads
- Variant Calling
- Annotating Variants
  - Prerequisites for Annotating Variants
  - Working VCF Objects
  - Adding Genomic Context to Variants

VAR-Seq Analysis Workflow

- Read quality filtering
- Read mapping with variant tolerant aligner
- Postprocess alignments: mark/remove PCR duplicates, indel refinement, quality score recalibration, etc.
- SNP/Indel calling
- Quality filtering of candidate variants
- Annotate variants

Most Common Sources of Error

False positive variant calls

- PCR errors/duplicates inflate read support
- Variants from low coverage areas
- Sequencing errors
- False read placements

False negative variant calls

- Low/no coverage
- Complex rearrangements prevent read mapping

Outline

Overview
- Workflow
- Software Resources
- Data Formats

VAR-Seq Analysis
- Aligning Short Reads
- Variant Calling
- Annotating Variants
  - Prerequisites for Annotating Variants
  - Working VCF Objects
  - Adding Genomic Context to Variants

Tools for Variant Calling

Variant Tolerant Aligners

- Bowtie2 Link, SOAPsnp Link, MAQ Link, BWA Link, *gmapR* Link, ...

Alignment Processing

- SAMtools Link, *Rsamtools* Link, Picard Link, ...

Variant Calling

- SAMtools/BCFtools Link, *VariantTools* Link VarScan Link, GATK Link, ...

Variant Annotation

- *VariantAnnotation* Link, SnpEff Link, VariantAnnotator Link, ...

Variant Visualization

- IGV Link, *ggbio* Link, *Gviz* Link, ...

Additional Bioconductor Tools for Variant Analysis

deepSNV	Sub-clonal SNVs in deep sequencing experiments Link
cn.mops	Mixture of Poissons copy number variation estimates Link
exomeCopy	Hidden Markov copy number variation estimates Link
ensemblVEP	Interface to the Ensembl Variant Effect Predictor Link
snpStats	SnpMatrix and XSnpMatrix classes and methods Link
GWAStools	Tools for Genome Wide Association Studies Link
GGtools	eQTL identification Link

Outline

Overview
- Workflow
- Software Resources
- Data Formats

VAR-Seq Analysis
- Aligning Short Reads
- Variant Calling
- Annotating Variants
  - Prerequisites for Annotating Variants
  - Working VCF Objects
  - Adding Genomic Context to Variants

Variant Call Format (VCF)

- The Variant Call Format (VCF) is a standard for storing variant data. BCF is the binary version of VCF.
- VCF consists of 3 main components: (i) meta-information (ii) one header line and (iii) data component
- The data component is a tab-delimited table containing the following columns:

CROM	Chromosome name
POS	1-based position. For an indel, this is the position preceding the indel.
ID	Variant identifier. Usually the dbSNP rsID.
REF	Reference sequence at POS involved in the variant. For a SNP, it is a single base.
ALT	Comma delimited list of alternative sequence(s).
QUAL	Phred-scaled probability of all samples being homozygous reference.
FILTER	Semicolon delimited list of filters that the variant fails to pass.
INFO	Semicolon delimited list of variant information.
FORMAT	Colon delimited list of the format of individual genotypes in the following fields.
Sample(s)	Individual genotype information defined by FORMAT.

- For details see here: SAMtools Link and 1000 Genomes Link

Outline

Overview
- Workflow
- Software Resources
- Data Formats

VAR-Seq Analysis
- Aligning Short Reads
- Variant Calling
- Annotating Variants
  - Prerequisites for Annotating Variants
  - Working VCF Objects
  - Adding Genomic Context to Variants

Data Sets and Experimental Variables

To make the following sample code work, please follow these instructions:

- Download and unpack the sample data Link for this practical.
- Direct your R session into the resulting `Rvarseq` directory. It contains four slimmed down FASTQ files (SRA023501 Link) from *A. thaliana*, as well as the corresponding reference genome sequence (FASTA) and annotation (GFF) file.
- Start the analysis by opening in your R session the `Rvarseq.R` script Link which contains the code shown in this slide show in pure text format.

The FASTQ files are organized in the provided `targets.txt` file. This is the only file in this analysis workflow that needs to be generated manually, *e.g.* in a spreadsheet program. To import `targets.txt`, we run the following commands from R:

```
> targets <- read.delim("./data/targets.txt")
> targets
```

```
         FileName SampleName Factor Factor_long
1 SRR064154.fastq   AP3_fl4a    AP3     AP3_fl4
2 SRR064155.fastq   AP3_fl4b    AP3     AP3_fl4
3 SRR064166.fastq    Tl_fl4a    TRL      Tl_fl4
4 SRR064167.fastq    Tl_fl4b    TRL      Tl_fl4
```

Outline

Overview
- Workflow
- Software Resources
- Data Formats

VAR-Seq Analysis
- Aligning Short Reads
- Variant Calling
- Annotating Variants
  - Prerequisites for Annotating Variants
  - Working VCF Objects
  - Adding Genomic Context to Variants

Align Reads with BWA and Output Indexed Bam Files

Note: this step requires the command-line tool BWA Link. If it is not available on a system then one can skip this mapping step and use the pre-generated Bam files provided in the results directory of this project.

Index reference genome

```
> library(modules); library(Rsamtools)
> moduleload("bwa/0.7.10") # loads BWA version 0.7.10 from module system
> system("bwa index -a bwtsw ./data/tair10chr.fasta") # Indexes reference genome; required for GATK
```

Read mapping with BWA and SAM to BAM conversion with Rsamtools

```
> dir.create("results") # Note: all output data will be written to results directory
> moduleload("bwa/0.7.10")
> for(i in seq(along=targets[,1])) {
+   system(paste("bwa mem -M -R '@RG\\tID:group1\\tSM:sample1\\tPL:illumina\\tLB:lib1\\tPU:unit1'", " ./data/tair10
+     asBam(file=paste("./results/", targets$FileName[i], ".sam", sep=""), destination=paste("./results/", targets
+     }
```

Align Reads with gsnap from gmapR Package

Index genome for gmap and create GmapGenome object

```
> library(gmapR); library(rtracklayer)
> fastaFile <- FastaFile(paste(getwd(), "/data/tair10chr.fasta", sep="")) # Needs to be full path!
> gmapGenome <- GmapGenome(fastaFile, directory="data", name="gmap_tair10chr/", create=TRUE)
```

Align reads with *gsnap*. See *'?GsnapParam'* for parameter settings.

```
> gmapGenome <- GmapGenome(fastaFile, directory="data", name="gmap_tair10chr/", create=FALSE)
> # To regenerate gmapGenome object, set 'create=FALSE'.
> param <- GsnapParam(genome=gmapGenome, unique_only = TRUE, molecule = "DNA", max_mismatches = 3)
> for(i in seq(along=targets[,1])) {
+ output <- gsnap(input_a=paste("./data/", targets[i,1], sep=""), input_b=NULL, param,
+ output=paste("results/gsnap_bam/", targets[i,1], sep=""))
+ }
```

Outline

Overview
- Workflow
- Software Resources
- Data Formats

VAR-Seq Analysis
- Aligning Short Reads
- Variant Calling
- Annotating Variants
  - Prerequisites for Annotating Variants
  - Working VCF Objects
  - Adding Genomic Context to Variants

Variant Calling with `callVariants` from *VariantTools*

Call variants from BWA alignments with `VariantTools`. Note: most variant calls in the sample data will be PCR artifacts. Those can be removed by filtering on the number of unique read positions for the alternate base, here column `n.read.pos` in `var`.

```
> library(VariantTools); library(gmapR)
> gmapGenome <- GmapGenome(genome="gmap_tair10chr", directory="data")
> tally.param <- TallyVariantsParam(gmapGenome, high_base_quality = 23L, indels = TRUE)
> bfl <- BamFileList(paste("./results/", as.character(targets[,1]), ".bam", sep=""), index=character())
> var <- callVariants(bfl[[1]], tally.param)
> length(var)

[1] 787

> var <- var[totalDepth(var) == altDepth(var) & totalDepth(var)>=5 & values(var)$n.read.pos >= 5] # Some arbitra
> length(var)

[1] 24

> sampleNames(var) <- "bwa"
> vcf <- asVCF(var)
> writeVcf(vcf, "./results/varianttools.vcf", index = TRUE)
```

Call variants from `gsnap` alignments with `VariantTools`

```
> bfl <- BamFileList(paste("./results/gsnap_bam/", as.character(targets[,1]), ".sam", ".bam", sep=""), index=cha
> var_gsnap <- callVariants(bfl[[1]], tally.param)
> var_gsnap <- var_gsnap[totalDepth(var_gsnap) == altDepth(var_gsnap) & totalDepth(var_gsnap)>=5 & values(var_gs
> sampleNames(var_gsnap) <- "gsnap"
> vcf_gsnap <- asVCF(var_gsnap)
> writeVcf(vcf_gsnap, "./results/varianttools_gnsap.vcf", index=TRUE)
```

Run callVariants Stepwise

The callVariants function wraps several other functions. Running them individually provides more control over the variant calling and filtering. The first step is to tally the variants from the BAM file with the tallyVariants function.

```
> raw.variants <- tallyVariants(bfl[[1]], tally.param)
```

The qaVariants function adds a soft filter matrix to the VRanges object generated in the previous step.

```
> qa.variants <- qaVariants(raw.variants)
> softFilterMatrix(qa.variants)[1:2,]
```

```
FilterMatrix (2 x 2)
    mdfne fisherStrand
[1]    NA         TRUE
[2]    NA         TRUE
```

The callVariants function calls the variants using a binomial likelihood ratio test.

```
> called.variants <- callVariants(qa.variants)
> length(called.variants)
```

```
[1] 787
```

VRanges Object Simplifies Variant Quality Filtering

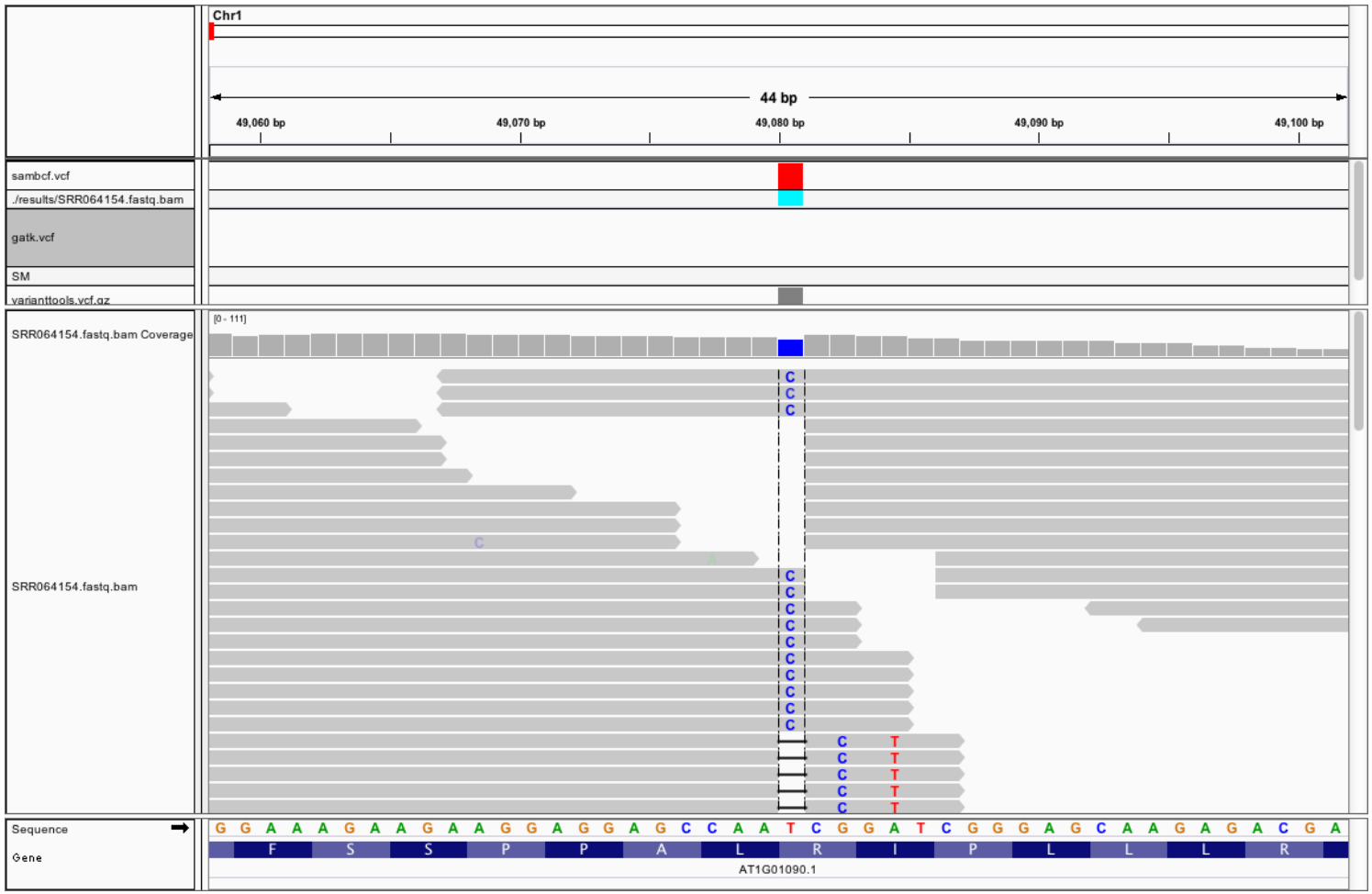
VRanges objects are convenient for SNP quality filtering. They can be easily generated from any external VCF file.

```
> library(VariantAnnotation)
> vcf_imported <- readVcf("results/varianttools.vcf.bgz", "ATH1")
> VRangesFromVCF <- as(vcf_imported, "VRanges")
> VRangesFromVCF[1:4,]
```

```
VRanges object with 4 ranges and 19 metadata columns:
                seqnames         ranges strand         ref               alt
                   <Rle>      <IRanges>  <Rle> <character> <characterOrRle>
  Chr1:49080_T/C     Chr1 [49080, 49080]      +           T                 C
  Chr1:73838_A/G     Chr1 [73838, 73838]      +           A                 G
   Chr2:6110_A/T     Chr2 [ 6110,  6110]      +           A                 T
  Chr2:77574_G/A     Chr2 [77574, 77574]      +           G                 A
                  totalDepth       refDepth       altDepth   sampleNames
                <integerOrRle> <integerOrRle> <integerOrRle> <factorOrRle>
  Chr1:49080_T/C            38              0             38           bwa
  Chr1:73838_A/G            35              0             35           bwa
   Chr2:6110_A/T           550              0            550           bwa
  Chr2:77574_G/A            14              0             14           bwa
                softFilterMatrix |      QUAL n.read.pos n.read.pos.ref
                        <matrix> | <numeric>  <integer>      <integer>
  Chr1:49080_T/C                 |      <NA>         13              0
  Chr1:73838_A/G                 |      <NA>         10              0
   Chr2:6110_A/T                 |      <NA>         47              0
  Chr2:77574_G/A                 |      <NA>          5              0
                raw.count raw.count.ref raw.count.total mean.quality
                <integer>     <integer>       <integer>    <numeric>
  Chr1:49080_T/C        40             0              40      31.0526
  Chr1:73838_A/G        36             0              36      32.5714
   Chr2:6110_A/T       579             0             579      31.5818
  Chr2:77574_G/A        14             0              14      32.5714
                mean.quality.ref count.plus count.plus.ref count.minus
```

View Variants in IGV

Open in IGV *A. thaliana* (TAIR10) genome. Then import SRR064154.fastq.bam and several of the generated VCF files. After loading everything, direct IGV to SNP position: `Chr1:49,080`.



Variant Calling with SAMtools/BCFtools

For details see here Link

```
> library(modules)
> moduleload("samtools")
> dedup <- paste("samtools rmdup -S ", path(bfl[[1]]), " ", path(bfl[[1]]), "ded
> system(dedup) # Removes PCR duplicates with identical read mappings!
> indexBam(file=paste(path(bfl[[1]]), "dedup.bam", sep=""))
> vcf1 <- paste("samtools mpileup -uf ./data/tair10chr.fasta ", path(bfl[[1]]),
+     " | bcftools view -bvcg  -> ./results/sambcf.raw.bcf", sep="")
> vcf2 <- paste("bcftools view ./results/sambcf.raw.bcf",
+     "| vcfutils.pl varFilter -D100 > ./results/sambcf.vcf")
> system(vcf1)
> system(vcf2)
```

Variant Calling with GATK

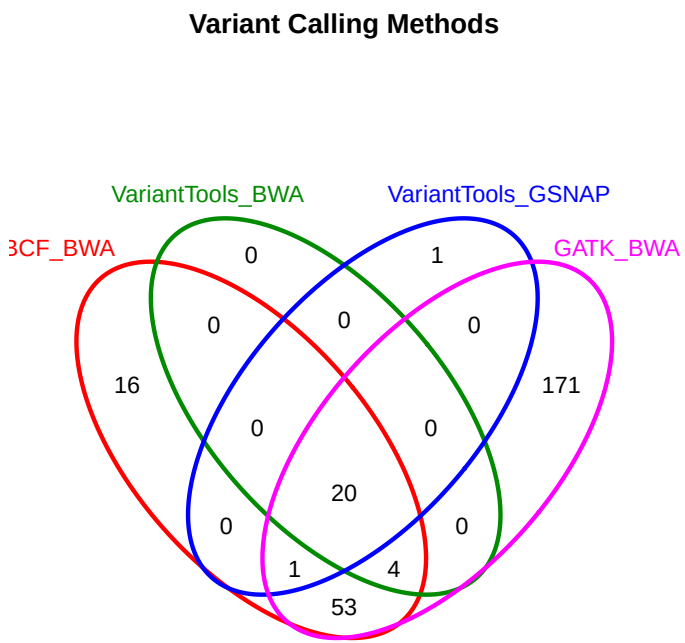
The following runs the GATK variant caller via a bash script: `gatk_runs.sh` Link

```
> library(modules)
> moduleload("java")
> system("java -jar /opt/picard/1.81/CreateSequenceDictionary.jar R=data/tair10chr.f
> dir.create("results/gatktmp", recursive = TRUE)
> file.copy("gatk_runs.sh", "results/gatktmp/gatk_runs.sh")
> file.copy("results/SRR064154.fastq.bam", "results/gatktmp/myfile.fastq.bam")
> setwd("results/gatktmp")
> system("./gatk_runs.sh")
> file.copy("vargatk.recalibrated.filtered.vcf", "../gatk.vcf")
> setwd("../../")
> unlink("results/gatktmp/", recursive=TRUE, force=TRUE)
```

Agreement Among Variant Calling Methods

Compare common and unique variant calls among results from *BCFtools*, *VariantTools* and *GATK*

```
> library(VariantAnnotation)
> vcfsam <- readVcf("results/sambcf.vcf", "ATH1")
> vcfvt <- readVcf("results/varianttools.vcf.bgz", "ATH1")
> vcfvt_gsnap <- readVcf("results/varianttools_gnsap.vcf.bgz", "ATH1")
> vcfgatk <- readVcf("results/gatk.vcf", "ATH1")
> vcfgatk <- vcfgatk[values(rowData(vcfgatk))$FILTER == "PASS"] # Uses GATK filters
> methods <- list(BCF_BWA=names(rowData(vcfsam)), VariantTools_BWA=names(rowData(vcfvt)), VariantTools_GSNAP=
> source("http://faculty.ucr.edu/~tgirke/Documents/R_BioCond/My_R_Scripts/overLapper.R")
> OLlist <- overLapper(setlist=methods, sep="_", type="vennsets")
> counts <- sapply(OLlist$Venn_List, length); vennPlot(counts=counts, mymain="Variant Calling Methods")
```



Exercise 1: Compare Variants Among Four Samples

Task 1 Identify variants in all 4 samples (BAM files) using *VariantTools* in a for loop.

Task 2 Compare the common and unique variants in a venn diagram.

Task 3 Extract the variant IDs that are common in all four samples.

Outline

Overview
- Workflow
- Software Resources
- Data Formats

VAR-Seq Analysis
- Aligning Short Reads
- Variant Calling
- Annotating Variants
  - Prerequisites for Annotating Variants
  - Working VCF Objects
  - Adding Genomic Context to Variants

Prerequisites for Annotating Variants

Requirements: *txdb*, *vcf* and *fa*

txdb: Annotation data as *TranscriptDb* object, here obtained from GFF3 file. Alternative sources: BioMart, Bioc Annotation packages, UCSC, etc.

```
> library(GenomicFeatures)
> chrominfo <- data.frame(chrom=c("Chr1", "Chr2", "Chr3", "Chr4", "Chr5", "ChrC
> txdb <- makeTranscriptDbFromGFF(file="data/TAIR10_GFF3_trunc.gff",
+         format="gff3",
+         dataSource="TAIR",
+         chrominfo=chrominfo,
+         species="Arabidopsis thaliana")
> saveDb(txdb, file="./data/TAIR10.sqlite")
> txdb <- loadDb("./data/TAIR10.sqlite")
```

vcf: Variant data (note: `seqlevels` need to match between *vcf* and *txdb*)

```
> library(VariantAnnotation)
> vcf <- readVcf("results/varianttools_gnsap.vcf.bgz", "ATH1")
> seqlengths(vcf) <- seqlengths(txdb)[names(seqlengths(vcf))]; isCircular(vcf)
```

fa: Genome sequence. Can be *FaFile* object pointing to FASTA file or *BSgenome* instance.

```
> library(Rsamtools)
> fa <- FaFile("data/tair10chr.fasta")
```

Working with Variant Call Format (VCF) Objects

Import VCF file into *VCF* container

```
> vcf <- readVcf("results/sambcf.vcf", "ATH1")
> seqlengths(vcf) <- seqlengths(txdb)[names(seqlengths(vcf))]; isCircular(vcf) <- isCircular(txdb)[names(se
```

Important arguments of `readVcf`:

`file`	path to VCF file or TabixFile instance
`genome`	genome identifier
`param`	range object (*e.g.* GRanges) for importing lines of VCF file mapping to specified genomic regions

```
> seqinfo(vcf)

Seqinfo object with 7 sequences from ATH1 genome:
  seqnames seqlengths isCircular genome
  Chr1         100000      FALSE   ATH1
  Chr2         100000      FALSE   ATH1
  Chr3         100000      FALSE   ATH1
  Chr4         100000      FALSE   ATH1
  Chr5         100000      FALSE   ATH1
  ChrC         100000      FALSE   ATH1
  ChrM         100000      FALSE   ATH1

> genome(vcf)

  Chr1   Chr2   Chr3   Chr4   Chr5   ChrC   ChrM
"ATH1" "ATH1" "ATH1" "ATH1" "ATH1" "ATH1" "ATH1"
```

Meta/Header Components of VCF

```
> header(vcf)
class: VCFHeader
samples(1): sample1
meta(3): fileformat samtoolsVersion reference
fixed(0):
info(24): DP DP4 ... MDV VDB
geno(7): GT GQ ... SP PL
> meta(header(vcf))
DataFrame with 3 rows and 1 column
                                       Value
                                 <character>
fileformat                           VCFv4.1
samtoolsVersion                0.1.19-44428cd
reference       file:///./data/tair10chr.fasta
> info(header(vcf))[1:3,]
DataFrame with 3 rows and 3 columns
         Number        Type
    <character> <character>
DP            1     Integer
DP4           4     Integer
MQ            1     Integer
                                                                  Description
                                                                  <character>
DP                                                              Raw read depth
DP4 # high-quality ref-forward bases, ref-reverse, alt-forward and alt-reverse bases
MQ                                 Root-mean-square mapping quality of covering reads
> geno(header(vcf))[1:3,]
DataFrame with 3 rows and 3 columns
        Number        Type                                        Description
   <character> <character>                                        <character>
GT           1      String                                           Genotype
GQ           1     Integer                                   Genotype Quality
```

Data Component of VCF

First 7 columns of VCF data component

```
> rowData(vcf)[1:3,]
```

```
GRanges object with 3 ranges and 5 metadata columns:
                 seqnames         ranges strand | paramRangeID            REF
                    <Rle>      <IRanges>  <Rle> |     <factor> <DNAStringSet>
  Chr1:49080_T/C     Chr1 [49080, 49080]      * |         <NA>              T
  Chr1:49107_A/T     Chr1 [49107, 49107]      * |         <NA>              A
  Chr1:57686_A/C     Chr1 [57686, 57686]      * |         <NA>              A
                              ALT      QUAL      FILTER
                 <DNAStringSetList> <numeric> <character>
  Chr1:49080_T/C                  C     196.0           .
  Chr1:49107_A/T                  T      84.5           .
  Chr1:57686_A/C                  C      29.0           .
  -------
  seqinfo: 7 sequences from ATH1 genome
```

8th column (INFO) of VCF data component, here split into data frame

```
> info(vcf)[1:3,1:6]
```

```
DataFrame with 3 rows and 6 columns
                      DP           DP4        MQ        FQ       AF1       AC1
                <integer> <IntegerList> <integer> <numeric> <numeric> <numeric>
Chr1:49080_T/C         12     0,0,7,...        60    -60.00    1.0000         2
Chr1:49107_A/T          4     0,0,2,...        60    -39.00    1.0000         2
Chr1:57686_A/C          3     1,0,1,...        60     -8.63    0.5032         1
```

Individual columns can be returned by accessors named after the column names: `rownames()`, `start()`, `ref()`, `alt`, `qual()`, etc. For example,

```
> alt(vcf)[1:3,]
```

```
DNAStringSetList of length 3
[[1]] C
[[2]] T
[[3]] C
```

Adding Genomic Context to Variants

Variants overlapping with common annotation features can be identified with `locateVariants`

```
> library(GenomicFeatures)
> vcf <- readVcf(file="results/varianttools_gnsap.vcf.bgz", genome="ATH1")
> seqlengths(vcf) <- seqlengths(txdb)[names(seqlengths(vcf))]; isCircular(vcf) <
> rd <- rowData(vcf)
> codvar <- locateVariants(rd, txdb, CodingVariants())
```

Supported annotation features

Type	Constructor	Definition
coding	`CodingVariants`	falls *within* a coding region
fiveUTR	`FiveUTRVariants`	falls *within* a 5' untranslated region
threeUTR	`ThreeUTRVariants`	falls *within* a 3' untranslated region
intron	`IntronVariants`	falls *within* an intron region
intergenic	`IntergenicVariants`	does not fall *within* gene region
spliceSite	`SpliceSiteVariants`	overlaps first 2 or last 2 nucleotides of an intron
promoter	`PromoterVariants`	falls *within* a promoter region of a transcript
all	`AllVariants`	all of the above

Obtain All Annotations in One Step

Obtain all annotations

```
> allvar <- locateVariants(rd, txdb, AllVariants())
> allvar[1:4]
```

```
GRanges object with 4 ranges and 9 metadata columns:
                 seqnames         ranges strand |   LOCATION  LOCSTART
                    <Rle>      <IRanges>  <Rle> |   <factor> <integer>
                     Chr1 [49080, 49080]      - |     coding        87
  Chr1:73838_A/G     Chr1 [73838, 73838]      * | intergenic      <NA>
                     Chr2 [ 6110,  6110]      + |   promoter      <NA>
   Chr2:6110_A/T     Chr2 [ 6110,  6110]      * | intergenic      <NA>
                    LOCEND   QUERYID      TXID     CDSID      GENEID
                 <integer> <integer> <integer> <integer> <character>
                        87         1        21        80   AT1G01090
  Chr1:73838_A/G      <NA>         2      <NA>      <NA>        <NA>
                      <NA>         3        27      <NA>   AT2G01021
   Chr2:6110_A/T      <NA>         3      <NA>      <NA>        <NA>
                          PRECEDEID                          FOLLOWID
                     <CharacterList>                    <CharacterList>
  Chr1:73838_A/G                    AT1G01010,AT1G01020,AT1G01030,...

   Chr2:6110_A/T AT2G01021,AT2G01023                          AT2G01008
  -------
  seqinfo: 7 sequences from ATH1 genome
```

Generate variant annotation report containing one line per variant and export to file

```
> source("Rvarseq_Fct.R")
> (varreport <- variantReport(allvar, vcf))[1:4,]
```

```
                       VARID            LOCATION     GENEID QUAL
Chr1:49080_T/C Chr1:49080_T/C              coding  AT1G01090   NA
Chr1:73838_A/G Chr1:73838_A/G          intergenic              NA
Chr2:6110_A/T   Chr2:6110_A/T promoter intergenic AT2G01021    NA
Chr2:77574_G/A Chr2:77574_G/A          intergenic              NA
```

```
> write.table(varreport, "results/varreport.xls", row.names=FALSE, quote=FALSE, sep="\t")
```

Consequences of Coding Variants

Synonymous/non-synonymous variants of coding sequences are computed by the `predictCoding` function for variants overlapping with coding regions.

```
> coding <- predictCoding(vcf, txdb, seqSource=fa)
> coding[1:3,c(12,16:17)]
```

```
GRanges object with 3 ranges and 3 metadata columns:
                 seqnames         ranges strand |      GENEID         REFAA
                    <Rle>      <IRanges>  <Rle> | <character> <AAStringSet>
  Chr1:49080_T/C     Chr1 [49080, 49080]      - |   AT1G01090             R
  Chr3:44729_T/G     Chr3 [44729, 44729]      - |   AT3G01130             A
  Chr4:11691_T/A     Chr4 [11691, 11691]      - |   AT4G00026             V
                        VARAA
                 <AAStringSet>
  Chr1:49080_T/C             R
  Chr3:44729_T/G             A
  Chr4:11691_T/A             V
  -------
  seqinfo: 7 sequences from ATH1 genome
```

Generate coding report containing one line per variant and export to file

```
> source("Rvarseq_Fct.R")
> (codereport <- codingReport(coding, txdb))[1:3,]
```

```
                       VARID Strand Consequence       Codon      AA
Chr1:49080_T/C Chr1:49080_T/C      -  synonymous  87_CGA/CGG  29_R/R
Chr3:44729_T/G Chr3:44729_T/G      -  synonymous 147_GCA/GCC  49_A/A
Chr4:11691_T/A Chr4:11691_T/A      -  synonymous 753_GTA/GTT 251_V/V
                     TXIDs    GENEID
Chr1:49080_T/C AT1G01090.1 AT1G01090
Chr3:44729_T/G AT3G01130.1 AT3G01130
Chr4:11691_T/A AT4G00026.1 AT4G00026
```

```
> write.table(codereport, "results/codereport.xls", row.names=FALSE, quote=FALSE, sep="\t")
```

Combine Variant and Coding Annotation Reports

Combine `varreport` and `codereport` in one data frame and export to file

```
> fullreport <- cbind(varreport, codereport[rownames(varreport),-1])
> write.table(fullreport, "results/fullreport.xls", row.names=FALSE, quote=FALSE, sep="\t", na="")
> fullreport[c(1,18),]
```

```
                       VARID   LOCATION    GENEID QUAL Strand Consequence
Chr1:49080_T/C Chr1:49080_T/C     coding AT1G01090   NA      -  synonymous
Chr5:77562_C/T Chr5:77562_C/T intergenic             NA   <NA>        <NA>
                    Codon     AA       TXIDs    GENEID
Chr1:49080_T/C 87_CGA/CGG 29_R/R AT1G01090.1 AT1G01090
Chr5:77562_C/T       <NA>   <NA>        <NA>      <NA>
```

Add Variant Statistics to Annotation Report

Select stats columns from `VRanges` object and append them to the annotation report.

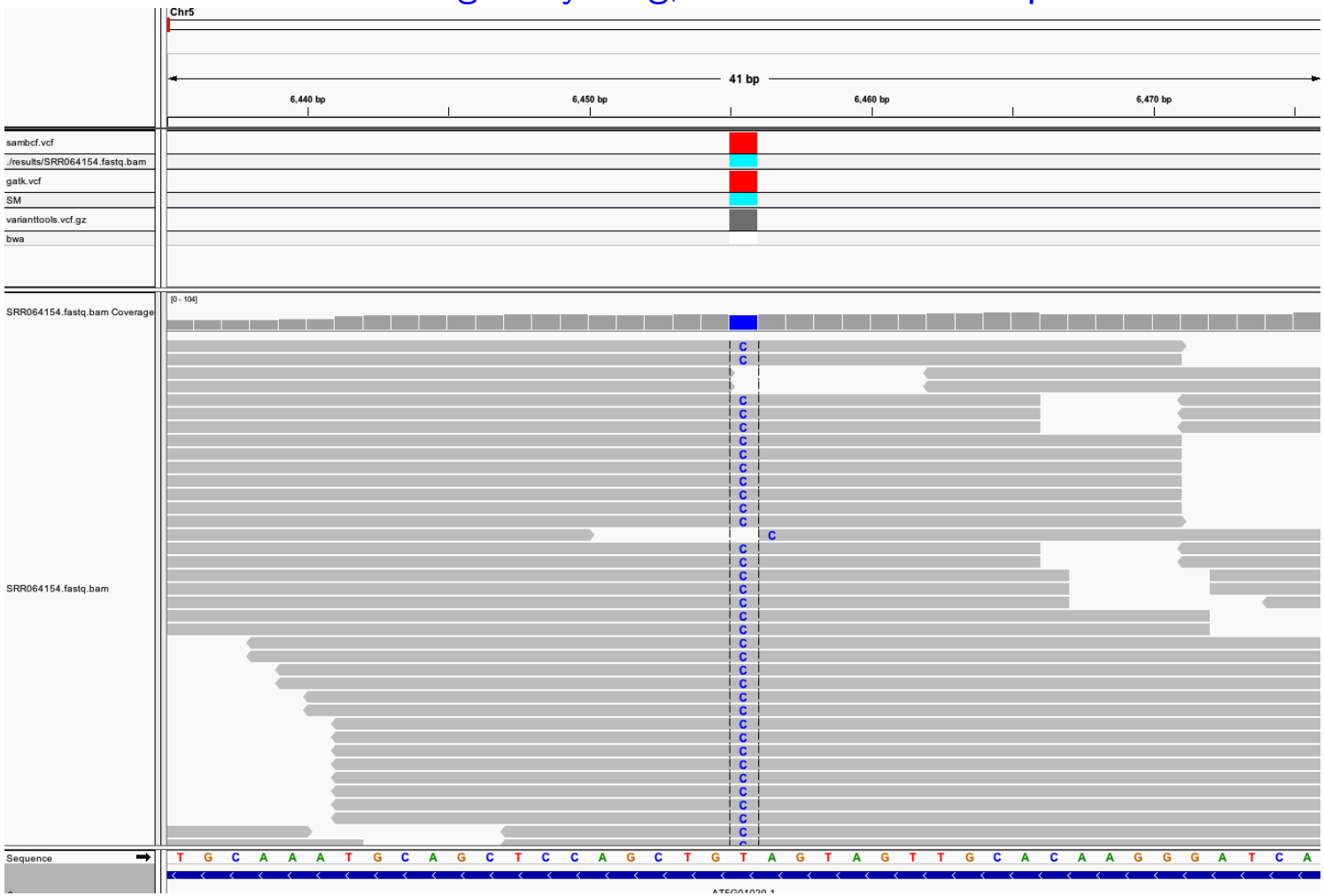
```
> library(VariantTools)
> vr <- as(vcf, "VRanges")
> varid <- paste(as.character(seqnames(vr)), ":", start(vr), "_", ref(vr), "/", alt(vr), sep="")
> vrdf <- data.frame(row.names=varid, as.data.frame(vr))
> vrdf <- vrdf[,c("totalDepth", "refDepth", "altDepth", "n.read.pos", "QUAL", "mean.quality")]
> fullreport <- cbind(VARID=fullreport[,1], vrdf[rownames(fullreport),], fullreport[,-1])
> fullreport[c(1,18),c(1:8,14)]
```

```
                      VARID totalDepth refDepth altDepth n.read.pos QUAL
Chr1:49080_T/C Chr1:49080_T/C         33        0       33         11   NA
Chr5:77562_C/T Chr5:77562_C/T         12        0       12          5   NA
               mean.quality   LOCATION     AA
Chr1:49080_T/C      30.9697     coding 29_R/R
Chr5:77562_C/T      33.0000 intergenic   <NA>
```

```
> write.table(fullreport, "results/fullreport.xls", row.names=FALSE, quote=FALSE, sep="\t", na="")
```

View Nonsynonymous Variant in IGV

Open in IGV *A. thaliana* (TAIR10) genome. Then import SRR064154.fastq.bam and several of the generated VCF files. After loading everything, direct IGV to SNP position: `Chr5:6455`.



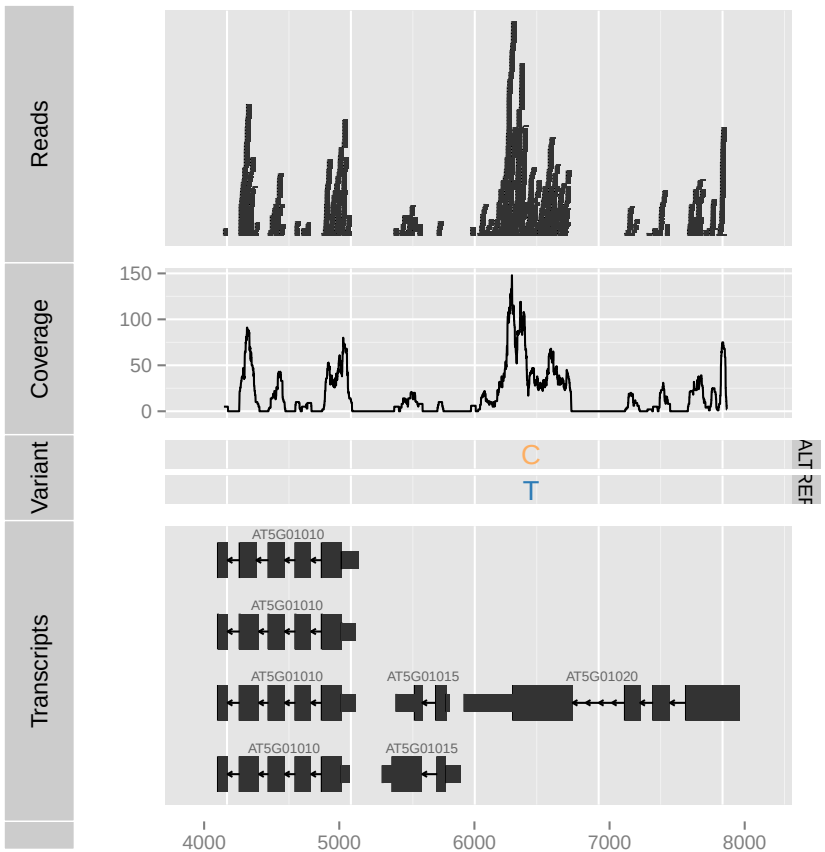
Controlling IGV from R

Create previous IGV session with required tracks automatically, and direct it to a specific position, here `Chr5:6455`.

```
> library(SRAdb)
> startIGV("lm")
> sock <- IGVsocket()
> session <- IGVsession(files=c("results/SRR064154.fastq.bam",
+                               "results/varianttools.vcf.bgz"),
+                               sessionFile="session.xml",
+                               genome="A. thaliana (TAIR10)")
> IGVload(sock, session)
> IGVgoto(sock, 'Chr5:6455')
```

Plot Variant Programmatically with *ggbio*

```
> library(ggbio); library(GenomicAlignments)
> ga <- readGAlignmentsFromBam(path(bfl[[1]]), use.names=TRUE, param=ScanBamParam(which=GRanges("Chr5", IRang
> p1 <- autoplot(ga, geom = "rect")
> p2 <- autoplot(ga, geom = "line", stat = "coverage")
> p3 <- autoplot(vcf[seqnames(vcf)=="Chr5"], type = "fixed") + xlim(4000, 8000) + theme(legend.position = "no
> p4 <- autoplot(txdb, which=GRanges("Chr5", IRanges(4000, 8000)), names.expr = "gene_id")
> tracks(Reads=p1, Coverage=p2, Variant=p3, Transcripts=p4, heights = c(0.3, 0.2, 0.1, 0.35)) + ylab("")
```



Exercise 2: Variant Annotation Report for All Four Samples

Task 1 Generate variant calls for all 4 samples as in Exercise 1.

Task 2 Combine all four reports in one data frame and export it to a tab delimited file.

Session Information

```
> sessionInfo()
R version 3.1.2 (2014-10-31)
Platform: x86_64-unknown-linux-gnu (64-bit)
locale:
[1] C

attached base packages:
[1] parallel  stats4    stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] GenomicAlignments_1.2.1  ggbio_1.14.0             ggplot2_1.0.0
 [4] GenomicFeatures_1.18.2   AnnotationDbi_1.28.1     Biobase_2.26.0
 [7] gmapR_1.8.0              VariantTools_1.8.1       VariantAnnotation_1.12.4
[10] Rsamtools_1.18.2         Biostrings_2.34.0        XVector_0.6.0
[13] GenomicRanges_1.18.3     GenomeInfoDb_1.2.3       IRanges_2.0.0
[16] S4Vectors_0.4.0          BiocGenerics_0.12.1

loaded via a namespace (and not attached):
 [1] BBmisc_1.8          BSgenome_1.34.0     BatchJobs_1.5
 [4] BiocParallel_1.0.0  DBI_0.3.1           Formula_1.1-2
 [7] GGally_0.4.8        Hmisc_3.14-6        MASS_7.3-35
[10] Matrix_1.1-4        OrganismDbi_1.8.0   RBGL_1.42.0
[13] RColorBrewer_1.0-5  RCurl_1.95-4.3      RSQLite_1.0.0
[16] Rcpp_0.11.3         XML_3.98-1.1        acepack_1.3-3.3
[19] base64enc_0.1-2     biomaRt_2.22.0      biovizBase_1.14.0
[22] bitops_1.0-6        brew_1.0-6          checkmate_1.5.0
[25] cluster_1.15.3      codetools_0.2-9     colorspace_1.2-4
[28] dichromat_2.0-0     digest_0.6.4        fail_1.2
[31] foreach_1.4.2       foreign_0.8-61      graph_1.44.0
[34] grid_3.1.2          gridExtra_0.9.1     gtable_0.1.2
[37] iterators_1.0.7     labeling_0.3        lattice_0.20-29
[40] latticeExtra_0.6-26 munsell_0.4.2       nnet_7.3-8
[43] plyr_1.8.1          proto_0.3-10        reshape_0.8.5
[46] reshape2_1.4        rpart_4.1-8         rtracklayer_1.26.2
```