VAR-Seq Workflow Template

Author: Daniela Cassol (danielac@ucr.edu) and Thomas Girke (thomas.girke@ucr.edu)

Last update: 16 April, 2019

Package

systemPipeR 1.17.9

Contents

1 Introduction

Users want to provide here background information about the design of their VAR-Seq project.

1.1 Background and objectives

This report describes the analysis of a VAR-Seq project studying the genetic differences among several strains . . . from *organism* . . . .

1.2 Experimental design

Typically, users want to specify here all information relevant for the analysis of their NGS study. This includes detailed descriptions of FASTQ files, experimental design, reference genome, gene annotations, etc.

2 Workflow environment

2.1 Generate workflow environment

Load workflow environment with sample data into your current working directory. The sample data are described here.

```
library(systemPipeRdata)
genWorkenvir(workflow = "varseq")
setwd("varseq")
```

Alternatively, this can be done from the command-line as follows:

```
Rscript -e "systemPipeRdata::genWorkenvir(workflow='varseq')"
```

In the workflow environments generated by `genWorkenvir` all data inputs are stored in a `data/` directory and all analysis results will be written to a separate `results/` directory, while the `systemPipeVARseq.Rmd` script and the `targets` file are expected to be located in the parent directory. The R session is expected to run from this parent directory. Additional parameter files are stored under `param/`.

To work with real data, users want to organize their own data similarly and substitute all test data for their own data. To rerun an established workflow on new data, the initial `targets` file along with the corresponding FASTQ files are usually the only inputs the user needs to provide.

2.2 Run workflow

Now open the R markdown script `systemPipeVARseq.Rmd`in your R IDE (*e.g.* vim-r or RStudio) and run the workflow as outlined below.

2.2.1 Run R session on computer node

After opening the `Rmd` file of this workflow in Vim and attaching a connected R session via the `F2` (or other) key, use the following command sequence to run your R session on a computer node.

```
q("no")  # closes R session on head node
```

```
srun --x11 --partition=short --mem=2gb --cpus-per-task 4 --ntasks 1 --time 2:00:00 --pty bash -l
module load R/3.4.2
R
```

Now check whether your R session is running on a computer node of the cluster and assess your environment.

```
system("hostname")  # should return name of a compute node starting with i or c
getwd()  # checks current working directory of R session
dir()  # returns content of current working directory
```

The `systemPipeR` package needs to be loaded to perform the analysis steps shown in this report (H Backman and Girke 2016).

```
library(systemPipeR)
```

If applicable users can load custom functions not provided by `systemPipeR`. Skip this step if this is not the case.

```
source("systemPipeVARseq_Fct.R")
```

3 Read preprocessing

3.1 Experiment definition provided by `targets` file

The `targets` file defines all FASTQ files and sample comparisons of the analysis workflow.

```
targetspath <- system.file("extdata", "targetsPE.txt", package = "systemPipeR")
targets <- read.delim(targetspath, comment.char = "#")
targets[1:4, 1:4]
##                     FileName1                   FileName2
## 1 ./data/SRR446027_1.fastq.gz ./data/SRR446027_2.fastq.gz
## 2 ./data/SRR446028_1.fastq.gz ./data/SRR446028_2.fastq.gz
## 3 ./data/SRR446029_1.fastq.gz ./data/SRR446029_2.fastq.gz
## 4 ./data/SRR446030_1.fastq.gz ./data/SRR446030_2.fastq.gz
##   SampleName Factor
## 1        M1A     M1
## 2        M1B     M1
## 3        A1A     A1
## 4        A1B     A1
```

3.2 Read quality filtering and trimming

The following removes reads with low quality base calls (here Phred scores below 20) from all FASTQ files.

```
args <- systemArgs(sysma = "param/trimPE.param", mytargets = "targetsPE.txt")[1:4]
# Note: subsetting!
filterFct <- function(fq, cutoff = 20, Nexceptions = 0) {
    qcount <- rowSums(as(quality(fq), "matrix") <= cutoff, na.rm = TRUE)
    fq[qcount <= Nexceptions]
    # Retains reads where Phred scores are >= cutoff with N
    # exceptions
}
preprocessReads(args = args, Fct = "filterFct(fq, cutoff=20, Nexceptions=0)",
    batchsize = 1e+05)
writeTargetsout(x = args, file = "targets_PEtrim.txt", overwrite = TRUE)
```

3.3 FASTQ quality report

The following `seeFastq` and `seeFastqPlot` functions generate and plot a series of useful quality statistics for a set of FASTQ files including per cycle quality box plots, base proportions, base-level quality trends, relative k-mer diversity, length and occurrence distribution of reads, number of reads above quality cutoffs and mean quality distribution. The results are written to a PDF file named `fastqReport.pdf`.

```
args <- systemArgs(sysma = "param/tophat.param", mytargets = "targets.txt")
fqlist <- seeFastq(fastq = infile1(args), batchsize = 1e+05,
    klength = 8)
pdf("./results/fastqReport.pdf", height = 18, width = 4 * length(fqlist))
seeFastqPlot(fqlist)
dev.off()
```

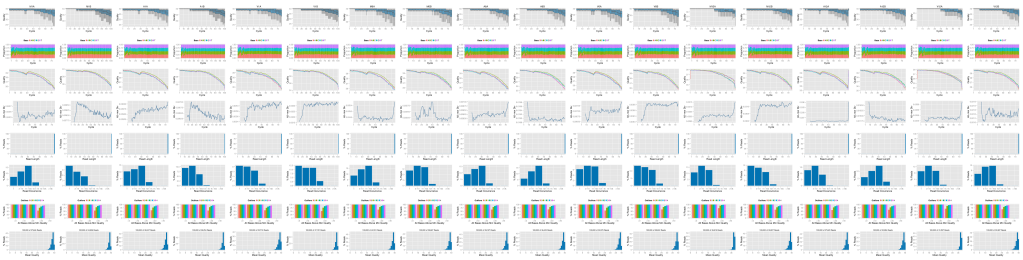

Figure 1: FASTQ quality report for 18 samples

4 Alignments

4.1 Read mapping with `BWA-MEM`

The NGS reads of this project are aligned against the reference genome sequence using the highly variant tolerant short read aligner `BWA-MEM` (Heng Li 2013; H Li and Durbin 2009). The parameter settings of the aligner are defined in the `bwa.param` file.

```
args <- systemArgs(sysma = "param/bwa.param", mytargets = "targets.txt")
sysargs(args)[1]  # Command-line parameters for first FASTQ file
```

Runs the alignments sequentially (*e.g.* on a single machine)

```
moduleload(modules(args))
system("bwa index -a bwtsw ./data/tair10.fasta")
bampaths <- runCommandline(args = args)
writeTargetsout(x = args, file = "targets_bam.txt", overwrite = TRUE)
```

Alternatively, the alignment jobs can be submitted to a compute cluster, here using 72 CPU cores (18 `qsub` processes each with 4 CPU cores).

```
moduleload(modules(args))
system("bwa index -a bwtsw ./data/tair10.fasta")
resources <- list(walltime = 120, ntasks = 1, ncpus = cores(args),
    memory = 1024)
reg <- clusterRun(args, conffile = ".batchtools.conf.R", Njobs = 18,
    template = "batchtools.slurm.tmpl", runid = "01", resourceList = resources)
getStatus(reg = reg)
waitForJobs(reg = reg)
writeTargetsout(x = args, file = "targets_bam.txt", overwrite = TRUE)
```

Check whether all BAM files have been created

```
file.exists(outpaths(args))
```

4.2 Read mapping with `gsnap`

An alternative variant tolerant aligner is `gsnap` from the `gmapR` package (Wu and Nacu 2010). The following code shows how to run this aligner on multiple nodes of a computer cluster that uses Torque as scheduler.

```
library(gmapR)
library(BiocParallel)
library(batchtools)
args <- systemArgs(sysma = "param/gsnap.param", mytargets = "targetsPE.txt")
gmapGenome <- GmapGenome(systemPipeR::reference(args), directory = "data",
    name = "gmap_tair10chr", create = TRUE)
f <- function(x) {
    library(gmapR)
    library(systemPipeR)
```

```
    args <- systemArgs(sysma = "param/gsnap.param", mytargets = "targetsPE.txt")
    gmapGenome <- GmapGenome(reference(args), directory = "data",
        name = "gmap_tair10chr", create = FALSE)
    p <- GsnapParam(genome = gmapGenome, unique_only = TRUE,
        molecule = "DNA", max_mismatches = 3)
    o <- gsnap(input_a = infile1(args)[x], input_b = infile2(args)[x],
        params = p, output = outfile1(args)[x])
}
resources <- list(walltime = 120, ntasks = 1, ncpus = cores(args),
    memory = 1024)
param <- BatchtoolsParam(workers = 4, cluster = "slurm", template = "batchtools.slurm.tmpl",
    resources = resources)
d <- bplapply(seq(along = args), f, BPPARAM = param)
writeTargetsout(x = args, file = "targets_gsnap_bam.txt", overwrite = TRUE)
```

4.3 Read and alignment stats

The following generates a summary table of the number of reads in each sample and how many of them aligned to the reference.

```
read_statsDF <- alignStats(args = args)
write.table(read_statsDF, "results/alignStats.xls", row.names = FALSE,
    quote = FALSE, sep = "\t")
```

4.4 Create symbolic links for viewing BAM files in IGV

The `symLink2bam` function creates symbolic links to view the BAM alignment files in a genome browser such as IGV. The corresponding URLs are written to a file with a path specified under `urlfile`, here `IGVurl.txt`.

```
symLink2bam(sysargs = args, htmldir = c("~/.html/", "projects/gen242/"),
    urlbase = "http://biocluster.ucr.edu/~tgirke/", urlfile = "./results/IGVurl.txt")
```

5 Variant calling

The following performs variant calling with `GATK`, `BCFtools` and `VariantTools` in parallel mode on a compute cluster (McKenna et al. 2010; Heng Li 2011). If a cluster is not available, the `runCommandline` function can be used to run the variant calling with `GATK` and `BCFtools` for each sample sequentially on a single machine, or `callVariants` in case of `VariantTools`. Typically, the user would choose here only one variant caller rather than running several ones.

5.1 Variant calling with `GATK`

The following creates in the inital step a new `targets` file (`targets_bam.txt`). The first column of this file gives the paths to the BAM files created in the alignment step. The new targets file and the parameter file `gatk.param` are used to create a new `SYSargs` instance for running GATK. Since GATK involves many processing steps, it is executed by a bash script `gatk_run.sh` where the user can specify the detailed run parameters. All three files are expected to be located in the current working directory. Samples files for `gatk.param` and `gatk_run.sh` are available in the `param` subdirectory provided by `systemPipeRdata`.

```
moduleload("picard/1.130")
moduleload("samtools/1.3")
system("picard CreateSequenceDictionary R=./data/tair10.fasta O=./data/tair10.dict")
system("samtools faidx data/tair10.fasta")
args <- systemArgs(sysma = "param/gatk.param", mytargets = "targets_bam.txt")
resources <- list(walltime = 120, ntasks = 1, ncpus = 4, memory = 1024)
reg <- clusterRun(args, conffile = ".batchtools.conf.R", Njobs = 18,
    template = "batchtools.slurm.tmpl", runid = "01", resourceList = resources)
getStatus(reg = reg)
waitForJobs(reg = reg)
# unlink(outfile1(args), recursive = TRUE, force = TRUE)
writeTargetsout(x = args, file = "targets_gatk.txt", overwrite = TRUE)
```

5.2 Variant calling with `BCFtools`

The following runs the variant calling with `BCFtools`. This step requires in the current working directory the parameter file `sambcf.param` and the bash script `sambcf_run.sh`.

```
args <- systemArgs(sysma = "param/sambcf.param", mytargets = "targets_bam.txt")
resources <- list(walltime = 120, ntasks = 1, ncpus = 4, memory = 1024)
reg <- clusterRun(args, conffile = ".batchtools.conf.R", Njobs = 18,
    template = "batchtools.slurm.tmpl", runid = "01", resourceList = resources)
getStatus(reg = reg)
waitForJobs(reg = reg)
# unlink(outfile1(args), recursive = TRUE, force = TRUE)
writeTargetsout(x = args, file = "targets_sambcf.txt", overwrite = TRUE)
```

5.3 Variant calling with `VariantTools`

```
library(gmapR)
library(BiocParallel)
library(batchtools)
args <- systemArgs(sysma = "param/vartools.param", mytargets = "targets_gsnap_bam.txt")
f <- function(x) {
    library(VariantTools)
    library(gmapR)
    library(systemPipeR)
    args <- systemArgs(sysma = "param/vartools.param", mytargets = "targets_gsnap_bam.txt")
```

```
    gmapGenome <- GmapGenome(systemPipeR::reference(args), directory = "data",
        name = "gmap_tair10chr", create = FALSE)
    tally.param <- TallyVariantsParam(gmapGenome, high_base_quality = 23L,
        indels = TRUE)
    bfl <- BamFileList(infile1(args)[x], index = character())
    var <- callVariants(bfl[[1]], tally.param)
    sampleNames(var) <- names(bfl)
    writeVcf(asVCF(var), outfile1(args)[x], index = TRUE)
}
resources <- list(walltime = 120, ntasks = 1, ncpus = cores(args),
    memory = 1024)
param <- BatchtoolsParam(workers = 4, cluster = "slurm", template = "batchtools.slurm.tmpl",
    resources = resources)
d <- bplapply(seq(along = args), f, BPPARAM = param)
writeTargetsout(x = args, file = "targets_vartools.txt", overwrite = TRUE)
```

5.4 Inspect VCF file

VCF files can be imported into R with the `readVcf` function. Both `VCF` and `VRanges` objects provide convenient data structure for working with variant data (*e.g.* SNP quality filtering).

```
library(VariantAnnotation)
args <- systemArgs(sysma = "param/filter_gatk.param", mytargets = "targets_gatk.txt")
vcf <- readVcf(infile1(args)[1], "A. thaliana")
vcf
vr <- as(vcf, "VRanges")
vr
```

6 Filter variants

The function `filterVars` filters VCF files based on user definable quality parameters. It sequentially imports each VCF file into R, applies the filtering on an internally generated `VRanges` object and then writes the results to a new subsetted VCF file. The filter parameters are passed on to the corresponding argument as a character string. The function applies this filter to the internally generated `VRanges` object using the standard subsetting syntax for two dimensional objects such as: `vr[filter, ]`. The parameter files (`filter_gatk.param`, `filter_sambcf.param` and `filter_vartools.param`), used in the filtering steps, define the paths to the input and output VCF files which are stored in new `SYSargs` instances.

6.1 Filter variants called by `GATK`

The below example filters for variants that are supported by `>=x` reads and >=80% of them support the called variants. In addition, all variants need to pass `>=x` of the soft filters recorded in the VCF files generated by GATK. Since the toy data used for this workflow is very small, the chosen settings are unreasonabley relaxed. A more reasonable filter setting is given in the line below (here commented out).

```
library(VariantAnnotation)
library(BBmisc)  # Defines suppressAll()
args <- systemArgs(sysma = "param/filter_gatk.param", mytargets = "targets_gatk.txt")[1:4]
filter <- "totalDepth(vr) >= 2 & (altDepth(vr) / totalDepth(vr) >= 0.8) & rowSums(softFilterMatrix(vr))>=1"
# filter <- 'totalDepth(vr) >= 20 & (altDepth(vr) /
# totalDepth(vr) >= 0.8) & rowSums(softFilterMatrix(vr))==6'
suppressAll(filterVars(args, filter, varcaller = "gatk", organism = "A. thaliana"))
writeTargetsout(x = args, file = "targets_gatk_filtered.txt",
    overwrite = TRUE)
```

6.2 Filter variants called by `BCFtools`

The following shows how to filter the VCF files generated by `BCFtools` using similar parameter settings as in the previous filtering of the GATK results.

```
args <- systemArgs(sysma = "param/filter_sambcf.param", mytargets = "targets_sambcf.txt")[1:4]
filter <- "rowSums(vr) >= 2 & (rowSums(vr[,3:4])/rowSums(vr[,1:4]) >= 0.8)"
# filter <- 'rowSums(vr) >= 20 &
# (rowSums(vr[,3:4])/rowSums(vr[,1:4]) >= 0.8)'
suppressAll(filterVars(args, filter, varcaller = "bcftools",
    organism = "A. thaliana"))
writeTargetsout(x = args, file = "targets_sambcf_filtered.txt",
    overwrite = TRUE)
```

6.3 Filter variants called by `VariantTools`

The following shows how to filter the VCF files generated by `VariantTools` using similar parameter settings as in the previous filtering of the GATK results.

```
library(VariantAnnotation)
library(BBmisc)  # Defines suppressAll()
args <- systemArgs(sysma = "param/filter_vartools.param", mytargets = "targets_vartools.txt")[1:4]
filter <- "(values(vr)$n.read.pos.ref + values(vr)$n.read.pos) >= 2 & (values(vr)$n.read.pos / (values(vr)$n
# filter <- '(values(vr)$n.read.pos.ref +
# values(vr)$n.read.pos) >= 20 & (values(vr)$n.read.pos /
# (values(vr)$n.read.pos.ref + values(vr)$n.read.pos) >=
# 0.8)'
filterVars(args, filter, varcaller = "vartools", organism = "A. thaliana")
writeTargetsout(x = args, file = "targets_vartools_filtered.txt",
    overwrite = TRUE)
```

Check filtering outcome for one sample

```
length(as(readVcf(infile1(args)[1], genome = "Ath"), "VRanges")[,
    1])
length(as(readVcf(outpaths(args)[1], genome = "Ath"), "VRanges")[,
    1])
```

7 Annotate filtered variants

The function `variantReport` generates a variant report using utilities provided by the `VariantAnnotation` package. The report for each sample is written to a tabular file containing genomic context annotations (*e.g.* coding or non-coding SNPs, amino acid changes, IDs of affected genes, etc.) along with confidence statistics for each variant. The parameter file `annotate_vars.param` defines the paths to the input and output files which are stored in a new `SYSargs` instance.

7.1 Basics of annotating variants

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

```
library("GenomicFeatures")
args <- systemArgs(sysma = "param/annotate_vars.param", mytargets = "targets_gatk_filtered.txt")
txdb <- loadDb("./data/tair10.sqlite")
vcf <- readVcf(infile1(args)[1], "A. thaliana")
locateVariants(vcf, txdb, CodingVariants())
```

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

```
fa <- FaFile(systemPipeR::reference(args))
predictCoding(vcf, txdb, seqSource = fa)
```

7.2 Annotate filtered variants called by `GATK`

```
library("GenomicFeatures")
args <- systemArgs(sysma = "param/annotate_vars.param", mytargets = "targets_gatk_filtered.txt")
txdb <- loadDb("./data/tair10.sqlite")
fa <- FaFile(systemPipeR::reference(args))
suppressAll(variantReport(args = args, txdb = txdb, fa = fa,
    organism = "A. thaliana"))
```

7.3 Annotate filtered variants called by `BCFtools`

```
args <- systemArgs(sysma = "param/annotate_vars.param", mytargets = "targets_sambcf_filtered.txt")
txdb <- loadDb("./data/tair10.sqlite")
fa <- FaFile(systemPipeR::reference(args))
suppressAll(variantReport(args = args, txdb = txdb, fa = fa,
    organism = "A. thaliana"))
```

7.4 Annotate filtered variants called by `VariantTools`

```
args <- systemArgs(sysma = "param/annotate_vars.param", mytargets = "targets_vartools_filtered.txt")
txdb <- loadDb("./data/tair10.sqlite")
fa <- FaFile(systemPipeR::reference(args))
suppressAll(variantReport(args = args, txdb = txdb, fa = fa,
    organism = "A. thaliana"))
```

View annotation result for single sample

```
read.delim(outpaths(args)[1])[38:40, ]
```

8 Combine annotation results among samples

To simplify comparisons among samples, the `combineVarReports` function combines all variant annotation reports referenced in a `SYSargs` instance (here `args`). At the same time the function allows to consider only certain feature types of interest. For instance, the below setting `filtercol=c(Consequence="nonsynonymous")` will include only nonsysynonymous variances listed in the `Consequence` column of the annotation reports. To omit filtering, one can use the setting `filtercol="All"`.

8.1 Combine results from `GATK`

```
args <- systemArgs(sysma = "param/annotate_vars.param", mytargets = "targets_gatk_filtered.txt")
combineDF <- combineVarReports(args, filtercol = c(Consequence = "nonsynonymous"))
write.table(combineDF, "./results/combineDF_nonsyn_gatk.xls",
    quote = FALSE, row.names = FALSE, sep = "\t")
```

8.2 Combine results from `BCFtools`

```
args <- systemArgs(sysma = "param/annotate_vars.param", mytargets = "targets_sambcf_filtered.txt")
combineDF <- combineVarReports(args, filtercol = c(Consequence = "nonsynonymous"))
write.table(combineDF, "./results/combineDF_nonsyn_sambcf.xls",
    quote = FALSE, row.names = FALSE, sep = "\t")
```

8.3 Combine results from `VariantTools`

```
args <- systemArgs(sysma = "param/annotate_vars.param", mytargets = "targets_vartools_filtered.txt")
combineDF <- combineVarReports(args, filtercol = c(Consequence = "nonsynonymous"))
write.table(combineDF, "./results/combineDF_nonsyn_vartools.xls",
    quote = FALSE, row.names = FALSE, sep = "\t")
combineDF[2:4, ]
```

9 Summary statistics of variants

The `varSummary` function counts the number of variants for each feature type included in the anntation reports.

9.1 Summary for `GATK`

```
args <- systemArgs(sysma = "param/annotate_vars.param", mytargets = "targets_gatk_filtered.txt")
varSummary(args)
write.table(varSummary(args), "./results/variantStats_gatk.xls",
    quote = FALSE, col.names = NA, sep = "\t")
```

9.2 Summary for `BCFtools`

```
args <- systemArgs(sysma = "param/annotate_vars.param", mytargets = "targets_sambcf_filtered.txt")
varSummary(args)
write.table(varSummary(args), "./results/variantStats_sambcf.xls",
    quote = FALSE, col.names = NA, sep = "\t")
```

9.3 Summary for `VariantTools`

```
args <- systemArgs(sysma = "param/annotate_vars.param", mytargets = "targets_vartools_filtered.txt")
varSummary(args)
write.table(varSummary(args), "./results/variantStats_vartools.xls",
    quote = FALSE, col.names = NA, sep = "\t")
```

10 Venn diagram of variants

The venn diagram utilities defined by the `systemPipeR` package can be used to identify common and unique variants reported for different samples and/or variant callers. The below generates a 4-way venn diagram comparing four sampes for each of the two variant callers.

```
args <- systemArgs(sysma = "param/annotate_vars.param", mytargets = "targets_gatk_filtered.txt")
varlist <- sapply(names(outpaths(args))[1:4], function(x) as.character(read.delim(outpaths(args)[x])$VARID))
vennset_gatk <- overLapper(varlist, type = "vennsets")
args <- systemArgs(sysma = "param/annotate_vars.param", mytargets = "targets_sambcf_filtered.txt")
varlist <- sapply(names(outpaths(args))[1:4], function(x) as.character(read.delim(outpaths(args)[x])$VARID))
vennset_bcf <- overLapper(varlist, type = "vennsets")
args <- systemArgs(sysma = "param/annotate_vars.param", mytargets = "targets_vartools_filtered.txt")
varlist <- sapply(names(outpaths(args))[1:4], function(x) as.character(read.delim(outpaths(args)[x])$VARID))
vennset_vartools <- overLapper(varlist, type = "vennsets")
pdf("./results/vennplot_var.pdf")
```

```
vennPlot(list(vennset_gatk, vennset_bcf, vennset_vartools), mymain = "",
    mysub = "GATK: red; BCFtools: blue; VariantTools: green",
    colmode = 2, ccol = c("red", "blue", "green"))
dev.off()
```

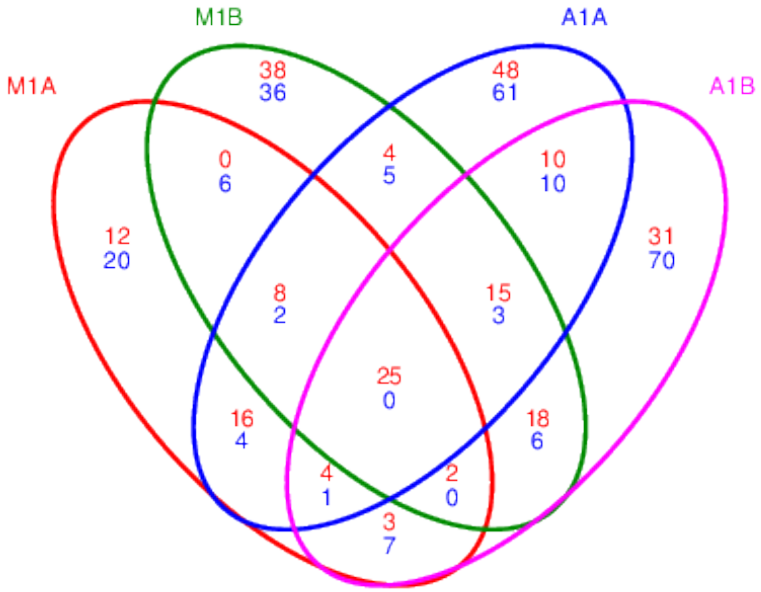

Figure 2: Venn Diagram for 4 samples from GATK and BCFtools

11 Plot variants programmatically

The following plots a selected variant with `ggbio`.

```
library(ggbio)
mychr <- "ChrC"
mystart <- 11000
myend <- 13000
args <- systemArgs(sysma = "param/bwa.param", mytargets = "targets.txt")
ga <- readGAlignments(outpaths(args)[1], use.names = TRUE, param = ScanBamParam(which = GRanges(mychr,
```

```
        IRanges(mystart, myend))))
p1 <- autoplot(ga, geom = "rect")
p2 <- autoplot(ga, geom = "line", stat = "coverage")
p3 <- autoplot(vcf[seqnames(vcf) == mychr], type = "fixed") +
    xlim(mystart, myend) + theme(legend.position = "none", axis.text.y = element_blank(),
    axis.ticks.y = element_blank())
p4 <- autoplot(txdb, which = GRanges(mychr, IRanges(mystart,
    myend)), names.expr = "gene_id")
png("./results/plot_variant.png")
tracks(Reads = p1, Coverage = p2, Variant = p3, Transcripts = p4,
    heights = c(0.3, 0.2, 0.1, 0.35)) + ylab("")
dev.off()
```

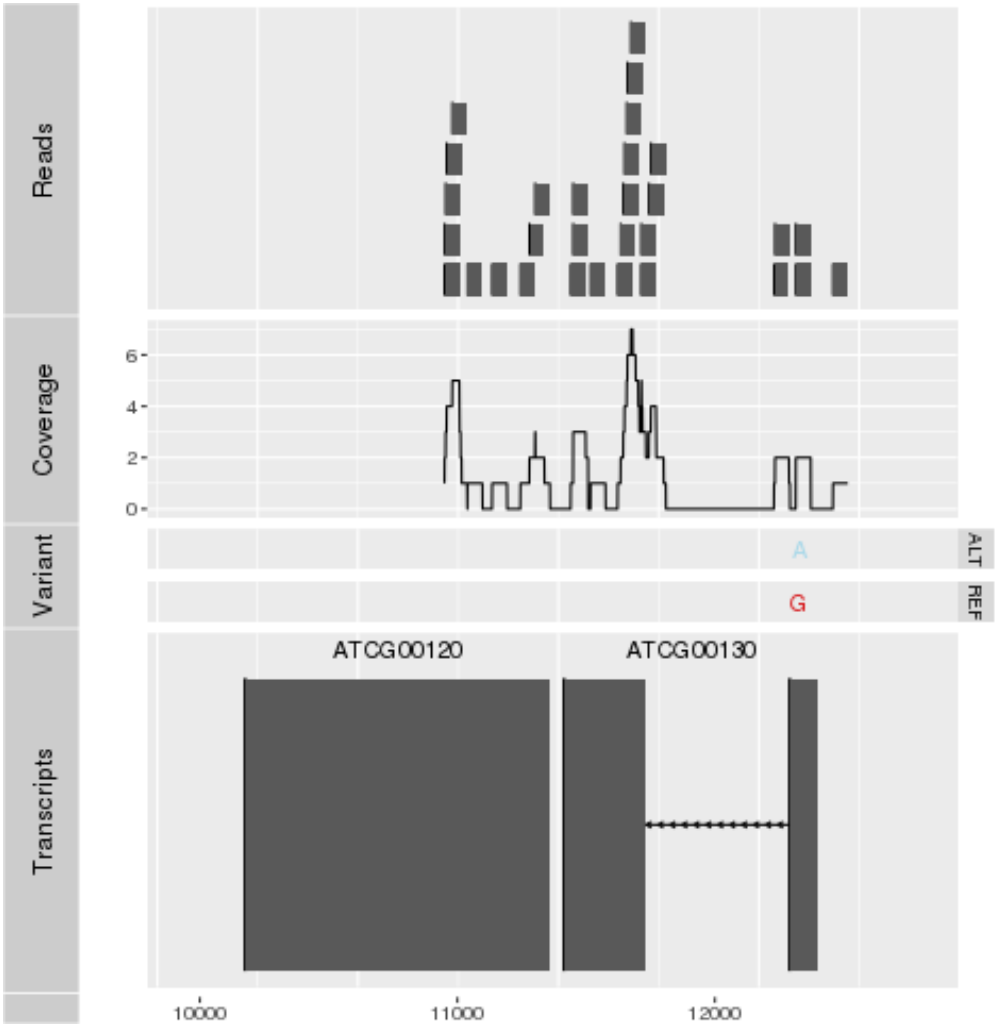

Figure 3: Plot variants with programmatically.

12 Version Information

```
sessionInfo()
## R Under development (unstable) (2019-04-03 r76310)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.2 LTS
##
## Matrix products: default
## BLAS:   /usr/local/lib/R/lib/libRblas.so
## LAPACK: /usr/local/lib/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices
## [6] utils     datasets  methods   base
##
## other attached packages:
##  [1] batchtools_0.9.11           data.table_1.12.0
##  [3] ape_5.3                     ggplot2_3.1.0
##  [5] systemPipeR_1.17.9          ShortRead_1.41.0
##  [7] GenomicAlignments_1.19.1    SummarizedExperiment_1.13.0
##  [9] DelayedArray_0.9.9          matrixStats_0.54.0
## [11] Biobase_2.43.1              BiocParallel_1.17.18
## [13] Rsamtools_1.99.5            Biostrings_2.51.5
## [15] XVector_0.23.2              GenomicRanges_1.35.1
## [17] GenomeInfoDb_1.19.3         IRanges_2.17.4
## [19] S4Vectors_0.21.22           BiocGenerics_0.29.2
## [21] BiocStyle_2.11.0
##
## loaded via a namespace (and not attached):
##  [1] nlme_3.1-137              Category_2.49.1
##  [3] bitops_1.0-6              bit64_0.9-7
##  [5] RColorBrewer_1.1-2        progress_1.2.0
##  [7] httr_1.4.0                Rgraphviz_2.27.0
##  [9] tools_3.7.0               backports_1.1.3
## [11] R6_2.4.0                  DBI_1.0.0
## [13] lazyeval_0.2.2            colorspace_1.4-1
## [15] withr_2.1.2               prettyunits_1.0.2
## [17] bit_1.1-14                compiler_3.7.0
## [19] graph_1.61.1              formatR_1.6
## [21] rtracklayer_1.43.3        bookdown_0.9
## [23] scales_1.0.0              checkmate_1.9.1
## [25] genefilter_1.65.0         RBGL_1.59.5
```

```
## [27] rappdirs_0.3.1            stringr_1.4.0
## [29] digest_0.6.18             rmarkdown_1.12
## [31] AnnotationForge_1.25.0    pkgconfig_2.0.2
## [33] htmltools_0.3.6           BSgenome_1.51.0
## [35] limma_3.39.14             rlang_0.3.3
## [37] RSQLite_2.1.1             GOstats_2.49.0
## [39] hwriter_1.3.2             VariantAnnotation_1.29.25
## [41] RCurl_1.95-4.12           magrittr_1.5
## [43] GO.db_3.7.0               GenomeInfoDbData_1.2.1
## [45] Matrix_1.2-17             Rcpp_1.0.1
## [47] munsell_0.5.0             stringi_1.4.3
## [49] yaml_2.2.0                edgeR_3.25.3
## [51] zlibbioc_1.29.0           plyr_1.8.4
## [53] grid_3.7.0                blob_1.1.1
## [55] crayon_1.3.4              lattice_0.20-38
## [57] splines_3.7.0             GenomicFeatures_1.35.9
## [59] annotate_1.61.1           hms_0.4.2
## [61] locfit_1.5-9.1            knitr_1.22
## [63] pillar_1.3.1              rjson_0.2.20
## [65] base64url_1.4             codetools_0.2-16
## [67] biomaRt_2.39.2            XML_3.98-1.19
## [69] evaluate_0.13             latticeExtra_0.6-28
## [71] BiocManager_1.30.4        gtable_0.3.0
## [73] assertthat_0.2.1          xfun_0.6
## [75] xtable_1.8-3              survival_2.44-1.1
## [77] tibble_2.1.1              pheatmap_1.0.12
## [79] AnnotationDbi_1.45.1      memoise_1.1.0
## [81] brew_1.0-6                GSEABase_1.45.0
```

13 Funding

This project was supported by funds from the National Institutes of Health (NIH) and the National Science Foundation (NSF).

References

H Backman, Tyler W, and Thomas Girke. 2016. "systemPipeR: NGS workflow and report generation environment." *BMC Bioinformatics* 17 (1): 388. doi:10.1186/s12859-016-1241-0.

Li, H, and R Durbin. 2009. "Fast and Accurate Short Read Alignment with Burrows-Wheeler Transform." *Bioinformatics* 25 (14): 1754–60. doi:10.1093/bioinformatics/btp324.

Li, Heng. 2011. "A Statistical Framework for SNP Calling, Mutation Discovery, Association Mapping and Population Genetical Parameter Estimation from Sequencing Data." *Bioinformatics* 27 (21): 2987–93. doi:10.1093/bioinformatics/btr509.

———. 2013. "Aligning Sequence Reads, Clone Sequences and Assembly Contigs with BWA-MEM." *arXiv [Q-bio.GN]*, March. http://arxiv.org/abs/1303.3997.

McKenna, Aaron, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, et al. 2010. "The Genome Analysis Toolkit: A MapReduce Framework for Analyzing Next-Generation DNA Sequencing Data." *Genome Res.* 20 (9): 1297–1303. doi:10.1101/gr.107524.110.

Wu, T D, and S Nacu. 2010. "Fast and SNP-tolerant Detection of Complex Variants and Splicing in Short Reads." *Bioinformatics* 26 (7): 873–81. doi:10.1093/bioinformatics/btq057.