VAR-Seq project report template: Some Descriptive Title

Project ID: VARseq_PI_Name_Organism_Mar2015
Project PI: First Last (first.last@inst.edu)
Author of Report: First Last (first.last@inst.edu)

July 12, 2015

Contents

1 Introduction

This report describes the analysis of an VAR-Seq project from Dr. First Last's lab which studies the genetic differences of ... in *organism* .... The experimental design is as follows...

2 Sample definitions and environment settings

2.1 Environment settings and input data

Typically, the user wants to record here the sources and versions of the reference genome sequence along with the corresponding annotations. In the provided sample data set all data inputs are stored in a `data` subdirectory and all results will be written to a separate `results` directory, while the `systemPipeVARseq.Rnw` 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.

To run this sample report, mini sample FASTQ and reference genome files can be downloaded from here. The chosen data set SRP010938 contains 18 paired-end (PE) read sets from *Arabidposis thaliana* Howard et al. (2013). This data set comes from a different NGS application area, but it is sufficient to demonstrate the analysis steps of this workflow. To minimize processing time during testing, each FASTQ file has been subsetted to 90,000-100,000 randomly sampled PE reads that map to the first 100,000 nucleotides of each chromosome of the *A. thalina* genome. The corresponding reference genome sequence (FASTA) and its GFF annotion files (provided in the same download) have been truncated accordingly. This way the entire test sample data set is less than 200MB in storage space. A PE read set has been chosen for this test data set for flexibility, because it can be used for testing both types of analysis routines requiring either SE (single end) reads or PE reads.

2.2 Required packages and resources

The *systemPipeR* package needs to be loaded to perform the analysis steps shown in this report (Girke, 2014).

```
library(systemPipeR)
```

If applicable load custom functions not provided by *systemPipeR*

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

2.3 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 = "#")[,1:5]
targets
```

```
                    FileName1                 FileName2 SampleName Factor SampleLong
1  ./data/SRR446027_1.fastq ./data/SRR446027_2.fastq        M1A     M1  Mock.1h.A
2  ./data/SRR446028_1.fastq ./data/SRR446028_2.fastq        M1B     M1  Mock.1h.B
3  ./data/SRR446029_1.fastq ./data/SRR446029_2.fastq        A1A     A1   Avr.1h.A
4  ./data/SRR446030_1.fastq ./data/SRR446030_2.fastq        A1B     A1   Avr.1h.B
5  ./data/SRR446031_1.fastq ./data/SRR446031_2.fastq        V1A     V1   Vir.1h.A
6  ./data/SRR446032_1.fastq ./data/SRR446032_2.fastq        V1B     V1   Vir.1h.B
7  ./data/SRR446033_1.fastq ./data/SRR446033_2.fastq        M6A     M6  Mock.6h.A
8  ./data/SRR446034_1.fastq ./data/SRR446034_2.fastq        M6B     M6  Mock.6h.B
9  ./data/SRR446035_1.fastq ./data/SRR446035_2.fastq        A6A     A6   Avr.6h.A
10 ./data/SRR446036_1.fastq ./data/SRR446036_2.fastq        A6B     A6   Avr.6h.B
11 ./data/SRR446037_1.fastq ./data/SRR446037_2.fastq        V6A     V6   Vir.6h.A
12 ./data/SRR446038_1.fastq ./data/SRR446038_2.fastq        V6B     V6   Vir.6h.B
13 ./data/SRR446039_1.fastq ./data/SRR446039_2.fastq       M12A    M12 Mock.12h.A
14 ./data/SRR446040_1.fastq ./data/SRR446040_2.fastq       M12B    M12 Mock.12h.B
15 ./data/SRR446041_1.fastq ./data/SRR446041_2.fastq       A12A    A12  Avr.12h.A
16 ./data/SRR446042_1.fastq ./data/SRR446042_2.fastq       A12B    A12  Avr.12h.B
17 ./data/SRR446043_1.fastq ./data/SRR446043_2.fastq       V12A    V12  Vir.12h.A
18 ./data/SRR446044_1.fastq ./data/SRR446044_2.fastq       V12B    V12  Vir.12h.B
```

3 Read preprocessing

3.1 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="bwa.param", mytargets="targets.txt")
fqlist <- seeFastq(fastq=infile1(args), batchsize=100000, klength=8)
pdf("./results/fastqReport.pdf", height=18, width=4*length(fqlist))
seeFastqPlot(fqlist)
dev.off()
```

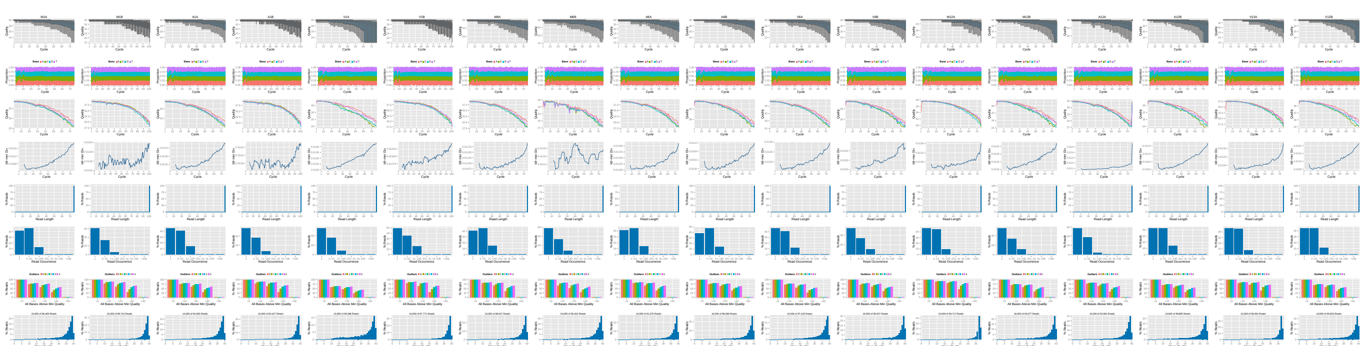

Figure 1: QC report for 18 FASTQ files.

4 Alignments

4.1 Read mapping with BWA

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

```
args <- systemArgs(sysma="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)

```
bampaths <- runCommandline(args=args)
```

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="20:00:00", nodes=paste0("1:ppn=", cores(args)), memory="10gb")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="torque.tmpl", Njobs=18, runid="01",
                  resourceList=resources)
waitForJobs(reg)
```

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. The following code shows how to run this aligner on multiple nodes of a comute cluster that uses Torque as scheduler.

```
library(gmapR); library(BiocParallel); library(BatchJobs)
gmapGenome <- GmapGenome(reference(args), directory="data", name="gmap_tair10chr", create=TRUE)
args <- systemArgs(sysma="gsnap.param", mytargets="targetsPE.txt")
f <- function(x) {
    library(gmapR); library(systemPipeR)
    args <- systemArgs(sysma="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])
}
funs <- makeClusterFunctionsTorque("torque.tmpl")
param <- BatchJobsParam(length(args), resources=list(walltime="20:00:00", nodes="1:ppn=1", memory="6gb"),
register(param)
d <- bplapply(seq(along=args), f)
writeTargetsout(x=args, file="targets_gsnap_bam.txt")
```

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/", "somedir/"),
            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; 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 subdirectory `./inst/extdata/` of the source file of the *systemPipeR* package. Alternatively, they can be downloaded directly from here.

```
writeTargetsout(x=args, file="targets_bam.txt")
system("java -jar CreateSequenceDictionary.jar R=./data/tair10.fasta O=./data/tair10.dict")
# system("java -jar /opt/picard/1.81/CreateSequenceDictionary.jar R=./data/tair10.fasta O=./data/tair10.di
args <- systemArgs(sysma="gatk.param", mytargets="targets_bam.txt")
resources <- list(walltime="20:00:00", nodes=paste0("1:ppn=", 1), memory="10gb")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="torque.tmpl", Njobs=18, runid="01",
                  resourceList=resources)
waitForJobs(reg)
writeTargetsout(x=args, file="targets_gatk.txt")
```

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="sambcf.param", mytargets="targets_bam.txt")
resources <- list(walltime="20:00:00", nodes=paste0("1:ppn=", 1), memory="10gb")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="torque.tmpl", Njobs=18, runid="01",
                  resourceList=resources)
waitForJobs(reg)
writeTargetsout(x=args, file="targets_sambcf.txt")
```

5.3 Variant calling with VariantTools

```
library(gmapR); library(BiocParallel); library(BatchJobs)
args <- systemArgs(sysma="vartools.param", mytargets="targets_gsnap_bam.txt")
f <- function(x) {
    library(VariantTools); library(gmapR); library(systemPipeR)
    args <- systemArgs(sysma="vartools.param", mytargets="targets_gsnap_bam.txt")
    gmapGenome <- GmapGenome(systemPipeR::reference(args), directory="data", name="gmap_tair10chr", create
    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)
}
funs <- makeClusterFunctionsTorque("torque.tmpl")
param <- BatchJobsParam(length(args), resources=list(walltime="20:00:00", nodes="1:ppn=1", memory="6gb"),
register(param)
d <- bplapply(seq(along=args), f)
writeTargetsout(x=args, file="targets_vartools.txt")
```

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)
args <- systemArgs(sysma="filter_gatk.param", mytargets="targets_gatk.txt")
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))
filterVars(args, filter, varcaller="gatk", organism="A. thaliana")
writeTargetsout(x=args, file="targets_gatk_filtered.txt")
```

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="filter_sambcf.param", mytargets="targets_sambcf.txt")
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)"
```

```
filterVars(args, filter, varcaller="bcftools", organism="A. thaliana")
writeTargetsout(x=args, file="targets_sambcf_filtered.txt")
```

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.

```
args <- systemArgs(sysma="filter_vartools.param", mytargets="targets_vartools.txt")
filter <- "(values(vr)$n.read.pos.ref + values(vr)$n.read.pos) >= 2 & (values(vr)$n.read.pos / (values(vr)
# filter <- "(values(vr)£n.read.pos.ref + values(vr)£n.read.pos) >= 20 & (values(vr)£n.read.pos / (values(
filterVars(args, filter, varcaller="vartools", organism="A. thaliana")
writeTargetsout(x=args, file="targets_vartools_filtered.txt")
```

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 Annotate filtered variants called by GATK

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

7.2 Annotate filtered variants called by BCFtools

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

7.3 Annotate filtered variants called by VariantTools

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

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="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="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="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")
```

9 Summary statistics of variants

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

9.1 Summary for GATK

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

9.2 Summary for BCFtools

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

9.3 Summary for VariantTools

```
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_vartools_filtered.txt")
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="annotate_vars.param", mytargets="targets_gatk_filtered.txt")
varlist <- sapply(names(outpaths(args))[1:4], function(x) as.character(read.delim(outpaths(args)[x])$VARID
vennset_gatk <- overLapper(varlist, type="vennsets")
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_sambcf_filtered.txt")
varlist <- sapply(names(outpaths(args))[1:4], function(x) as.character(read.delim(outpaths(args)[x])$VARID
vennset_bcf <- overLapper(varlist, type="vennsets")
pdf("./results/vennplot_var.pdf")
vennPlot(list(vennset_gatk, vennset_bcf), mymain="", mysub="GATK: red; BCFtools: blue", colmode=2, ccol=c(
dev.off()
```

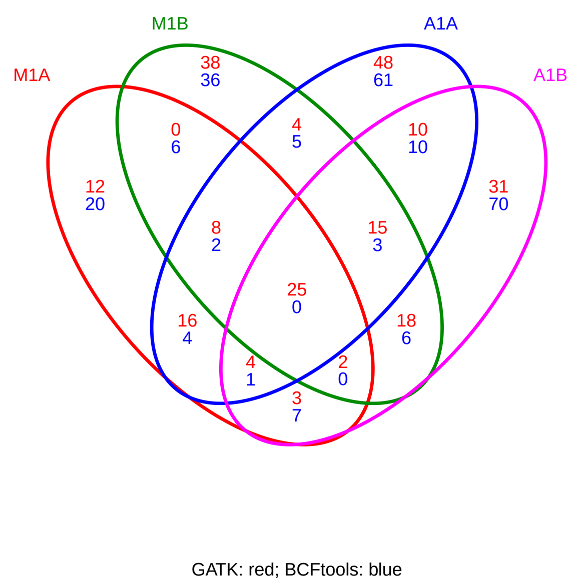

Figure 2: Venn Diagram for 4 samples from GATK and BCFtools.

11 Version Information

```
toLatex(sessionInfo())
```

- R version 3.2.0 (2015-04-16), `x86_64-unknown-linux-gnu`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: BiocGenerics 0.14.0, BiocParallel 1.2.7, Biostrings 2.36.1, DBI 0.3.1, GenomeInfoDb 1.4.1, GenomicAlignments 1.4.1, GenomicRanges 1.20.5, IRanges 2.2.5, RSQLite 1.0.0, Rsamtools 1.20.4, S4Vectors 0.6.1, ShortRead 1.26.0, XVector 0.8.0, knitr 1.10.5, systemPipeR 1.2.9

- Loaded via a namespace (and not attached): AnnotationDbi 1.30.1, AnnotationForge 1.10.1, BBmisc 1.9, BatchJobs 1.6, Biobase 2.28.0, BiocStyle 1.6.0, Category 2.34.2, GO.db 3.1.2, GOstats 2.34.0, GSEABase 1.30.2, MASS 7.3-42, Matrix 1.2-1, RBGL 1.44.0, RColorBrewer 1.1-2, Rcpp 0.11.6, XML 3.98-1.3, annotate 1.46.0, base64enc 0.1-2, bitops 1.0-6, brew 1.0-6, checkmate 1.6.0, colorspace 1.2-6, digest 0.6.8, edgeR 3.10.2, evaluate 0.7, fail 1.2, formatR 1.2, futile.logger 1.4.1, futile.options 1.0.0, genefilter 1.50.0, ggplot2 1.0.1, graph 1.46.0, grid 3.2.0, gtable 0.1.2, highr 0.5, hwriter 1.3.2, lambda.r 1.1.7, lattice 0.20-31, latticeExtra 0.6-26, limma 3.24.12, magrittr 1.5, munsell 0.4.2, pheatmap 1.0.7, plyr 1.8.3, proto 0.3-10, reshape2 1.4.1, rjson 0.2.15, scales 0.2.5, sendmailR 1.2-1, splines 3.2.0, stringi 0.5-5, stringr 1.0.0, survival 2.38-3, tools 3.2.0, xtable 1.7-4, zlibbioc 1.14.0

12 Funding

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

13 References

Thomas Girke. systemPipeR: NGS workflow and report generation environment, 28 June 2014. URL https://github.com/tgirke/systemPipeR.

Brian E Howard, Qiwen Hu, Ahmet Can Babaoglu, Manan Chandra, Monica Borghi, Xiaoping Tan, Luyan He, Heike Winter-Sederoff, Walter Gassmann, Paola Veronese, and Steffen Heber. High-throughput RNA sequencing of pseudomonas-infected arabidopsis reveals hidden transcriptome complexity and novel splice variants. *PLoS One*, 8 (10):e74183, 1 October 2013. ISSN 1932-6203. doi: 10.1371/journal.pone.0074183. URL http://dx.doi.org/10.1371/journal.pone.0074183.

H Li and R Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14): 1754–1760, July 2009. ISSN 1367-4803. doi: 10.1093/bioinformatics/btp324. URL http://dx.doi.org/10.1093/bioinformatics/btp324.

Heng Li. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics*, 27(21):2987–2993, 1 November 2011. ISSN 1367-4803. doi: 10.1093/bioinformatics/btr509. URL http://bioinformatics.oxfordjournals.org/content/27/21/2987.abstract.

Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. 03 2013. URL http://arxiv.org/abs/1303.3997.

Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, and Mark A DePristo. The genome analysis toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res.*, 20(9):1297–1303, 19 July 2010. ISSN 1088-9051. doi: 10.1101/gr.107524.110. URL http://dx.doi.org/10.1101/gr.107524.110.