Code accompanying *CrispRVariants: precisely charting the mutation spectrum in genome engineering experiments*

Helen Lindsay

11th February 2016

Contents

Notes about code

In this guide, I use double colon notation to indicate which package a function belongs to, e.g. `rtracklayer::import`, when I think it may be of interest. This notation is not necessary for running the code. Note that some packages import other packages, so the parent package may not be one of the imported packages.

The analyses for the Supplementary Notes are not completely independent of each other. The sections of this guide are therefore ordered in a logical way for running the code, not as they appear in the Supplementary material.

Most of the code is R code. Occasional segments are Bash code. These are indicated by a comment "(BASH)" at the start of the section.

Software requirements

This code requires the command line programs *bwa* (version 0.7.12-r1039), *samtools* (version 1.1-33-g311926c), *pear* (version 0.9.4-64), *seqprep*, *blat* (version 35x1), *CRISPResso* and *ART* read simulator (version ChocolateCherryCake) be installed. Paths may need to be altered to reflect installation location. Scripts `samGetSEQfast.pl` from *rackJ* and *psl2sam.pl* from *samtools* also required for converting *blat* to *bam* output. We assume these are stored in the *src* directory.

The python scripts for primer splitting require the python libraries *argparse*, *regex*, *gzip*, *os* and *sys*.

The following R packages are used: *CrispRVariants*, *BiocParallel*, *Biostrings*, *BSgenome.Drerio.UCSC.danRer7*, *BSgenome.Hsapiens.UCSC.hg19*, *gdata*, *GenomicAlignments*, *GenomicFeatures*, *GenomicRanges*,*ggplot2*, *grid*, *gridExtra reshape2*, *Rsamtools*, *rtracklayer*, *seqinr*, *scales*, *ShortRead*.

For the analyses with CRISPR-GA, we also use the packages *httr* and *rvest*.

Versions are listed in the `sessionInfo()` at the end of this document.

Directory structure

The working directory for this analysis has the following structure:

```
|-- ampliconDIVider-master
|-- annotation
|-- bam
|-- Burger_MiSeq_data
|-- Burger_Sanger_data
|-- Cho_data
|-- fastq
|-- idx
|-- merged_split
|-- README.md
|-- results
|-- simulation
    |--amplicondivider
    |--crispresso
    |--merged
|-- split_merged
|-- src
```

Scripts are in *src* and bwa genome and faidx indices for danRer7 are in *idx*. External information such as guide locations and sample names is in *annotation*. *fastq* contains the original and renamed fastq files for the Shah *et al* data. *merged_split* and *split_merged* contain corresponding reads that have been merged and separated by forward primer sequence and *bam* contains corresponding aligned reads.

Data from Burger *et al* and Cho *et al* is stored in the *Burger_MiSeq_data*, *Burger_Sanger_data* and *Cho_data* folders.

The *ampliconDIVider-master* folder is a clone of the AmpliconDIVider repository, plus the additional *ampliconDIV_minimal.sh* script we use to run the counting functions.

The results of running other tools are in the results folder. Code for analyses with CrispRVariants is in this pdf.

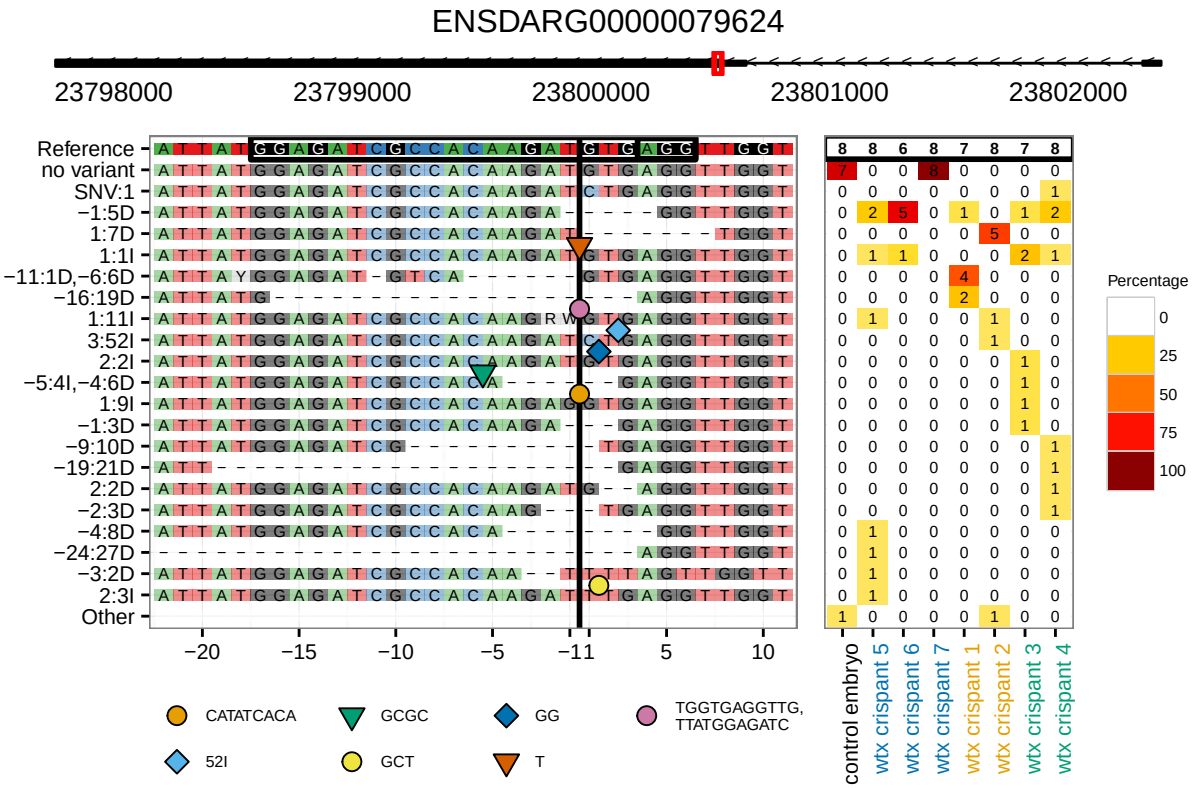
This code depends on this directory structure and will fail if the data have not been downloaded and unpacked from the correct repositories. All code is run from the *src* directory.

Analysis of *wtx* in main text

For this analysis we use a pre-generated transcript database (txdb) corresponding to the danRer7 genes. This was generated from the Ensembl gtf file. For information about generating a transcript database see the GenomicFeatures Bioconductor package.

```
library("CrispRVariants")
library("gdata")
library("BSgenome.Drerio.UCSC.danRer7")
library("GenomicFeatures")
library("rtracklayer")

# Load the genome and the transcript database
danRer7 <- BSgenome.Drerio.UCSC.danRer7
txdb <- loadDb("~/zebrafish_txdb.sqlite")

# Import the guide sequence location
gd <- rtracklayer::import("../annotation/Burger_Sanger_guides.bed")
gd <- gd[gd$name == "wtx_ccA"]
# Add extra bases to both sides of the guide for counting variants
gdl <- gd+5

md_fname <- "../annotation/Burger_wtx_metadata.xls"
md <- read.xls(md_fname)
guide_n <- "wtx_ccA"
bdir <- "../Burger_Sanger_data/wtx_ccA/bam"
bams <- paste0(gsub("[\ |\\/]", "_", md$directory), "_s.bam")
bam_fnames <- file.path(bdir, bams)
ref <- getSeq(danRer7, gdl)

# After manual inspection, we considered the alignment of "AB1060"
# to be poor and so remove this sequence.
cset <- readsToTarget(bam_fnames, gdl, reference = ref, target.loc = 22,
                      names = as.character(md$Short.name),
                      exclude.names = "AB1060")
group <- md[md$Short.name %in% names(cset$crispr_runs),"Group"]
```

```
plotVariants(cset,txdb=txdb, plotAlignments.args = list(legend.cols = 4),
             plotFreqHeatmap.args = list(group = group),
  left.plot.margin = grid::unit(c(0.5,0.5,1,0.5), "lines"))
```




```
## TableGrob (2 x 1) "arrange": 2 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange  gtable[layout]
## 2 2 (2-2,1-1) arrange gtable[arrange]
```

Supplementary Analyses

Preprocessing Shah data

The raw data from Shah *et al* was downloaded from DDBJ and extracted using `fastq-dump --split-files`.

We map with **BWA-MEM** (http://bio-bwa.sourceforge.net/) to genome version danRer7. We previously generated indices for this genome using *bwa index* and stored them in the *idx* directory.

Determine guide location by mapping to the danRer7 genome

The guides and primer sequences were extracted from the Supplementary metadata table and mapped to danRer7. When a sequence mapped equally well to multiple locations, the correct location was chosen by manual inspection. We confirmed the correct primer sequences with Shah *et al*. In the original supplementary table file, two primers were swapped. These are corrected in the version used here.

```
library(seqinr)
library(gdata)

# Write the guides to FASTA
shah_results <- read.xls("../annotation/Shah_metadata.xls")
guide_nms <- gsub("\ ", "", shah_results$Gene)
guide_seqs <- gsub("\ ", "", shah_results$sgRNA)
write.fasta(as.list(guide_seqs), guide_nms, file = "guides.fa")

# Map the guides with BWA allowing at most 2 mismatches and disallowing gaps
bwa_idx <- "danRer7.fa"
bwa_cmd <- paste0(c("bwa aln -n 2 -o 0 %s %s | bwa samse %s - %s | ",
                    "samtools view -Sb - > %s; samtools sort %s %s && rm %s"),
                  collapse = "")
system(sprintf(bwa_cmd, bwa_idx, "guides.fa", bwa_idx, "guides.fa",
       "guides.bam", "guides.bam", "guides_s", "guides.bam"))

system(sprintf(bwa_cmd, bwa_idx, "primers.fa", bwa_idx, "primers.fa",
        "primers.bam", "primers.bam", "primers_s", "primers.bam"))
```

The bed files listing the primer and guide locations were manually curated from the sequences generated here.

Mapping paired reads with BWA MEM

This section is run in BASH.

```
#(BASH)
# "index" is the prefix of our set of bwa indices for danRer7
# We used bwa version 0.7.5a-r405
index=../idx/danRer7.fa
fq1=../fastq/SRR1769728_1.fastq.gz
fq2=../fastq/SRR1769728_2.fastq.gz
bwa mem -t 4 $index $fq1 $fq2 | samtools view -Sb - > ../bam/SRR1769728.bam
samtools sort ../bam/SRR1769728.bam ../bam/original && rm ../bam/SRR1769728.bam
samtools index ../bam/original.bam
```

Merging reads with PEAR and SEQ PREP

The read names are in the format "readName length=251". The read lengths were first stripped from the read names, as these are not always identical between the pairs and cause SeqPrep to crash.

This section is run in BASH.

```
#(BASH)

fq1r=../fastq/SRR1769728_1_renamed.fastq
fq2r=../fastq/SRR1769728_2_renamed.fastq
zcat $fq1 | awk '{if (NR % 2 == 1) print $1, $2; else print $1}' > $fq1r; gzip $fq1r
zcat  $fq2 | awk '{if (NR % 2 == 1) print $1, $2; else print $1}' > $fq2r; gzip $fq2r
```

Then paired end reads were merged with SeqPrep. The parameter "-n 1" specifies that reads must have identical length (100% overlap), "-L 55" specifies that the merged reads must be at least 55 bases long. The

longest combined forward and reverse primer length is 52 bases. Reads at least 55 bases long should therefore not consist of primer dimers.

Reads were also merged with PEAR, with default settings.
Running this section requires SeqPrep and PEAR to be installed.

```
#(BASH)

# 1. Merge with SeqPrep as in paper
out1=../fastq/SRR1769728_1_sp.fastq.gz
out2=../fastq/SRR1769728_2_sp.fastq.gz
merged_sp_fq=../fastq/SRR1769728_merged_seqprep.fastq.gz
~/SeqPrep -f $fq1r -r $fq2r -1 $out1 -2 $out2 -g -n 1.0 -s ${merged_sp_fq}

# 2. Merge with SeqPrep filtering only on length (default -n is 0.9)
out3=../fastq/SRR1769728_1_sp_l55.fastq.gz
out4=../fastq/SRR1769728_2_sp_l55.fastq.gz
merged_sp_l55_fq=../fastq/SRR1769728_merged_seqprep_l55.fastq.gz
~/SeqPrep -f $fq1r -r $fq2r -1 $out3 -2 $out4 -L 55 -s ${merged_sp_l55_fq}

# Cleanup files that are no longer needed
rm $fq1r $fq2r $out1 $out2 $out3 $out4

# Also merge with PEAR
# Start with unzipped fastq (BZIP2 also works)
gunzip ${fq1} ${fq2}
fq1=../fastq/SRR1769728_1.fastq
fq2=../fastq/SRR1769728_2.fastq

merged_pear_fq=../fastq/SRR1769728_merged_pear
pear-0.9.4-64 -j 12 -f $fq1 -r $fq2 -o ${merged_pear_fq}
o1=SRR1769728_merged_pear.unassembled.forward.fastq
o2=SRR1769728_merged_pear.unassembled.reverse.fastq
o3=SRR1769728_merged_pear.discarded.fastq
rm ${o1} ${o2} ${o3}
gzip ${fq1} ${fq2} ${merged_pear_fq}.assembled.fastq
merged_pear_fq=../fastq/SRR1769728_merged_pear.assembled.fastq.gz
```

Mapping of the remaining fastq reads was performed with identical parameters. Details are in `run_mapping.R`

Splitting reads by PCR primer

In Shah *et al*, paired end reads were merged using **Seq-Prep** then split into amplicon sequences using **NGS-Utils**. We compared two preprocessing strategies using the data from Shah *et al*: *split-then-merge* where reads were first grouped by PCR primer then merged; and *merge-then-split* where reads first merged then split by primer. As **fastqutils barcode_split** in **NGS-Utils** accepts a single, i.e. merged, fastq file, we wrote a python script *split_by_primers.py* to allow splitting paired-end reads by primer sequence either before or after merging. As in Shah *et al*, to assign reads to an amplicon, we required a match with at most one error to the forward primer only and allowed matches to either the forward primer or its reverse complement

***merge-then-split*: Separate merged reads by PCR primer**

```
#(in BASH)

fwd_primers=../annotation/shah_fwd_primers.txt

for merged in $merged_sp_fq $merged_sp_155_fq $merged_pear_fq
do
  python split_by_primers.py -r1 ${merged} -o ../merged_split -p ${fwd_primers}
done
```

***merge-then-split* Comparison of tolerance settings for matching primers**

Comparison of read splitting settings was done for the reads merged with PEAR.

```
#(In BASH)
python split_by_primers.py -c
```

split-then-merge:

Separate unmerged reads by PCR primer then merge.

```
#(in BASH)

out=../split_merged
fq1=../fastq/SRR1769728_1.fastq.gz
fq2=../fastq/SRR1769728_2.fastq.gz

python split_by_primers.py -r1 ${fq1} -r2 ${fq2} -o ${out} -p ${fwd_primers}
```

Merge the separated reads with PEAR.

```
#(in BASH)

fq1=SRR1769728_1.fastq.gz
fq2=SRR1769728_2.fastq.gz

cd ../split_merged

for f in *
do
  cd ${f}
  gunzip ${fq1} ${fq2}
  pear-0.9.4-64 -j 12 -f ${fq1%.*} -r ${fq2%.*} -o SRR1769728_merged_pear
  rm *unassembled* *discarded*
  mv SRR1769728_merged_pear.assembled.fastq SRR1769728_merged_pear.fastq
  gzip SRR1769728_merged_pear.fastq
  cd ..
done
cd ../src
```

Mapping to genome with BWA MEM

Mapping with blat to the danRer7 was done by running the script `run_blat_global.sh`

Mapping to amplicon references with BLAT

Generation of the amplicon references and local alignment with blat is detailed in the script `run_blat_local.sh`. Conversion to *bam* format is not possible with early verions of *blat*. We use *blat* version 35.

Data setup

```
library("CrispRVariants")
library("BiocParallel")
library("Biostrings")
library("BSgenome.Drerio.UCSC.danRer7")
library("gdata")
library("GenomicFeatures")
library("ggplot2")
library("grid")
library("gridExtra")
library("reshape2")
library("rtracklayer")
library("scales")

# Load the danRer7 / Zv9 genome
danRer7 <- BSgenome.Drerio.UCSC.danRer7

# Import the guide locations (inc PAM), add 5 to either end
guides <- rtracklayer::import("../annotation/shah_guides.bed")
names(guides) <- guides$name
guides <- guides + 5

# Get guide sequences from the genome
# (note: guide sequences could also be input directly)
references <- getSeq(danRer7, guides)

# Import the primer (amplicon) endpoints
primers <- rtracklayer::import("../annotation/shah_primers.bed")
names(primers) <- primers$name

# Directories
# each directory corresponds to a different preprocessing condition
bam_dir <- "../bam"
unmerged <- file.path(bam_dir, "original.bam")
merged <- file.path(bam_dir, "merged_seqprep.bam")
merged_seqprep <- file.path(bam_dir, "merged_split_seqprep")
merged_split_155 <- file.path(bam_dir, "merged_split_155_n90")
merged_split_pear <- file.path(bam_dir, "merged_split_pear")
split_merged_pear <- file.path(bam_dir, "split_merged_pear")
```

```
tolerant_pear <- file.path(bam_dir, "merged_split_pear_tolerant")
strict_pear <- file.path(bam_dir, "merged_split_pear_strict")
blat_local <- file.path(bam_dir, "blat_local")
blat_global <- file.path(bam_dir, "blat_global")
```

Functions

The following functions are used in multiple sections throughout this document.

getCrisprSets

For comparing methods, we have a set of directories each containing one bam file per guide. This function processes all files in a given directory.

```
getCrisprSets <- function(directory, references, guides,
                          bpparam = BiocParallel::MulticoreParam(8)){

  bams <- list.files(directory, pattern = "*.bam$", full.names = TRUE)

  # Get the guide names from the bam file names
  gd_nms <- gsub(".*/(.*).bam", "\\1", bams)

  # Match guide names to the guide locations.
  # Get guides and reference sequences in this order
  gd_to_bam <- match(gd_nms, names(guides))
  guides <- guides[gd_to_bam]
  references <- references[gd_to_bam]

  # Count variants for each bam by making CrisprSet objects
  result <- bplapply(seq_along(bams), function(i){
    bam <- bams[i]
    ref <- references[[i]]
    gd <- guides[i]
    nm <- names(gd)

    # Here we call SNVs in the entire guide region (upstream.snv = 17).
    # The default settings call SNVs from 8 bases upstream to 6 bases downstream.
    # This is necessary for Supplementary Note 3.
    crispr_set <- CrispRVariants::readsToTarget(bam, gd,
                            reference = ref, target.loc = 22,
                            upstream.snv = 17, names = nm)
    crispr_set

   }, BPPARAM = bpparam)

  # Name the CrisprSets according to the guides
  names(result) <- gd_nms
  result
}
```

getSplitEff

This function gets a table of mutation efficiencies for a directory containing bam files where one bam file corresponds to one guide. It is very similar to the previous function getCrisprSets, except for returning only efficiencies instead of CrisprSets.

```
getSplitEff <- function(directory, references, guides, primers,
                        bpparam = BiocParallel::MulticoreParam(8)){

  bams <- list.files(directory, pattern = "*.bam$", full.names = TRUE)
  gd_nms <- gsub(".*/(.*).bam", "\\1", bams)
  gd_to_bam <- match(gd_nms, names(guides))

  guides <- guides[gd_to_bam]
  references <- references[gd_to_bam]

  result <- bplapply(seq_along(bams), function(i){
    bam <- bams[i]
    ref <- references[[i]]
    gd <- guides[i]
    nm <- names(gd)

    # The guides coordinates are genomic.  Here we adjust the
    # guide locations for local alignments by considering where
    # the guide starts with respect to the amplicon sequence
    if (grepl("local", directory) == TRUE){
      primer <- primers[primers$name == nm]
      offset <-  1 - start(primer)
      gd <- shift(gd, offset)
    }
    crispr_set <- CrispRVariants::readsToTarget(bam, gd,
                            reference = ref, target.loc = 22)
    if (is.null(crispr_set)) return(c(NA, NA))
    mutationEfficiency(crispr_set)[c("Overall", "ReadCount")]
   }, BPPARAM = bpparam)

  result <- t(data.frame(result))
  rownames(result) <- gd_nms
  result
}
```

parseCRISPResso

This function extracts the (reads with NHEJ)/(total reads) % from the file named "Quantification_of_editing_frequency.txt" in the supplied results directory

```
parseCRISPResso <- function(results_dir){
  results_f <- file.path(results_dir, "Quantification_of_editing_frequency.txt")
  system(paste0("echo '\n' >>", results_f))

  f <- file(results_f)
  lns <- readLines(f)
  close(f)
```

```
  nhej <- lns[grep(".* NHEJ:", lns)]
  total <- lns[grep("Total", lns)]
  counts <- as.numeric(gsub(".*:([0-9]+)\ .*", "\\1", c(nhej, total)))
  result <- counts[1]/counts[2]*100
  c(result, counts[2])
}
```

getLegend

This function gets the legend from a ggplot2 object. This is useful for when two subplots should share a legend From Stack Overflow.

```
getLegend<-function(a.gplot){
  tmp <- ggplot_gtable(ggplot_build(a.gplot))
  leg <- which(sapply(tmp$grobs, function(x) x$name) == "guide-box")
  legend <- tmp$grobs[[leg]]
  return(legend)}
```

onTargetPercent

This function calculates the percentage of on-target reads for every bam file in the given directory. It works by selecting mapped, primary alignments (-F 2048 to exclude supplementary alignments plus -F 4 to exclude unmapped reads). Both members of a pair are counted. Using this function requires samtools to be installed and the bam files to be sorted and indexed.

```
onTargetPercent <- function(directory, guides){
  # Templates for counting on and off-target reads
  samtools_all <- "samtools view -F 2052 %s | wc -l"
  samtools_ontarget <- "samtools view -F 2052 %s %s:%s-%s | wc -l"

  # List all files ending in .bam
  bams <- list.files(directory, pattern = "*.bam$", full.names = TRUE)

  # Match file names to guide names, order guides accordingly
  gd_nms <- gsub(".*/(.*).bam", "\\1", bams)
  gd_to_bam <- match(gd_nms, names(guides))
  guides <- guides[gd_to_bam]

  # Run samtools commands, get results
  result <- lapply(seq_along(bams), function(i){
    gd <- guides[i]
    bm <- bams[[i]]
    total <- as.numeric(system(sprintf(samtools_all, bm), intern = TRUE))
    on <- as.numeric(system(sprintf(samtools_ontarget, bm, seqnames(gd)[1],
             start(gd), end(gd)), intern = TRUE))
    # off-target is the difference between the total and the off-target reads
    c("on" = on, "off" = total-on)

  })

  result <- do.call(rbind, result)
```

```
  rownames(result) <- gd_nms
  result
}
```

onTargetCount

This function counts on-target reads for every bam file in the given directory. It works by counting the unique names of all reads mapped to a target (amplicon) region. A read mapped as a chimera is considered on-target if any of the mapped segments overlap the target region. Using this function requires samtools to be installed and the bam files to be sorted and indexed.

```
onTargetCount <- function(directory, primers){
  #Template samtools command for selecting on-target reads
  ontgt <- "samtools view -F 4 %s %s:%s-%s | awk '{print $1}' | sort | uniq | wc -l"

  # Get bam file names
  bams <- list.files(directory, pattern = "*.bam$", full.names = TRUE)
  gd_nms <- gsub(".*/(.*).bam", "\\1", bams)

  # Match the primer (amplicon) regions to the file names
  pr_to_bam <- match(gd_nms, names(primers))
  # Order the primers accordingly
  primers <- primers[pr_to_bam]

  result <- lapply(seq_along(bams), function(i){
    pr <- primers[i]
    bm <- bams[[i]]
    on <- as.numeric(system(sprintf(ontgt, bm, seqnames(pr)[1],
              start(pr), end(pr)), intern = TRUE))
    on
  })

  result <- do.call(rbind, result)
  rownames(result) <- gd_nms
  result
}
```

dotplot

Produce a dotplot showing runs of homology of length at least `min.length` between two sequences `seq1` and `seq2` (or within one sequence if only one is supplied). Sequences must be `Biostrings::DNAString` objects. `annotations` are optional regions to highlight with gray boxes, and should be numbered with respect to seq1.

Although written to allow two different sequences in future, this function has only been tested for identical sequences.

```
dotplot <- function(seq1, seq2=seq1, min.length = 3,
                    tick.alpha = 200, annotations = IRanges(),
                    plot.title = NULL){

  if (! identical(seq1, seq2) & ! length(annotations) > 0){
    stop("Annotation only implemented for seq1 == seq2")
```

```
}

# Set of starts of each run at least min.length long
strts <- c(1:(length(seq2)- min.length + 1))

# Get all motifs that occur in seq1
motifs <- unique(as.character(Views(seq1,
             IRanges(start = strts, width = min.length))))

# Split sequences into single characters
sq1_chr <- strsplit(as.character(seq1), "")[[1]]
sq2_chr <- strsplit(as.character(seq2), "")[[1]]

# Colours for the nucleotides, to match CrispRVariants::plotVariants
colours <- c(rgb(77,175,74, tick.alpha, maxColorValue=255),
             rgb(55,126,184, tick.alpha, maxColorValue=255),
             rgb(228,26,28, tick.alpha, maxColorValue=255),
             rgb(0,0,0, tick.alpha, maxColorValue=255),
             rgb(128,128,128, tick.alpha, maxColorValue=255))

names(colours) <- c("A","C","T","G","N")

add_wdth <- c(0:(min.length-1))

# Convert a list of homology starts to a list of dot locations
to_range <- function(mm, min.length, add_wdth){
  temp <- replicate(min.length, mm)
  as.vector(t(temp) + add_wdth)
}

# Find the motifs from seq1 in seq2
sq2_matches <- lapply(motifs, function(x){
  to_range(start(Biostrings::matchPattern(x, seq2)),
           min.length, add_wdth)
})
sq2_lengths <- elementLengths(sq2_matches)/min.length

# Check for multiple occurrences of the same motif
sq1_matches <- lapply(seq_along(motifs), function(i){
  x <- motifs[i]
  mtch <- Biostrings::matchPattern(x, seq1)
  mtch_ln <- length(mtch)
  mtch <- rep(start(mtch), each = sq2_lengths[i])
  rng <- to_range(mtch, min.length, add_wdth)
  data.frame(x = rng, y = sq2_matches[[i]])
})

# Make coordinates for annotation boxes
# Add 1 to ends to cover from start to end of the boxes
bounds <- rep(Inf, length(annotations))
annot_box <- data.frame(xmin = c(start(annotations), -1 * bounds),
                        xmax = c(end(annotations)+1, bounds),
                        ymin = c(-1 * bounds, start(annotations)),
```

```
                          ymax = c(bounds, end(annotations))+1)

  dat <- unique(do.call(rbind, sq1_matches))

  # Plotting
  p <- ggplot(annot_box, aes(xmin = xmin, xmax=xmax,ymin=ymin,ymax=ymax)) +
        # First make the annotation as this should be in the background
        geom_rect(alpha = 0.5, fill= "gray") +

    # Add the points, where each point indicates a homologous run
    geom_point(data = dat, aes(x=x, y=y, xmin=NULL,xmax=NULL,ymin=NULL,ymax=NULL),
               size = 0.75) +

    # Add lines for the plot borders,
    # otherwise this gets lost when the nucleotides are shown
    geom_hline(yintercept=c(0), size = 1) + geom_vline(xintercept=0, size = 1) +
    theme_bw() + xlab(NULL) + ylab(NULL) +

    # Set one axis tick line per nucleotide
    scale_x_continuous(breaks = seq_along(sq1_chr)) +
    scale_y_continuous(breaks = seq_along(sq2_chr)) +

    # Colour the axis ticks according to the nucleotide
    theme(axis.ticks.x = element_line(colour = colours[sq1_chr], size = 1),
        axis.ticks.y = element_line(colour = colours[sq2_chr], size = 1),
        axis.text.x = element_blank(),
        axis.text.y = element_blank(),
        axis.ticks.length = unit(0.3, "cm"),
        panel.grid.major = element_blank()) +

    # Zoom the plot to just the seq, without changing data
    coord_cartesian(xlim = c(-0.5, length(sq1_chr) + 1),
                    ylim = c(-0.5, length(sq2_chr) + 1), expand = FALSE)

  if (! is.null(plot.title)){
    p <- p + annotate("text", label = plot.title,
                      x = 5, y = Inf, hjust = 0, vjust = 2)
  }
  return(p)
}
```

plot_variants

This is similar to CrispRVariants::plotVariants, but does not include the transcript plot and adds a title.

```
plot_variants <- function(cset, nm){
  gene_p <- grid::grid.rect(gp=grid::gpar(col="white"), draw = FALSE)
  alns <-  CrispRVariants::plotAlignments(cset, min.freq = 1)
  alns <- alns + ggtitle(nm)
  heat <- CrispRVariants::plotFreqHeatmap(cset, min.freq = 1,
                                  x.angle = 0, x.hjust = 0.5,
              legend.key.height = grid::unit(1, "lines"))
```

```
  return(CrispRVariants:::arrangePlots(gene_p, alns, heat,
              left.plot.margin = grid::unit(c(0.1,0,0.2,0.2), "lines")))
}
```

getReferenceGaps

getReferenceGaps converts pairwise alignments into alignments with respect to a reference, and returns a list of data that can be input to CrispRVariants::plotVariants.matrix

This function takes paired.alns, a Biostrings::PairwiseAlignments object of a set of reads to a reference sequence, and optionally reverse.alns, a matching PairwiseAlignments object of the reverse complement reads.

```
getReferenceGaps <- function(paired.alns, reverse.alns = NULL){
  refs <- as.character(pattern(paired.alns))
  qrys <- DNAStringSet(as.character(subject(paired.alns)))

  # Use the reverse complement alignment when it is better
  if (! is.null(reverse.alns)){
    rc_better <- score(reverse.alns) > score(paired.alns)
    refs[rc_better] <- as.character(pattern(reverse.alns))[rc_better]
    tmp <- DNAStringSet(as.character(subject(reverse.alns)[rc_better]))
    qrys[rc_better] <- tmp
  }

  # Split into nucleotides
  ref_chrs <- strsplit(refs, "")

  # Get locations of gaps in reference sequence
  # make these into a list of insertions in the reads
  irl <- lapply(seq_along(ref_chrs), function(i){
     x <- ref_chrs[[i]]
     rls <- rle(x == "-")
     ir <- as(PartitioningByEnd(cumsum(rls$lengths)), "IRanges")
     gap_rngs <- ir[rls$values]
     gap_seqs <- Views(qrys[[i]], ir[rls$values])
     ungap_qry <- paste0(Views(qrys[[i]], ir[!rls$values]), collapse = "")
     list(insertions = gap_rngs, ungapped = ungap_qry,
          gap_seqs = as.character(gap_seqs))
  })
  qrys <- DNAStringSet(unlist(lapply(irl, "[[", "ungapped")))
  ins <- lapply(irl, "[[", "insertions")

  # Offset the starts for the previous insertions
  temp <- IRangesList(ins)
  tt <- unlist(cumsum(width(temp))) # count inserted sequences
  tt <- c(0,tt[1:(length(tt)-1)]) # offset by one to consider only previous ins
  ee <- unlist(elementLengths(ins))
  ee <- cumsum(ee[ee>0])
  tt[ee[1:(length(ee)-1)] + 1] <- 0 # Zeros at the start of new reads
  starts <- unlist(start(temp)) - tt

  # Make a table of insertions
  insertions_t <- data.frame(start = starts,
```

```
                    seq = unlist(lapply(irl, "[[", "gap_seqs")),
                    id = rep(seq_along(qrys), elementLengths(ins)))

  list(insertions = insertions_t, seqs = qrys)
}
```

Parse mutation efficiency estimates

This section shows how efficiency estimates for AmpliconDIVider, CRISPResso and CRISPR-GA were parsed. The efficiency estimates were generated by running the scripts `run_amplicondivider.sh`, `run_crispresso.R` and `run_crisprga.R`.

AmpliconDIVider

```
amplicon_div <- "../results/ampliconDIV_filtered/merged_split_pear"
adiv_files <- list.files(amplicon_div, full.names = TRUE)

adiv_results <- sapply(adiv_files, function(fn){
  tt <- read.table(fn, sep = "\t")[1,c(6,2)]
})
colnames(adiv_results) <- gsub(".*pear/(.*$)", "\\1", colnames(adiv_results))
adiv_counts <- unlist(adiv_results[2,])
adiv_results <- unlist(adiv_results[1,] ) * 100 # Report as a %
```

CRISPResso

We generated an amplicon-specific reference by considering differences from the standard *danRer7* reference in the reads mapped by **bwa**. The custom reference sequences are located in "../annotation/shah_custom_amplicons.txt".

```
# Get CRISPResso efficiency with amplicon-specific reference
crispresso <- "../results/CRISPResso"
crispresso_results <- sapply(list.files(crispresso, full.names = TRUE),
                             parseCRISPResso)
colnames(crispresso_results) <- basename(colnames(crispresso_results))
crispresso_counts <- crispresso_results[2,]
crispresso_results <- crispresso_results[1,]

# Get CRISPResso-pooled efficiency
crispresso_pooled <- "../results/CRISPResso_pooled_mixed"
pooled_results <- sapply(list.files(crispresso_pooled, pattern = "CRISPResso_on",
                         full.names = TRUE), parseCRISPResso)
colnames(pooled_results) <- gsub(".*_on_", "", colnames(pooled_results))
crispresso_pooled_counts <- pooled_results[2,]
crispresso_pooled_results <- pooled_results[1,]

# Get CRISPResso efficiency with standard reference
std <- "../results/CRISPResso_std_ref"
std_results <- sapply(list.files(std, full.names = TRUE), parseCRISPResso)
```

```
colnames(std_results) <- basename(colnames(std_results))
crispresso_std_counts <- std_results[2,]
crispresso_std_results <- std_results[1,]

# Get CRISPResso efficiency with standard reference, merged reads
mrged <- "../results/CRISPResso_std_ref_merged"
mrged_results <- sapply(list.files(mrged, full.names = TRUE), parseCRISPResso)
colnames(mrged_results) <- basename(colnames(mrged_results))
crispresso_mrged_counts <- mrged_results[2,]
crispresso_mrged_results <- mrged_results[1,]


nms <- unique(c(names(crispresso_pooled_results),
                names(crispresso_results),
                names(crispresso_std_results),
                names(crispresso_mrged_results)))
crispresso_dat <- data.frame(crispresso_results[nms],
                  crispresso_std_results[nms],
                  crispresso_mrged_results[nms])

rownames(crispresso_dat) <- nms
crispresso_dat <- cbind(crispresso_dat, crispresso_pooled_results[nms])
colnames(crispresso_dat) <- c("Custom ref", "Std ref", "Merged", "Pooled")
```

CRISPR-GA

```
cga <- "../results/CRISPRGA/fwd_only_pear_crisprga_refonly.txt"
temp <- read.table(cga, sep = "\t", header = TRUE)
crisprga <- temp$crisprga
names(crisprga) <- temp$name

cga_w_ref <- "../results/CRISPRGA/crisprga_w_guide_fwd_only_pear.txt"
temp <- read.table(cga_w_ref, sep = "\t", header = TRUE)
crisprga_w_ref <- temp$w_guide
names(crisprga_w_ref) <- temp$name
```

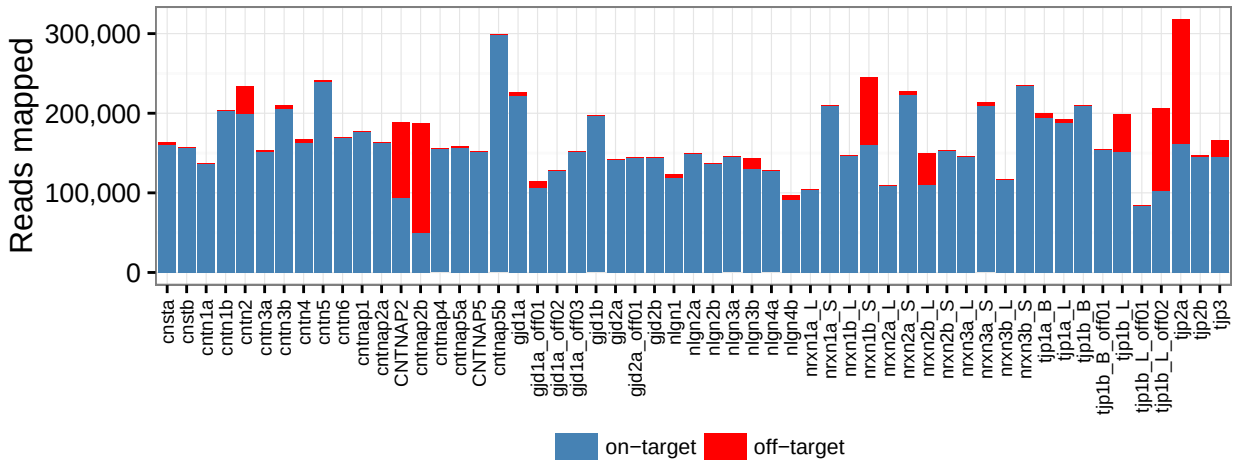
Supplementary Note 5: Local alignment is affected by PCR off-target reads and can inflate efficiency estimates

Plot reads aligned on- and off-target

This plot counts the number of reads for each guide mapped by **bwa** that do and do not overlap the guide regions. Pooled reads were separated by guide by matching the PCR primer sequences.

```
result <- onTargetPercent(merged_split_pear, guides)
result <- reshape2::melt(result)

ggplot(result, aes(x = Var1, y=value, group=Var2, fill=Var2)) +
# Stack counts for on- and off-target reads, don't transform to percentage
```

```
geom_bar(position = "stack", stat= "identity") + theme_bw() + xlab(NULL) +
 ylab("Reads mapped") +
theme(axis.text.x = element_text(angle = 90, size = 7, hjust = 1, vjust = 0.5),
      legend.position = "bottom", legend.text = element_text(size = 8),
      legend.key.height = grid::unit(0.4, "cm"),
      legend.key = element_blank(),
      legend.margin = grid::unit(0.05, "cm"),
      plot.margin = grid::unit(c(0.1,0.5,0.1,0,1), "lines")) +
scale_y_continuous(labels = comma) +
scale_fill_manual("", values = c("steel blue", "red"),
                  labels = c("on-target", "off-target"))
```



Comparison of local and global alignment strategies

```
# Calculate CrispRVariants efficiencies
directories <- c(merged_split_pear, blat_global, blat_local)
dir_nms <- c("BWA mem", "Blat global", "Blat local")

effs <- lapply(directories, getSplitEff,
               references = references, guides = guides,
               primers = primers)
effs <- do.call(cbind, effs)
effs <- data.frame(effs)
colnames(effs) <- paste(rep(dir_nms, each=2), c("efficiency", "reads"))
crv_counts <- effs[, grep("reads", colnames(effs)), drop = FALSE]
effs <- effs[, grep("efficiency", colnames(effs)), drop = FALSE]

# Merge results
effs[,"CRISPR-GA efficiency"] <- crisprga[rownames(effs)]
effs[,"CRISPR-GA efficiency (inc guide)"] <- crisprga_w_ref[rownames(effs)]
effs[,"AmpliconDIVider efficiency"] <- adiv_results[rownames(effs)]
effs[,"CRISPResso efficiency"] <- crispresso_results[rownames(effs)]

print(head(effs))
```

```
##        BWA mem efficiency Blat global efficiency Blat local efficiency
```

```
## cnsta               72.63                 71.38                 71.41
## cnstb               67.60                 67.47                 67.49
## cntn1a               5.65                  7.63                  5.64
## cntn1b              13.12                 13.02                 13.65
## cntn2               48.27                 48.14                 48.85
## cntn3a               1.70                  4.76                  2.08
##        CRISPR-GA efficiency CRISPR-GA efficiency (inc guide)
## cnsta             73.102622                        58.228176
## cnstb             65.056208                        57.318387
## cntn1a             6.437424                         5.095772
## cntn1b            12.898986                         7.194730
## cntn2             55.288287                        38.357939
## cntn3a             1.959631                         1.028416
##        AmpliconDIVider efficiency CRISPResso efficiency
## cnsta                   71.51620             48.434551
## cnstb                   66.61850             47.438024
## cntn1a                   5.54596              2.472495
## cntn1b                  12.99970              7.601280
## cntn2                   47.55570             33.714853
## cntn3a                   1.66324              1.195816
```

```
# Order by column medians, from highest to lowest
effs <- effs[order(apply(effs,1, median, na.rm = TRUE),
             decreasing = TRUE), ]

# Store the order and the medians, these are reused in other figures
sgrna_order <- rownames(effs)
meds <- data.frame("sgRNA" = rownames(effs),
         "medians" = apply(effs[,2:ncol(effs)],1, median, na.rm = TRUE))
meds$sgRNA <- factor(meds$sgRNA, levels = rownames(effs))


print(head(effs))
```

```
##          BWA mem efficiency Blat global efficiency Blat local efficiency
## nrxn1a_L              76.50                  75.99                 75.98
## cntnap4               72.25                  71.76                 71.74
## cnsta                 72.63                  71.38                 71.41
## gjd1a                 71.13                  70.58                 70.59
## cnstb                 67.60                  67.47                 67.49
## cntnap2a              63.18                  63.18                 63.11
##          CRISPR-GA efficiency CRISPR-GA efficiency (inc guide)
## nrxn1a_L             76.65245                               NA
## cntnap4              76.29833                         32.15917
## cnsta                73.10262                         58.22818
## gjd1a                22.69025                         33.60636
## cnstb                65.05621                         57.31839
## cntnap2a             63.44283                         45.46610
##          AmpliconDIVider efficiency CRISPResso efficiency
## nrxn1a_L                    74.6797              46.23358
## cntnap4                     71.0021              58.31127
## cnsta                       71.5162              48.43455
## gjd1a                       69.7860              50.77042
## cnstb                       66.6185              47.43802
```

```
## cntnap2a                    62.5420              51.52375
```

```
# Calculate the difference between the CrispRVariants estimates and the others.
# (Plotted in Figure 11 b.)
effs <- cbind(rownames(effs), effs)
diff_from_crv <- apply(effs[,3:ncol(effs)], 2, function(x) x - effs[,2])
diffs <- reshape2::melt(diff_from_crv)
```

```
# Create plot comparing efficiencies between methods

# Reorder columns so CrispRVariants plots on top
effs <- effs[,c(1,8,7,6,5,4,3,2)]

# Reformat into long format for plotting with ggplot2
m <- melt(effs)
```

```
## Using rownames(effs) as id variables
```

```
colnames(m) <- c("sgRNA", "Condition", "Efficiency")
m$sgRNA <- factor(m$sgRNA, levels = rownames(effs))
m$Condition <- factor(m$Condition, levels = rev(unique(m$Condition)))
print(head(m))
```

```
##      sgRNA             Condition Efficiency
## 1 nrxn1a_L CRISPResso efficiency   46.23358
## 2  cntnap4 CRISPResso efficiency   58.31127
## 3    cnsta CRISPResso efficiency   48.43455
## 4    gjd1a CRISPResso efficiency   50.77042
## 5    cnstb CRISPResso efficiency   47.43802
## 6 cntnap2a CRISPResso efficiency   51.52375
```

```
# Colour-blind safe palette with seven rainbow colours
sevenPalette <- c("#D92120","#E77C30","#D8Af3D","#91BD61",
                  "#4B91C0","#3F56A7","#781C81")

# Labels for legend
cnd_nms <- c("1. CrispRVariants (BWA MEM)","2. CrispRVariants (BLAT global)",
             "3. CrispRVariants (BLAT local)", "4. CRISPR-GA",
             "5. CRISPR-GA with guide",
             "6. AmpliconDIVider (BWA MEM)","7. CRISPResso")

# Find the rows which are mentioned specifically in the text of
# Supplementary Material text
annot <- which(rownames(effs) %in% c("gjd1a_off01", "tjp1b_L",
               "tjp2a","gjd1b","nrxn1b_S"))

# Make a data from of coordinates for the guides to be highlighted
annot <- data.frame(xmin = annot-0.4, xmax=annot+0.4, ymin = -Inf, ymax=Inf)

# Convert to numeric for using scale_x_continuous
# (just for easier formatting of the plot)
m$sgRNA <- as.numeric(m$sgRNA)
```

```
# Plot the mutation efficiency estimates
p1 <- ggplot(m, aes(x=sgRNA, y=Efficiency, group=Condition)) +
  # Highlight the guides discuessed in the text
  geom_rect(data = annot, aes(xmin=xmin,xmax=xmax,ymin=ymin,ymax=ymax,
            group=NULL, x=NULL,y=NULL), alpha = 0.5, fill = "light gray") +
  # Add jitter for visibility
  geom_jitter(size=2.5, alpha = 0.8, height = 0, width = 0.2, aes(color=Condition)) +
  theme_bw() + ggtitle('a.') +
  theme(plot.title=element_text(hjust=-0.1),
        axis.text.x = element_text(angle=90, hjust = 1, vjust = 0.5, size = 7),
        legend.position="bottom", legend.title=element_blank(),
        legend.key=element_blank(),
        plot.margin = grid::unit(c(0.2,0.5,0,0.5), "cm")) +
  ylab("Mutation Efficiency (%)") +

  # Manually set colour and labels
  scale_colour_manual(labels = c(cnd_nms), values=sevenPalette) +

  # Set x labels, remove axis padding
  scale_x_continuous(breaks = seq_along(rownames(effs)),
        labels = rownames(effs), expand = c(0,0))+
  guides(color = guide_legend(ncol=2), title=NULL)

shared_legend <- getLegend(p1)
```

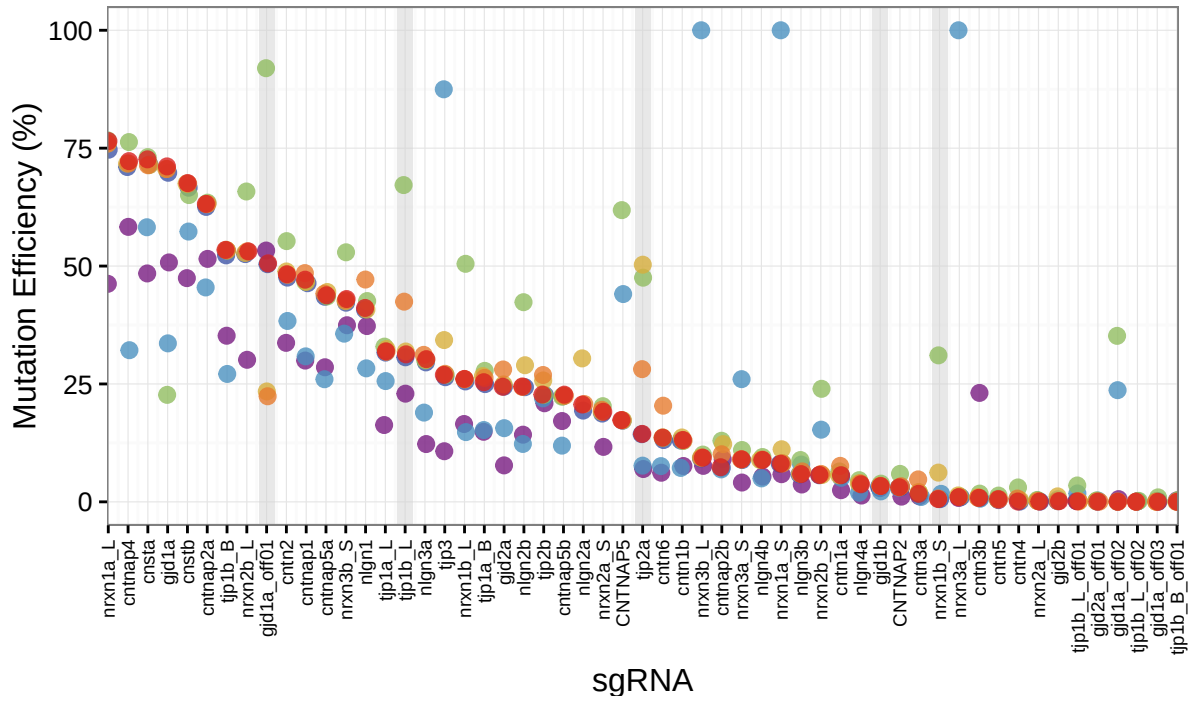
```
## Warning: Removed 13 rows containing missing values (geom_point).
```

```
# Plot the difference between the CrispRVariants estimates and the others
p2 <- ggplot(diffs, aes(x = Var2, y = value, fill = Var2)) +
  # Violin plot showing the distribution
  geom_violin(alpha = 0.5) +
  # Jittered points showing all estimates
  geom_jitter(aes(color = Var2), alpha = 0.8)  + theme_bw() +
  ggtitle('b.') +
  theme(plot.title=element_text(hjust=-0.1),
        axis.text.x = element_text(size = 14),
        axis.title.x = element_blank(),
        legend.position = "none",
        plot.margin = grid::unit(c(0.5,0.5,0,0.5), "cm")) +
  ylab("Eff - Eff(CrispRVariants BWA MEM)") +
  # Use the same palette minus the first colour for CrispRVariants
  scale_fill_manual(values=sevenPalette[2:7]) +
  scale_color_manual(values=sevenPalette[2:7]) +
  scale_x_discrete(labels = c(2:7))

# Arrange the two plots with the shared legend
p3 <- grid.arrange(arrangeGrob(p1 + theme(legend.position="none"),
                          p2 + theme(legend.position="none"),
                          nrow=2),
             shared_legend, nrow=2,heights=c(10, 1.5), newpage = FALSE)
```

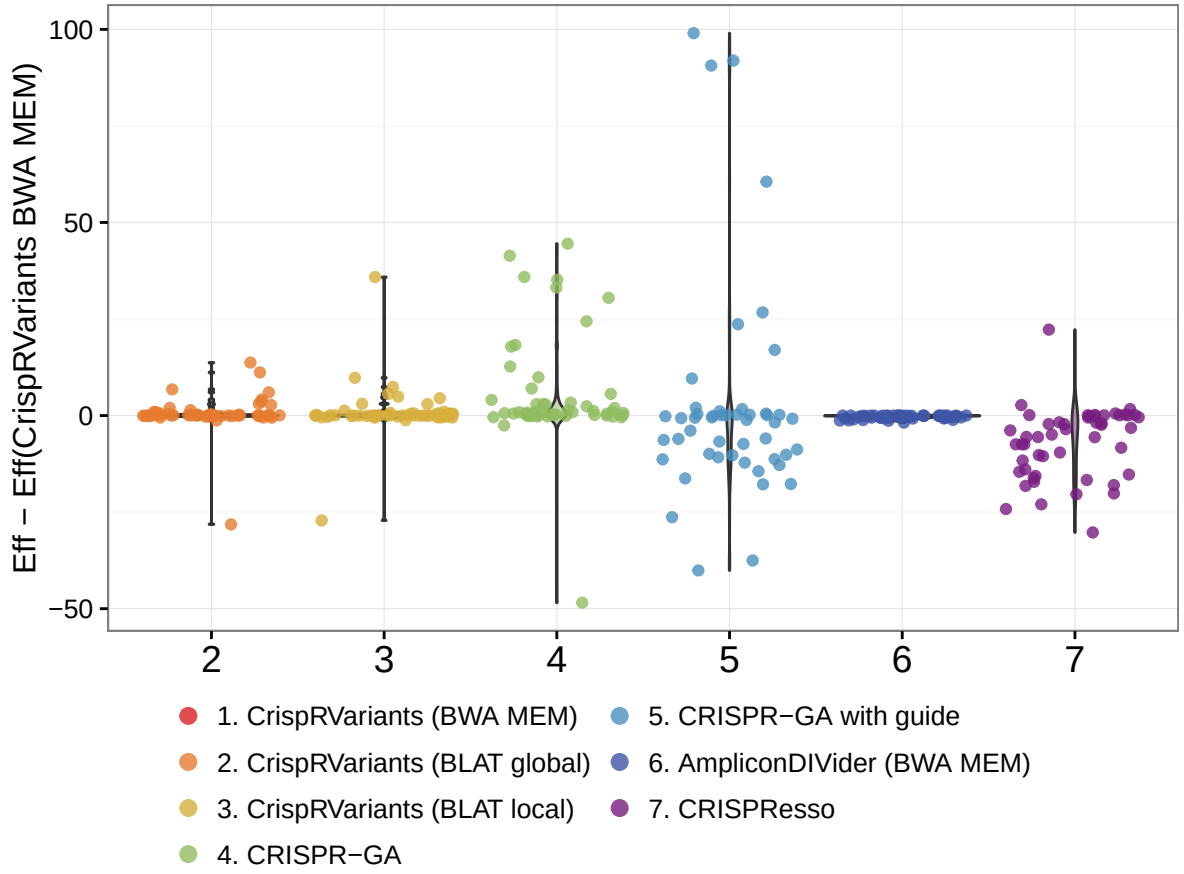
```
## Warning: Removed 13 rows containing missing values (geom_point).
```

```
## Warning: Removed 13 rows containing non-finite values (stat_ydensity).

## Warning: Removed 13 rows containing missing values (geom_point).
```

a.

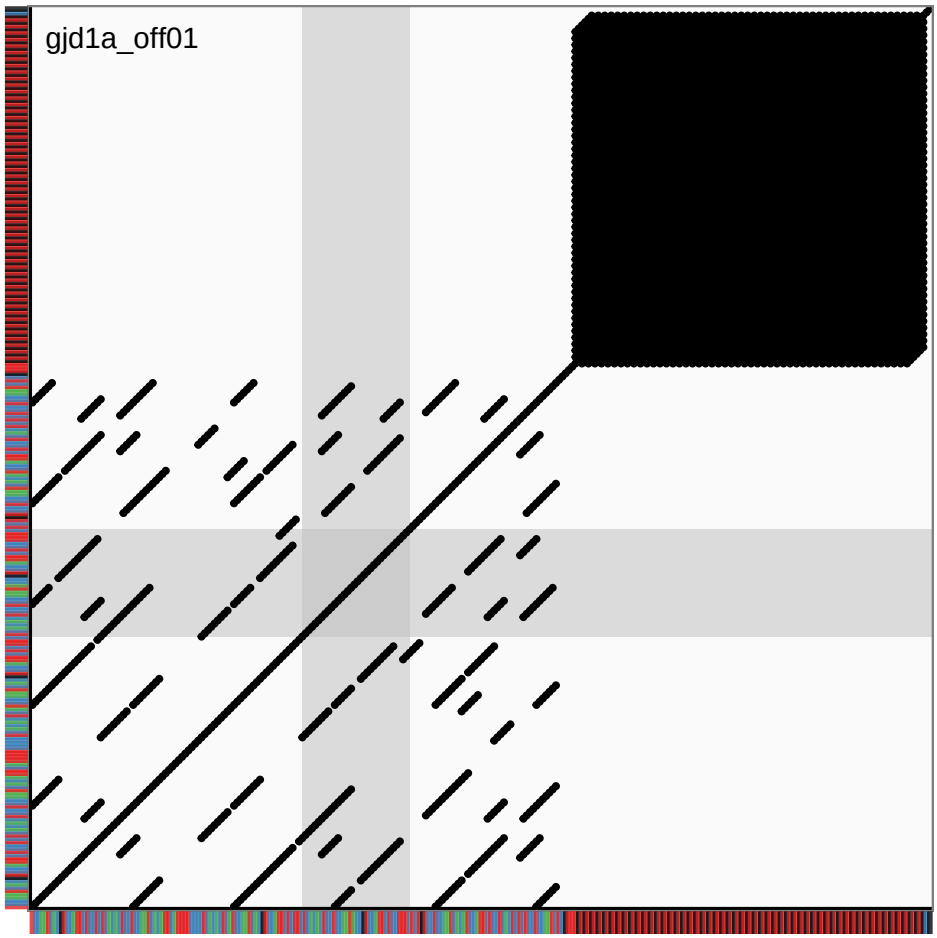

Mutation Efficiency (%)
100
75
50
25
0
nrxn1a_L
cntnap4
cnsta
gjd1a
cnstb
cntnap2a
tjp1b_B
nrxn2b_L
gjd1a_off01
cntn2
cntnap1
cntnap5a
nrxn3b_S
nlgn1
tjp1a_L
tjp1b_L
nlgn3a
tjp3
nrxn1b_L
tjp1a_B
gjd2a
nlgn2b
tjp2b
cntnap5b
nlgn2a
nrxn2a_S
CNTNAP5
tjp2a
cntn6
cntn1b
nrxn3b_L
cntnap2b
nrxn3a_S
nlgn4b
nrxn1a_S
nlgn3b
nrxn2b_S
cntn1a
nlgn4a
gjd1b
CNTNAP2
cntn3a
nrxn1b_S
nrxn3a_L
cntn3b
cntn5
cntn4
nrxn2a_L
gjd2b
tjp1b_L_off01
gjd2a_off01
gjd1a_off02
tjp1b_L_off02
gjd1a_off03
tjp1b_B_off01
sgRNA

b.


Eff − Eff(CrispRVariants BWA MEM)
100
50
0
−50
2
3
4
5
6
7
1. CrispRVariants (BWA MEM)
2. CrispRVariants (BLAT global)
3. CrispRVariants (BLAT local)
4. CRISPR−GA
5. CRISPR−GA with guide
6. AmpliconDIVider (BWA MEM)
7. CRISPResso

Dotplot showing homology within the amplicon sequence for *gjd1a_off01*

```
# Get the primer and guide locations
pr <- primers[primers$name == "gjd1a_off01"]
gd <- guides[guides$name == "gjd1a_off01"]

# Get the amplicon sequence, select first to make a DNAString object
sq <- getSeq(danRer7, pr)[[1]]

# Shift guide location to be with respect to start of the amplicon
annot <- shift(gd, -1 * start(pr) + 1)

# Make the dotplot for runs of at least 6 with no mismatches
p <- dotplot(sq, annotations = annot, min.length = 6,
             plot.title = "gjd1a_off01")
p
```



Plot showing number of reads contributing to efficiency calcuations

In this section, the number of reads mapped by **bwa mem** to the amplicon sequences is counted. If a program counts members of a read pair separately, counts would be consistently 100% greater than the mapped counts.

CRISPResso uses **bowtie2** rather than **bwa mem**. We checked the number of reads mapped by each tool. The number of reads mapped does not differ substantially although the alignments themselves do.

```
# Count number of unique reads mapped to amplicons
result <- onTargetCount(merged_split_pear, primers)
identical(rownames(result), rownames(crv_counts))
```

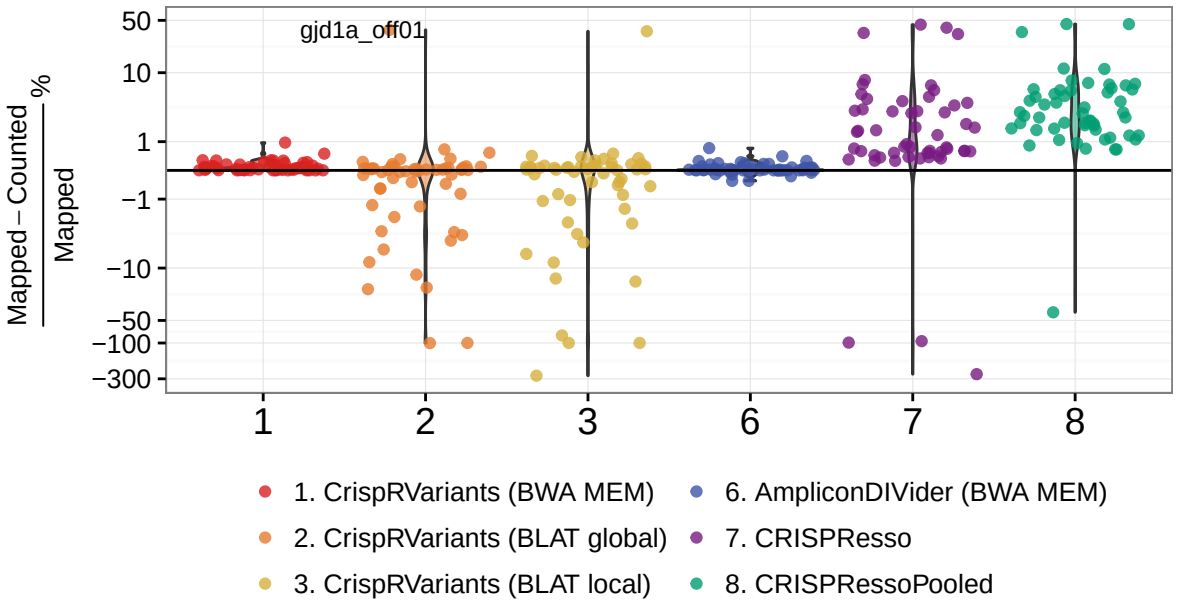
```
## [1] TRUE
```

```
# Make a data frame of read counts reported by each method
counts <- do.call(cbind, list(crv_counts, adiv_counts[rownames(crv_counts)],
                  crispresso_std_counts[rownames(crv_counts)],
                  crispresso_pooled_counts[rownames(crv_counts)]))
colnames(counts) <- c(colnames(crv_counts), "AmpliconDIVider",
                      "CRISPResso","CRISPRessoPooled")

# Calculate the percentage of the reads aligned to the amplicon
# that contribute to the count
counts <- result - counts
counts <- as.matrix(counts)/result[,1] * 100

# These are the five of the seven conditions that report counts
# (for consistent colours between points).  CRISPRessoPooled
# additionally included in this table.
cnds_w_counts <- c(1,2,3,6,7)
cnts <- melt(counts)

# Use asinh transformation for symmetrically compressing the larger
# data values when plotting
asinh_trans <- function(){
  trans_new(name = 'asinh', transform = function(x) asinh(x),
            inverse = function(x) sinh(x))
}

# Disable exponential notation
options(scipen = 999)
ybreaks <- c(-300, -100, -50, -10, -1,1, 10, 50, 100)

# Annotate gjd1a_off01 in the BLAT
annot <- cnts[cnts$Var1 == "gjd1a_off01" &
                cnts$Var2 == "Blat global reads","value"]

p1 <- ggplot(cnts, aes(x = Var2, y = value, fill = Var2)) +
  # Violin plot of overall distribution
  geom_violin(alpha = 0.5) +
  # Plot individual points with jitter for visibility
  geom_jitter(aes(color = Var2), alpha = 0.8)  + theme_bw() +
  # Horizontal line at y = 0
  geom_hline(yintercept=0) +

  theme(plot.title=element_text(hjust=-0.1),
        axis.text.x = element_text(size = 14),
        axis.title.x = element_blank(),
```

```
        axis.title.y = element_text(size = 10),
        axis.text.y = element_text(size = 10),
        legend.position = "bottom",
        legend.key=element_blank(),
        plot.margin = grid::unit(c(0.5,0.5,0,0.5), "cm")) +

  ylab(expression(frac(Mapped - Counted, Mapped)~"%")) +

  # Add colours and labels for CRISPRessoPooled
  scale_color_manual(values=c(sevenPalette[cnds_w_counts],"#009E73"),
                     labels = c(cnd_nms[cnds_w_counts],
                                "8. CRISPRessoPooled")) +
  scale_fill_manual(guide = FALSE,
                    values=c(sevenPalette[cnds_w_counts], "#009E73")) +
  scale_x_discrete(labels = c(cnds_w_counts,"8")) +

  # Transform the y axis to compress the large ranges
  scale_y_continuous(trans = "asinh", breaks = ybreaks,
                     labels = as.character(ybreaks)) +

  guides(color = guide_legend(ncol=2, title = NULL)) +
  annotate("text", label = "gjd1a_off01", x = 2, y = annot, size = 3, hjust = 1)

p1
```

```
## Warning: Removed 1 rows containing non-finite values (stat_ydensity).
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



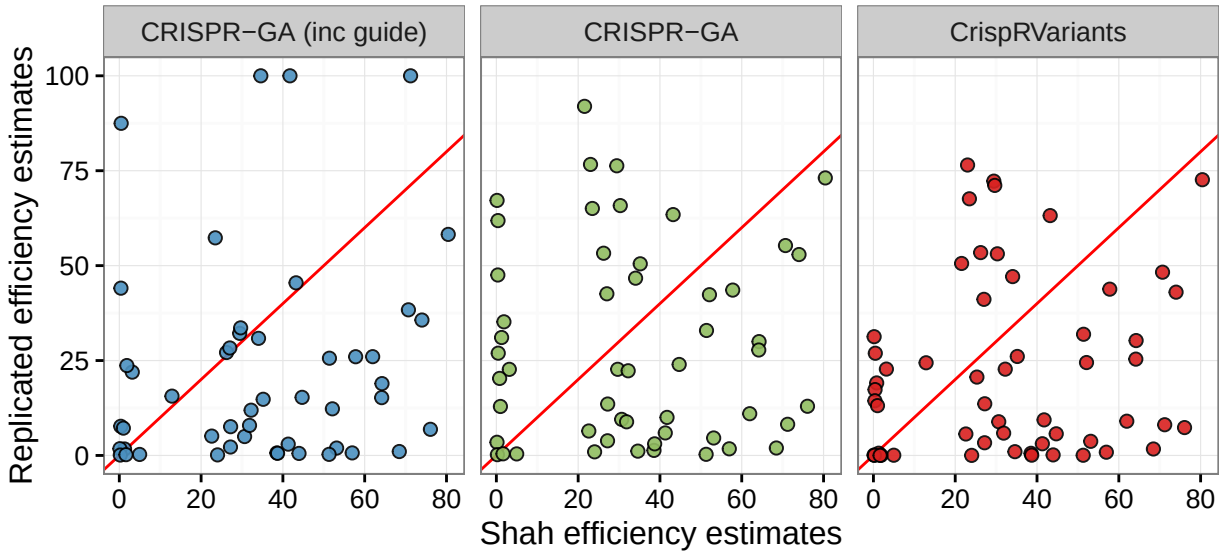
Supplementary Note 9: Background to the data and tools used

```
shah_results <- read.table("../annotation/Shah_metadata_edited.txt", sep = "\t",
                           header = TRUE)
guide_nms <- gsub("\ ", "", shah_results$Gene)

compare_effs <- effs[,c("CRISPR-GA efficiency (inc guide)",
                        "CRISPR-GA efficiency","BWA mem efficiency")]

compare_effs <- cbind(rownames(compare_effs), compare_effs)
colnames(compare_effs) <- c("sgRNA", "CRISPR-GA (inc guide)",
                            "CRISPR-GA", "CrispRVariants")
original_eff <- shah_results$Efficiency
names(original_eff) <- shah_results$Gene

m <- melt(compare_effs)
```

```
## Using sgRNA as id variables
```

```
m$Original <- original_eff[m$sgRNA]

ggplot(m, aes(x=Original, y=value, fill = variable)) +
     geom_abline(slope = 1, colour = "red", size = 0.5) +
     geom_point(size = 2, shape = 21, alpha = 0.9) +
     facet_grid(~variable) +
xlab("Shah efficiency estimates") +
ylab("Replicated efficiency estimates") + theme_bw() +
theme(axis.title.x=element_text(vjust=-0.5),
      axis.title.y=element_text(vjust=1)) +
scale_fill_manual(values = c("#4B91C0","#91BD61","#D92120"), guide = FALSE)
```

```
## Warning: Removed 9 rows containing missing values (geom_point).
```



Supplementary Note 6: Merging paired reads affects efficiency estimates by non-randomly filtering reads

```
# Setup conditions, condition names and plotting colours
directories <- c(merged_seqprep, merged_split_l55,
                 merged_split_pear, split_merged_pear,
                 tolerant_pear, strict_pear)

cnd_nms <- c("1. Unmerged","2. Merged (SeqPrep)",
             "3. SeqPrep (merge-then-split)","4. SeqPrep (length 50, merge-then-split)",
             "5. PEAR (merge-then-split)","6. PEAR (split-then-merge)",
             "7. PEAR (tolerant)", "8. PEAR (strict)")

cbbPalette <- c("#000000", "#E69F00", "#56B4E9", "#009E73",
                "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
```

We first calculate mutation efficiencies for the two unsplit bam files. Here we use CrispRVariants::readsToTargets as there are reads from multiple targets in each bam file.

```
# Unmerged = reads mapped as pairs
bpparam = BiocParallel::MulticoreParam(8)
originals <- readsToTargets(unmerged, targets = guides, references = references,
                    target.loc = 22, collapse.pairs = TRUE, bpparam = bpparam,
                    verbose = FALSE)
original_effs <- data.frame(t(sapply(originals, function(crispr_set){
  mutationEfficiency(crispr_set)[c("Overall", "ReadCount")]})))

rm(originals)
invisible(gc())

# Merged = reads merged
merged_csets <- readsToTargets(merged, targets = guides, references = references,
                    target.loc = 22, collapse.pairs = FALSE, bpparam = bpparam,
                    verbose = FALSE)

merged_effs <- data.frame(t(sapply(merged_csets, function(crispr_set){
  mutationEfficiency(crispr_set)[c("Overall", "ReadCount")]})))

rm(merged_csets)
invisible(gc())
```

We next get the mutation efficiencies for the reads split by guide, as in Supplementary Note 5.

```
split_effs <- lapply(directories, getSplitEff,
                    references = references, guides =  guides,
                    primers = primers)

# Make a data frame of the results, checking that order is consistent
effs <- cbind(original_effs[rownames(split_effs[[1]]),],
         merged_effs[rownames(split_effs[[1]]),], split_effs)
cols <- rep(c(c("Original", "Merged"), basename(directories)), each = 2)
```

```r
colnames(effs) <- paste0(cols, c("Eff", "ReadCount"))

# Split into read counts and mutation efficiencies
counts <- effs[,grep("ReadCount", colnames(effs))]
effs <- effs[,grep("Eff", colnames(effs))]

# Order by median as in Supplementary Note 5, format for plotting
effs <- cbind(sgRNA = rownames(effs), effs)
effs$sgRNA <- factor(effs$sgRNA, levels = sgrna_order)
m <- melt(effs)
```

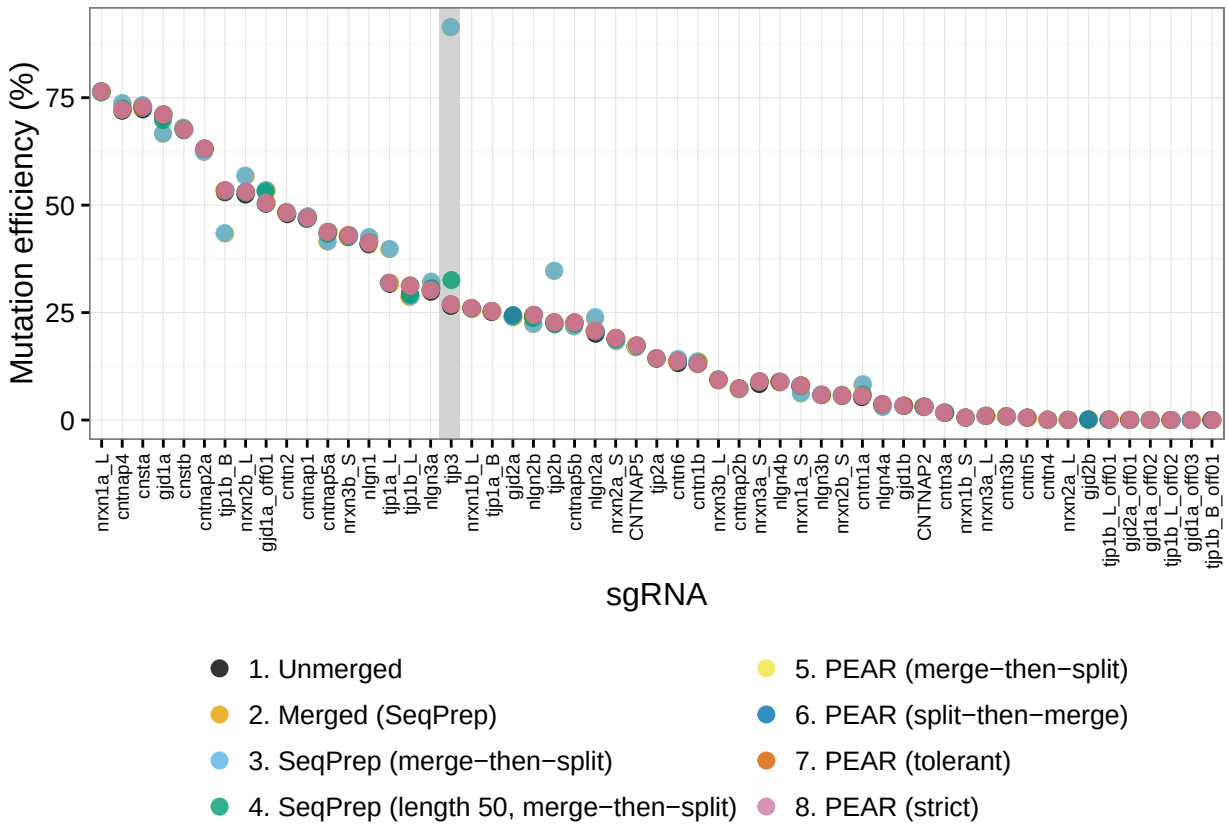
```
## Using sgRNA as id variables
```

```r
# Check how different the efficiency estimates are from
# those for the unprocessed data

ee <- effs[,2:9]
mins <- apply(ee, 1, min, na.rm = TRUE)
maxs <- apply(ee, 1, max, na.rm = TRUE)
table(maxs - mins > 10)
```

```
##
## FALSE  TRUE
##    52     3
```

```r
# Plot
ggplot(m, aes(x=sgRNA, y=value, group=variable)) +
  geom_rect(xmin=17.4, xmax=18.4, ymin=-Inf, alpha = 0.2,
            ymax=Inf, fill = "light gray") +  # highlight the outlier
  geom_jitter(size=2.5, alpha = 0.8, height = 0, width = 0.1, aes(color=variable)) +
  theme_bw() +
  theme(axis.text.x = element_text(angle=90, hjust = 1, vjust = 0.5, size = 7),
        legend.position="bottom", legend.title=element_blank(),
        legend.key=element_blank(),
        plot.margin = grid::unit(c(0.2,0.2,0,0.5), "lines")) +
        ylab("Mutation efficiency (%)") +
  scale_colour_manual(labels = c(cnd_nms), values=cbbPalette) +
  guides(color = guide_legend(ncol=2), title=NULL)
```

```
## Warning: Removed 4 rows containing missing values (geom_point).
```



Supplementary Note 7: Data preprocessing affects efficiency estimates

Effect of preprocessing on CRISPR-GA estimates

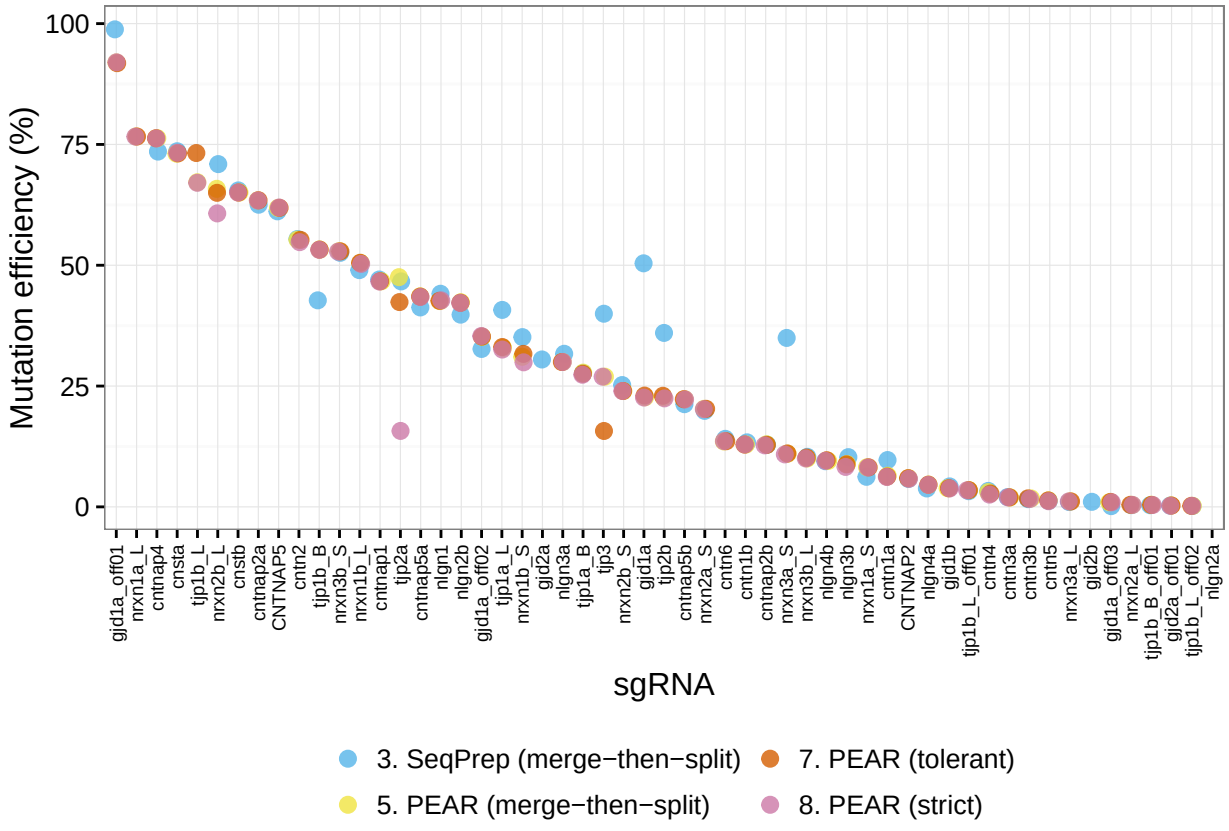
```
crisprga <- "../results/CRISPRGA"

fnms <- c("seqprep_originals.txt",
          "fwd_only_pear_crisprga_refonly.txt",
          "tolerant_pear_crisprga_refonly.txt",
          "strict_pear_crisprga_refonly.txt")

get_cga <- function(fname, cga_dir = crisprga){
  temp <- read.table(file.path(cga_dir,fname), sep = "\t", header = TRUE)
  crisprga <- temp$crisprga
  names(crisprga) <- temp$name
  crisprga
}

effs <- lapply(fnms, function(nm) get_cga(nm))
effs <- lapply(effs, function(x) x[names(effs[[1]])])

effs <- data.frame(do.call(cbind, effs))
colnames(effs) <- gsub("merged_|split_", "", fnms)
```

```
effs <- effs[order(apply(effs[,1:ncol(effs)],1, median, na.rm = TRUE),
             decreasing = TRUE), ]
effs <- cbind(sgRNA = rownames(effs), effs)
effs$sgRNA <- factor(effs$sgRNA, levels = effs$sgRNA)

m <- melt(effs)
```

```
## Using sgRNA as id variables
```

```
ggplot(m, aes(x=sgRNA, y=value, group=variable, color=variable)) +
  geom_jitter(size=2.5, alpha = 0.8, height = 0, width = 0.2) + theme_bw() +
  theme(axis.text.x = element_text(angle=90, hjust = 1, vjust = 0.5, size = 7),
        legend.position="bottom", legend.title=element_blank(),
        legend.key=element_blank(),
        plot.margin = grid::unit(c(0.2,0.2,0,0.5), "lines")) +
        ylab("Mutation efficiency (%)") +
  scale_colour_manual(labels = c(cnd_nms)[c(3,5,7,8)], values=cbbPalette[c(3,5,7,8)]) +
  guides(color = guide_legend(ncol=2), title=NULL)
```

```
## Warning: Removed 12 rows containing missing values (geom_point).
```



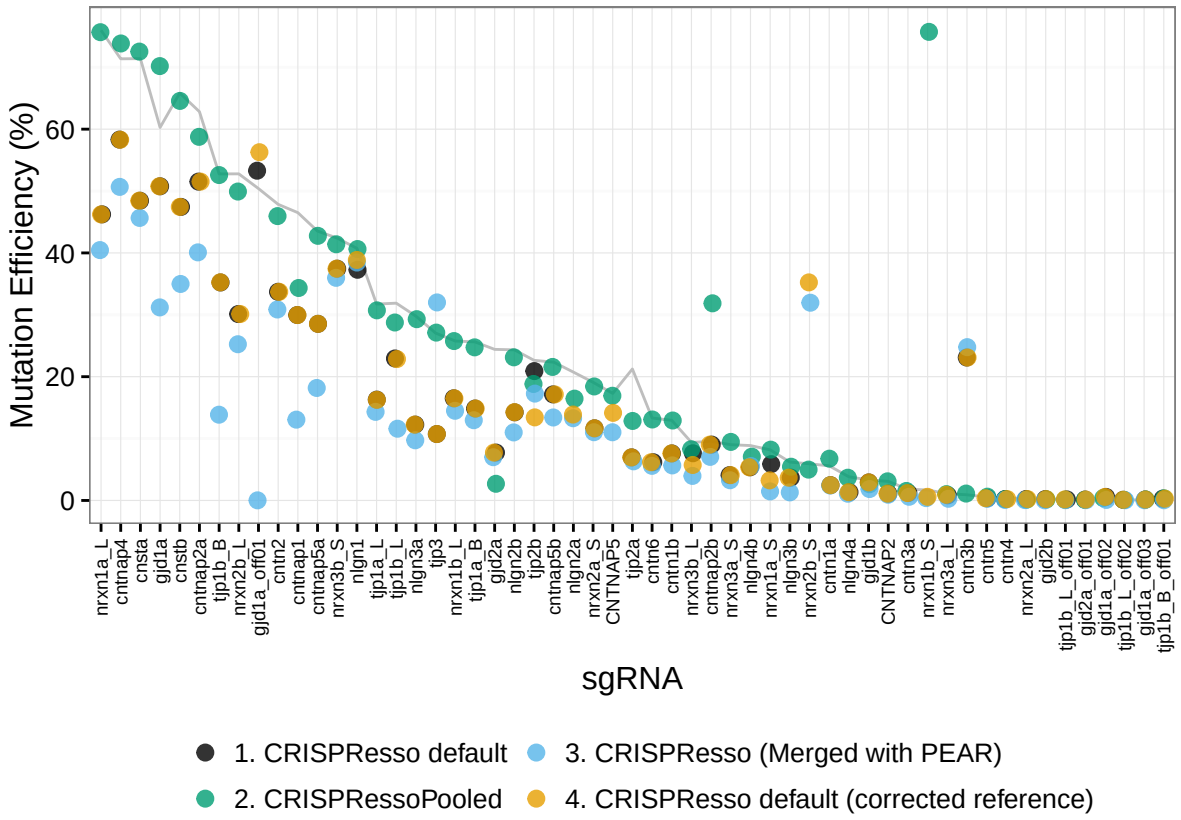
Effect of preprocessing on CRISPResso estimates

```
cnd_nms <- c("1. CRISPResso default", "2. CRISPRessoPooled",
             "3. CRISPResso (Merged with PEAR)",
             "4. CRISPResso default (corrected reference)")

cbbPalette <- c("#000000", "#009E73", "#56B4E9", "#E69F00")

# Reorder as in local-global comparison
crispresso_dat <- crispresso_dat[sgrna_order, c(1,4,3,2)]
crispresso_dat <- cbind(rownames(crispresso_dat), crispresso_dat)
m <- melt(crispresso_dat)
```

```
## Using rownames(crispresso_dat) as id variables
```

```
colnames(m) <- c("sgRNA","Condition","Efficiency")
m$sgRNA <- factor(m$sgRNA, levels = sgrna_order)

# Note that data.frame meds was created in the local-global comparison

p1 <- ggplot(meds, aes(x=sgRNA)) +
  geom_line(aes(y = medians,group = 1), color = "gray") +
  geom_jitter(data = m, aes(group = Condition, y=Efficiency, color=Condition),
              size=2.5, alpha = 0.8, height = 0, width = 0.2) + theme_bw() +
  theme(plot.title=element_text(hjust=-0.1),
        axis.text.x = element_text(angle=90, hjust = 1, vjust = 0.5, size = 7),
        legend.position="bottom", legend.title=element_blank(),
        legend.key=element_blank(), plot.margin = grid::unit(c(0.2,0.5,0,0.5), "cm")) +
  ylab("Mutation Efficiency (%)") +
  scale_colour_manual(labels = c(cnd_nms), values=cbbPalette) +
  guides(color = guide_legend(ncol=2), title=NULL)
p1
```

```
## Warning: Removed 5 rows containing missing values (geom_point).
```



Supplementary Note 1: Localising variants facilitates comparison of sgRNAs

The CrispRVariants representation of variant alleles allows variant spectra to be directly compared between guides. There are many possible ways to compare spectra. Here we compare variant location with variant type.

```
csets <- getCrisprSets(merged_split_pear, references, guides)
# Keep on-target guides only
on_tgt <- !grepl("off", names(csets))


# Select frequent variants, extract locations and variant types
dat <- lapply(seq_along(csets[on_tgt]), function(i){
  nm <- names(csets[on_tgt])[[i]]

  # Get a table of proportions for variants at least 2% frequent,
  # including only indel variants and excluding chimeras
  vc <- CrispRVariants::variantCounts(csets[on_tgt][[i]], min.freq = 2,
     result = "proportions", include.nonindel = FALSE,
     include.chimeras = FALSE)

  if (length(vc) == 0) return(NULL)

  # Exclude complex variants, i.e. variants with two different indels.
```

```
  # Variants are represened as location:variant, split the names
  complex_var <- grepl(",", rownames(vc))
  temp <- strsplit(rownames(vc)[!complex_var], ":")
  loc <- as.numeric(lapply(temp, "[[", 1))
  variant <- sapply(temp, "[[", 2)

  # Make a table of variant location, type, frequency and guide names
  result <- data.frame("Location" = loc, "Variant" = variant,
                       "Frequency" = vc[!complex_var,1], "sgRNA" = nm)
  result
})

# Merge results into a single table
dat <- do.call(rbind, dat)

# Exclude gjd2a as the mapped variants do not correspond to the mapped guide,
# possibly indicating something wrong with the mapping
dat <- dat[dat$sgRNA != "gjd2a",]

# Collapse large deletions (15 bases or more) into a single category
large_del <- as.numeric(gsub("D", "", dat$Variant)) >= 15
```

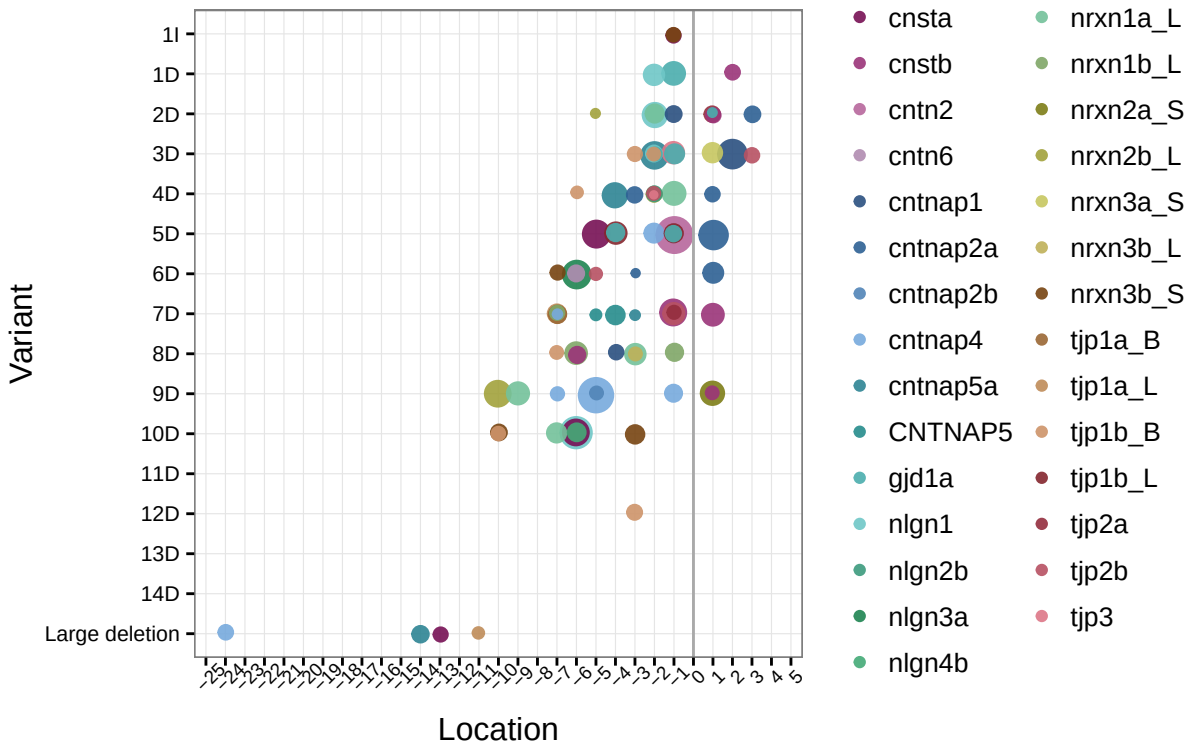
```
## Warning: NAs introduced by coercion
```

```
large_del[is.na(large_del)] <- FALSE
dat$Variant <- as.character(dat$Variant)
dat$Variant[large_del] <- "Large deletion"

# For visibility, order so that the largest frequencies plot first
dat <- dat[order(dat$Frequency, decreasing = TRUE),]
```

Create plot for Supplementary Note 1.

```
# This is a large, colour-blind safe palette.
large_palette <- c("#771155","#AA4488","#CC99BB","#114477","#4477AA",
                   "#77AADD","#117777", "#44AAAA","#77CCCC","#117744",
                   "#44AA77","#88CCAA","#777711","#AAAA44",
                   "#DDDD77","#774411","#AA7744","#DDAA77",
                   "#771122","#AA4455","#DD7788")

# Extrapolate palette to the number of guides
pal <- colorRampPalette(large_palette)(length(unique(dat$sgRNA)))

# Set up the labels for the axes.  Note that I previously the
# range of the data before setting this range.
temp <- c(-14:-1, 1)
is_del <- temp < 0
temp[is_del] <- gsub("-", "", paste0(temp[is_del], "D"))
is_ins <- !is_del
temp[is_ins] <- paste0(temp[is_ins], "I")
temp <- c("Large deletion", temp)
```

```
# Convert Location and Variant to factors, set the order
dat$Location <- factor(dat$Location, levels = c(-25:5))
dat$Variant <- factor(dat$Variant, levels = temp)

# Create the plot, using position_jitter to make points more visible
ggplot(dat, aes(x = Location, y = Variant,  color=sgRNA)) +
 geom_point(aes(size = Frequency),
            position=position_jitter(width = 0.1, height = 0.1),
      alpha = 0.9) + theme_bw() +
      geom_vline(xintercept = 26, color = "darkgray", size = 0.5) +

# Don't drop unused categories from the axes
scale_x_discrete(drop=FALSE) + scale_y_discrete(drop=FALSE) +
theme(axis.text.x = element_text(size = 7, hjust = 0.5, angle = 45),
      axis.text.y = element_text(size = 8),
      legend.key = element_blank()) +

# Make a guide for colour but not for size
guides(color=guide_legend(ncol=2, title=  NULL)) +
  scale_size_continuous(guide = FALSE) +

# Manually set colours to the chosen palette
scale_color_manual(values = pal)
```

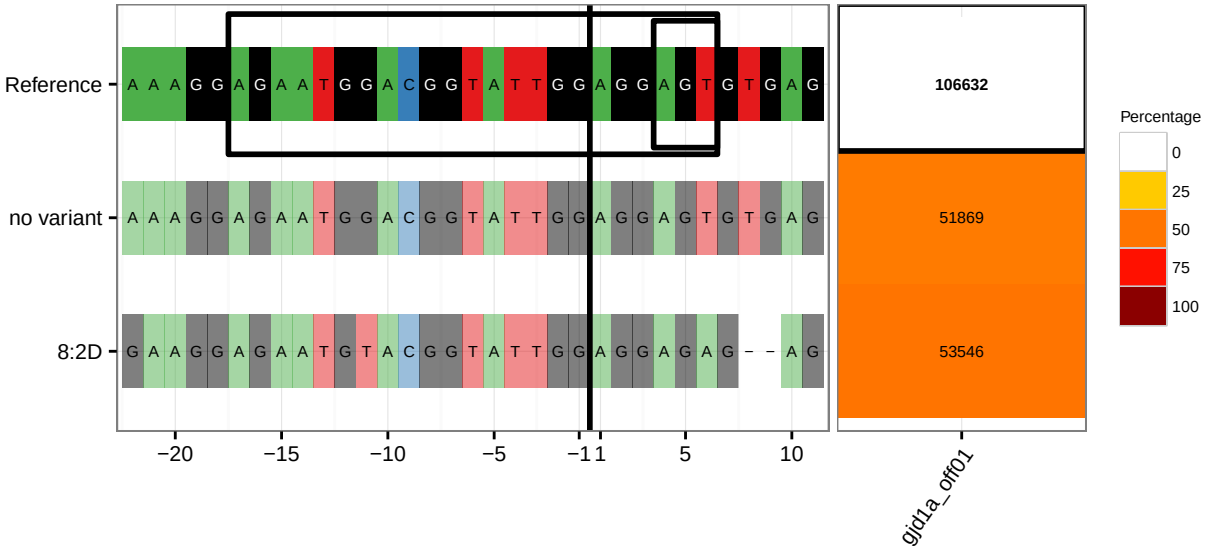


Supplementary Note 3: CrispRVariants can detect and adjust efficiency estimates to account for genomic variants

This analysis is based on the merged-split-pear preprocessing pipeline. We can use the CrisprSet objects created for Supplementary Note 1.

Plot heterozygous 2bp deletion

```
# One heterozygous 2bp deletion was detected by looking at the allele
# summary plots.  Plot the alleles for this guide (gjd1a_off01)
gjd1a_off01 <- csets[["gjd1a_off01"]]
plotVariants(gjd1a_off01, plotAlignments.args = list(min.freq = 1),
             plotFreqHeatmap.args = list(min.freq = 1, x.angle = 55))
```




```
## TableGrob (2 x 1) "arrange": 2 grobs
##   z     cells    name                grob
## 1 1 (1-1,1-1) arrange rect[GRID.rect.2906]
## 2 2 (2-2,1-1) arrange      gtable[arrange]
```

```
# Identify SNVs with frequency at least 0.2
existing_snps <- unlist(sapply(csets, function(cset) findSNVs(cset, freq = 0.2)))
existing_snps
```

```
##    cntn3b.4  CNTNAP5.-5 nrxn1b_S.-1 nrxn1b_S.-5  nrxn2b_S.5
##    21.50077    71.88104    75.12340    24.75988    30.68157
```

```
# Make a list of SNVs to remove per guide
snps <- do.call(rbind, strsplit(names(existing_snps), "\\."))
colnames(snps) <- c("guide", "location")
snps[,"location"] <- paste0("SNV:", snps[,"location"])
remove_snps <- split(snps[,"location"], snps[,"guide"])
remove_snps
```

```
## $cntn3b
## [1] "SNV:4"
##
## $CNTNAP5
## [1] "SNV:-5"
##
## $nrxn1b_S
## [1] "SNV:-1" "SNV:-5"
##
## $nrxn2b_S
## [1] "SNV:5"
```

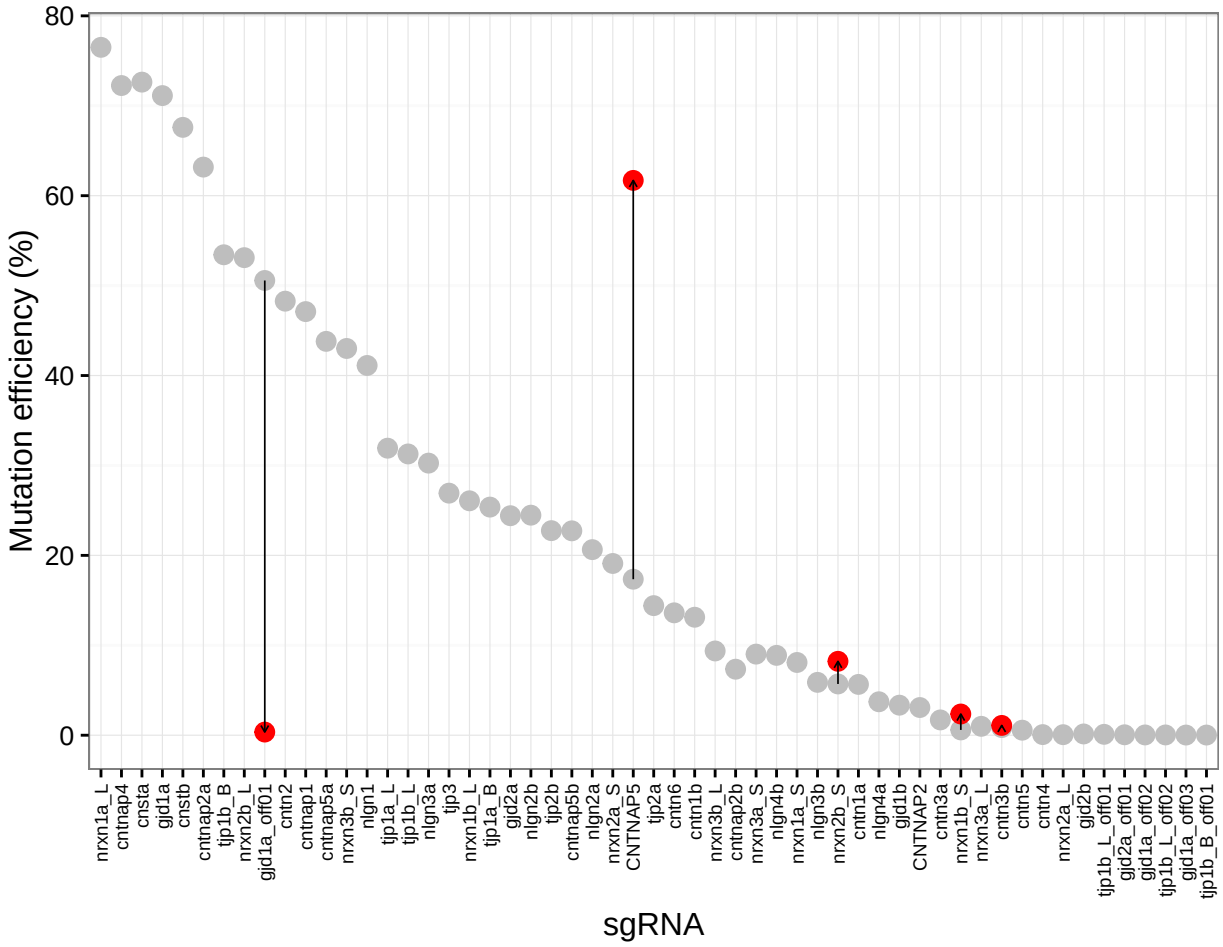
```
# Add the 2bp deletion to this list
remove_snps["gjd1a_off01"] <- "8:2D"

# Calculate the original mutation efficiency
efficiency <- sapply(csets, function(x) mutationEfficiency(x)[["Overall"]])

# Re-calculate mutation efficiency, removing the identified SNPs
efficiency_custom <- sapply(names(csets), function(x){
   eff <- mutationEfficiency(csets[[x]], filter.vars = remove_snps[[x]])
   eff[["Overall"]]
})
```

Plot the efficiency estimates before and after adjusting for SNPs

```
# Plot the original and re-estimated mutation efficiency estimates

snp <- data.frame("Original" = efficiency, "Adjusted" = efficiency_custom)
snp <- cbind("sgRNA" = rownames(snp), snp)

#Re-order SNPs to match the Supplementary Note 5
snp <- snp[sgrna_order,]

snpm <- reshape2::melt(snp)
```

```
## Using sgRNA as id variables
```

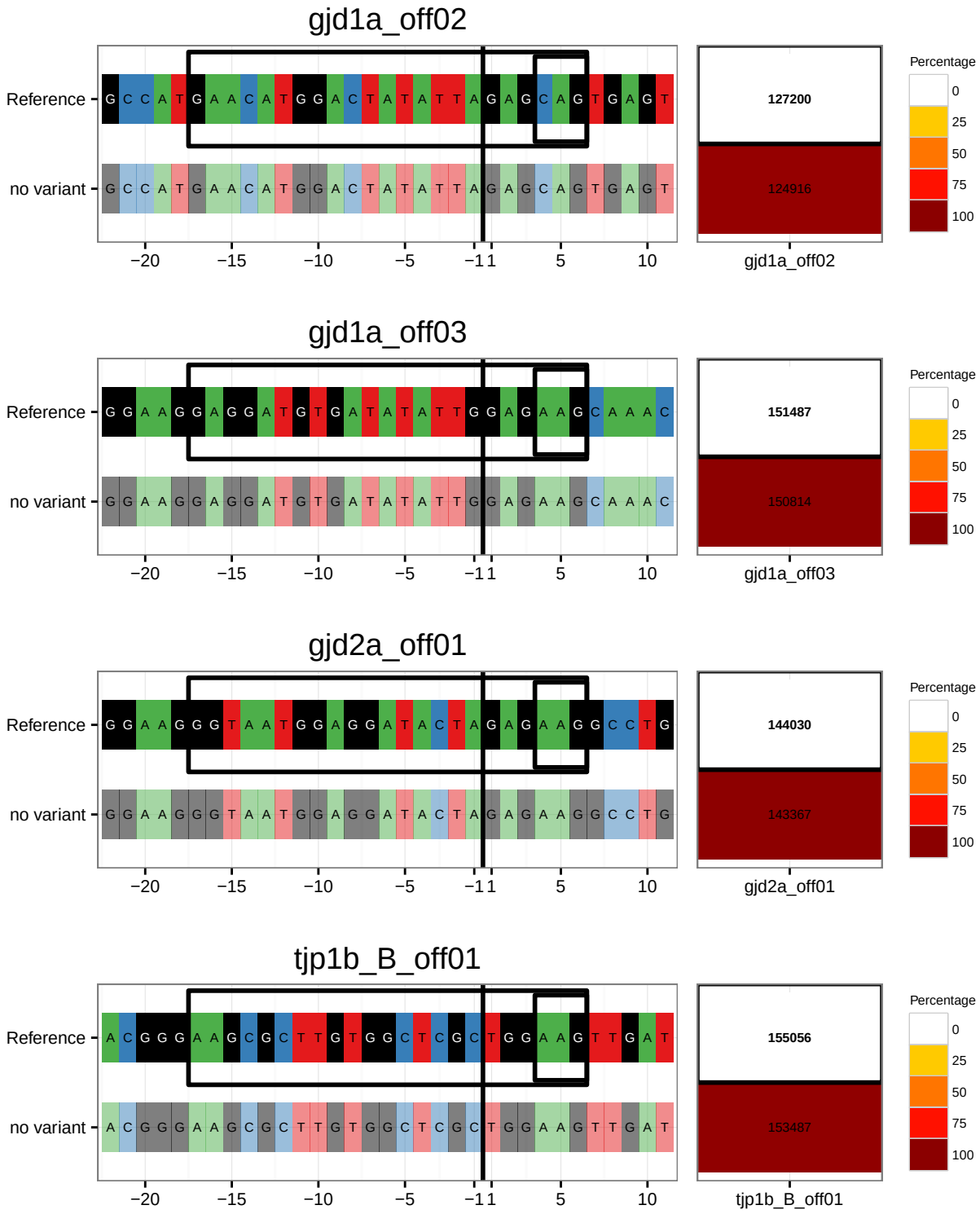
```
# Remove re-calculated estimates that don't change
snpm <- snpm[(snpm$sgRNA %in% names(remove_snps)|snpm$variable == "Original"), ]
snpm$sgRNA <- factor(snpm$sgRNA, levels = sgrna_order)

lns <- snp[names(remove_snps),]

ggplot(snpm) + geom_point(size=3,aes(x=sgRNA, y=value, color=variable)) +
  geom_segment(data = lns, aes(x=sgRNA, xend=sgRNA, y=Original,yend=Adjusted),
               arrow = arrow(length = unit(0.1,"cm")), size = 0.3) +
  theme_bw() + ylab("Mutation efficiency (%)") +
  theme(axis.text.x = element_text(angle=90, hjust = 1, vjust = 0.5, size = 7)) +
  scale_color_manual(values=c("gray","red")) + guides(color = FALSE)
```



Plot the off-target guides with SNPs

```
p1 <- plot_variants(csets[["gjd1a_off02"]], "gjd1a_off02")
p2 <- plot_variants(csets[["gjd1a_off03"]], "gjd1a_off03")
p3 <- plot_variants(csets[["gjd2a_off01"]], "gjd2a_off01")
p4 <- plot_variants(csets[["tjp1b_B_off01"]], "tjp1b_B_off01")
```

```
p5 <- grid.arrange(arrangeGrob(p1,p2,p3,p4, nrow = 4), newpage = FALSE)
```



Supplementary Note 2: Sequencing errors and alignment uncertainty affect variant count, placement and size

This section uses two examples of Sanger sequencing data and one example of MiSeq data from Burger *et al* (in review) to show how alignment issues can affect results.

The MiSeq data is available from ArrayExpress with accession E-MTAB-4143.

Creating the txdb

The transcript database was created using the `GenomicFeatures::makeTxDbFromUCSC` function, with Ensembl gene annotation.

Supplementary Note 4: Synthetic datasets for benchmarking CRISPR sequencing analysis tools

The synthetic data set is included in the CrispRVariants package as raw and mapped reads. The guide is available in bed format.

Synthetic data set 1

Pairwise-align synthetic data and plot using CrispRVariants

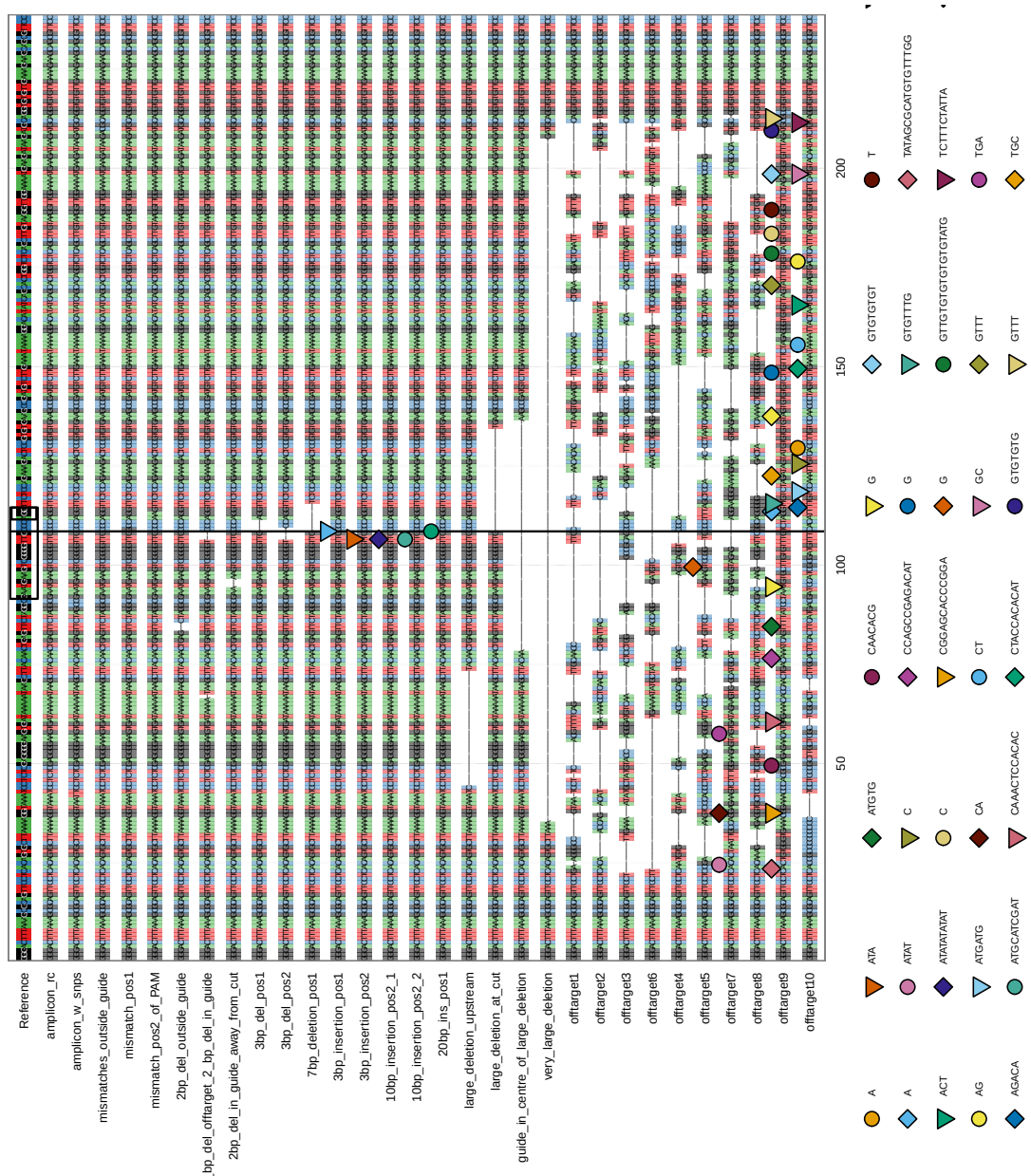
This section demonstrates how to plot data using CrispRVariants::plotVariants.matrix, i.e., without making a CrisprSet object.

```
library(ShortRead)

amp <- paste0("GGGACTTTAAAGCGCAGTTCCTCACAGTGCTTAAAAGGTAAATCCTCTCGA",
              "GGGGAAGTGATAAAAATAAGCTTACAACTGAGTTCTAGGCGAATGAAGTCG",
              "GGGTTGCCCAGGTTCTCCAGAAAGACTCCGTGTGAAGCCGATGTCTTGAAA",
              "TAAAAGGACATATCAGCACTGGTCTCAGCTTGTAAGGTTGGAAAATGAAGA",
              "TAAGATGCAGGTGTGTTGAAAGAAGCAGCGTTCC")

amp <- Biostrings::DNAString(amp)

test_fastq <- system.file("extdata", "cntnap2b_test_data.fastq.gz",
                          package = "CrispRVariants")

# Read the fastq file, format the names
sr <- ShortRead::readFastq(test_fastq)
reads <- ShortRead::sread(sr)
names(reads) <- gsub("cntnap2b_", "", id(sr))

# Make Needleman-Wunsch pairwise alignments of the amplicon sequence to the
# reads and their reverse complements
pa <- Biostrings::pairwiseAlignment(DNAStringSet(replicate(length(reads), amp)),
                                    reads)
rc <- Biostrings::pairwiseAlignment(DNAStringSet(replicate(length(reads), amp)),
                          ShortRead::reverseComplement(reads))

# Format pairwise alignments for CrispRVariants::plotVariants
result <- getReferenceGaps(pa, reverse.alns = rc)
names(result$seqs) <- names(reads)
reads <- as.character(result$seqs)
```

```
id_to_nm <- names(reads)
names(id_to_nm) <- seq_along(reads)

# The "cigar" column should match the row names of the plot,
# in this case these are individual reads, not sets of reads
result$insertions$cigar <- id_to_nm[result$insertions$id]
result$insertions$count <- 1

# Order the reads by the size of the mutation
read_order <- c("amplicon_rc","amplicon_w_snps","mismatches_outside_guide",
    "mismatch_pos1","mismatch_pos2_of_PAM","2bp_del_outside_guide",
    "1_bp_del_offtarget_2_bp_del_in_guide","2bp_del_in_guide_away_from_cut",
    "3bp_del_pos1","3bp_del_pos2","7bp_deletion_pos1","3bp_insertion_pos1",
    "3bp_insertion_pos2","10bp_insertion_pos2_1","10bp_insertion_pos2_2",
    "20bp_ins_pos1","large_deletion_upstream","large_deletion_at_cut",
    "guide_in_centre_of_large_deletion","very_large_deletion",
    "offtarget1","offtarget2","offtarget3","offtarget6","offtarget4",
    "offtarget5","offtarget7","offtarget8","offtarget9","offtarget10")

reads <- reads[read_order]
```

```
# Plot the pairwise alignments using CrispRVariants::plotVariants.matrix

p <- plotAlignments(amp, alns = reads, ins.sites = result$insertions,
               pam.start = 112, pam.end = 114, target.loc = 108,
               legend.cols = 8, plot.text.size = 1.5, axis.text.size = 6,
               legend.symbol.size = 3, legend.text.size = 5, line.weight = 0.5)
p <- p + theme(plot.margin = grid::unit(c(0.3,0.3,0,0.3), "lines"),
               legend.margin = grid::unit(0, "lines"),
               axis.ticks = element_blank())
```

p


Reference
amplicon_rc
amplicon_w_snps
mismatches_outside_guide
mismatch_pos1
mismatch_pos2_of_PAM
2bp_del_outside_guide
1_bp_del_offtarget_2_bp_del_in_guide
2bp_del_in_guide_away_from_cut
3bp_del_pos1
3bp_del_pos2
7bp_deletion_pos1
3bp_insertion_pos1
3bp_insertion_pos2
10bp_insertion_pos2_1
10bp_insertion_pos2_2
20bp_ins_pos1
large_deletion_upstream
large_deletion_at_cut
guide_in_centre_of_large_deletion
very_large_deletion
offtarget1
offtarget2
offtarget3
offtarget6
offtarget4
offtarget5
offtarget7
offtarget8
offtarget9
offtarget10
50
100
150
200
A
A
ACT
AG
AGACA
ATA
ATAT
ATATATATAT
ATGATG
ATGCATCGAT
ATGTG
C
C
CA
CAAACTCCACAC
CAACACG
CCAGCCGAGACAT
CGGAGCACCCGGA
CT
CTACCACACAT
G
G
G
GC
GTGTGTG
GTGTGTGT
GTGTTTG
GTGTGTGTGTGTGTGATG
GTTT
GTTT
T
TATAGCGCATGTGTTTGG
TCTTTCTATTA
TGA
TGC

CrispRVariants results on synthetic data set 1

For showing how CrispRVariants behaves on the synthetic data set, we wish to see how individual reads are treated, including reads that were filtered out, i.e. do not occur in the CrisprSet object.

In other words, here we will add extra alleles to plotAlignments, create a custom heatmap, and combine the plots as in CrispRVariants::plotVariants.

The following code creates an alignment plot with an extra blank row.

```
library("BSgenome.Drerio.UCSC.danRer7")
library("rtracklayer")

# Load the danRer7 / Zv9 genome
danRer7 <- BSgenome.Drerio.UCSC.danRer7

test_bam <- system.file("extdata", "cntnap2b_test_data_s.bam",
                        package = "CrispRVariants")
test_guide <- system.file("extdata", "cntnap2b_test_data_guide.bed",
                        package = "CrispRVariants")

guide <- rtracklayer::import(test_guide)
guide <- guide + 5
reference <- getSeq(danRer7, guide)[[1]]

cset <- CrispRVariants::readsToTarget(test_bam, guide,
                reference = reference, target.loc = 22,
                verbose = FALSE)

# Get the alignments
alns <- cset$crispr_runs[[1]]$alns

# Check that there are no chimeric alignments
! isTRUE(cset$crispr_runs[[1]]$chimeras)
```

```
## [1] TRUE
```

```
# Get the sequences and cigar labels
plot_seqs <- mcols(alns)$seq
cig_labels <- cset$crispr_runs[[1]]$cigar_labels
names(plot_seqs) <- cig_labels

# Get consensus sequences
temp <- split(plot_seqs, factor(cig_labels))
temp_cigs <- split(cigar(alns), factor(cig_labels))
temp_cigs <- sapply(temp_cigs, unique)
temp <- DNAStringSet(unlist(sapply(temp, consensusString)))
starts <- sapply(split(start(alns), factor(cig_labels)), unique)

# The (hidden) CrispRVariants function seqsToAln trims a set of
# Biostrings::DNAString objects with corresponding cigar strings
# and introduces gaps where appropriate.  This differs from
# GenomicAlignments::sequenceLayer in that gaps wider than the target
# region are trimmed.
```

```
plot_seqs <- CrispRVariants:::seqsToAln(temp_cigs[names(plot_seqs)],
                      plot_seqs, guide, aln_start = starts[names(plot_seqs)])

# Add in an extra row for showing the sequences that were filtered out
plot_seqs <- plot_seqs[rownames(variantCounts(cset))]
plot_seqs["Excluded"] <- paste0(rep(" ",nchar(plot_seqs[[1]])), collapse = "")

# Get the insertion site locations
ins <- cset$insertion_sites

# Manually set the x-tick locations for plotAlignments
genomic_coords <- c(start(cset$target):end(cset$target))
target_coords <- cset$genome_to_target[as.character(genomic_coords)]
xbreaks = which(target_coords %% 5 == 0 | abs(target_coords) == 1)
target_coords <- target_coords[xbreaks]

# Make the alignment plot
p <- plotAlignments(reference, alns = plot_seqs, ins.sites = ins, target.loc = 22,
    legend.cols = 4, xtick.labs = target_coords, xtick.breaks = xbreaks)
p <- p + theme(plot.margin = grid::unit(c(0.1,0,0.5,0.2), "lines"))
```

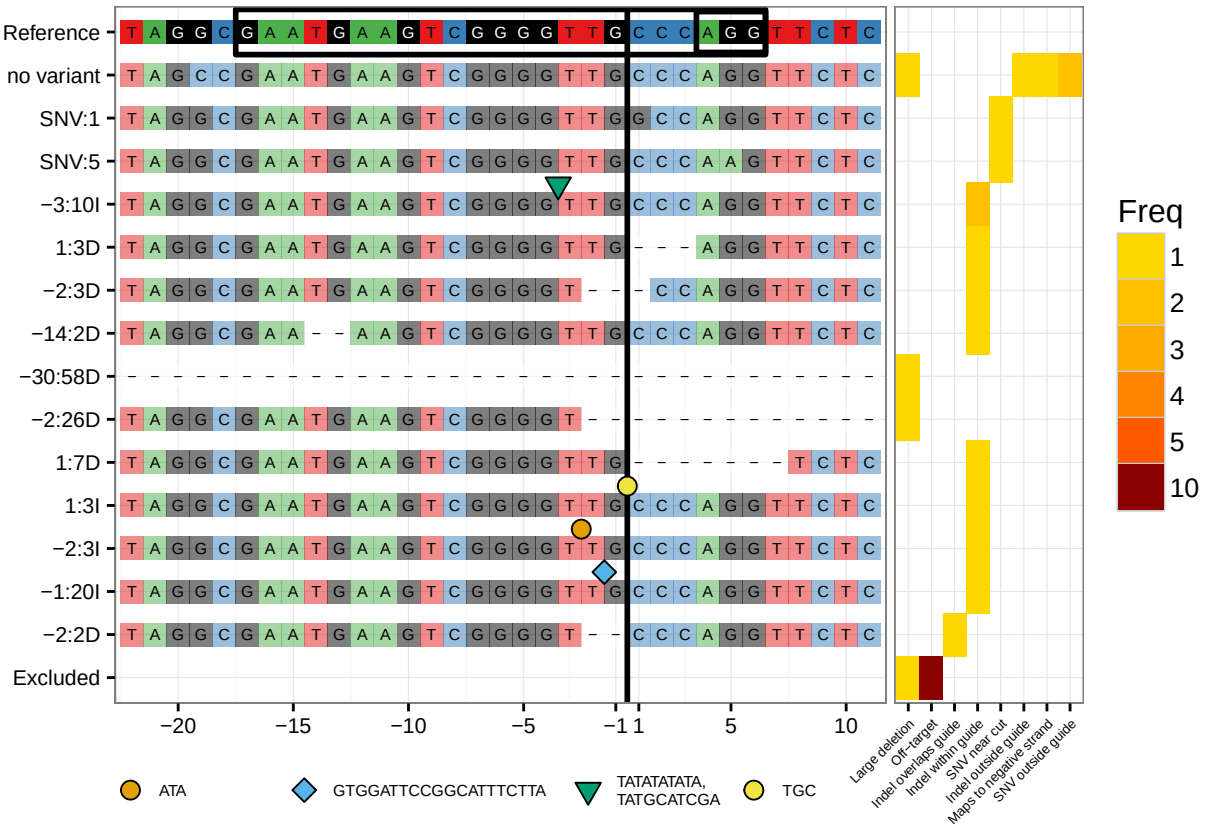
We now create a customised heatmap with alleles matching the alignment plot.

```
# Categories reads by the type of mutation they contain
heatmap_nms <- c("Off-target","Indel within guide","Indel overlaps guide",
                 "Indel outside guide", "SNV near cut", "SNV outside guide",
                 "Large deletion", "Maps to negative strand")

name_to_category <- c("cntnap2b_offtarget1" = "Off-target",
    "cntnap2b_offtarget_9" = "Off-target",
    "cntnap2b_offtarget_10" = "Off-target",
    "cntnap2b_offtarget2" = "Off-target",
    "cntnap2b_offtarget3" = "Off-target",
    "cntnap2b_offtarget6" = "Off-target",
    "cntnap2b_offtarget4" = "Off-target",
    "cntnap2b_offtarget5" = "Off-target",
    "cntnap2b_offtarget_7" = "Off-target",
    "cntnap2b_offtarget_8" = "Off-target",
    "cntnap2b_amplicon_w_snps" = "SNV outside guide",
    "cntnap2b_3bp_del_pos1" = "Indel within guide",
    "cntnap2b_3bp_del_pos-2" = "Indel within guide",
    "cntnap2b_2bp_del_outside_guide" = "Indel outside guide",
    "cntnap2b_2bp_del_in_guide_away_from_cut" = "Indel within guide",
    "cntnap2b_mismatch_pos_1" = "SNV near cut",
    "cntnap2b_mismatch_pos2_of_PAM" = "SNV near cut",
    "cntnap2b_guide_in_centre_of_large_deletion" = "Large deletion",
    "cntnap2b_large_deletion_at_cut" = "Large deletion",
    "cntnap2b_large_deletion_upstream" = "Large deletion",
    "cntnap2b_7bp_deletion_pos1" = "Indel within guide",
    "cntnap2b_mismatches_outside_guide" = "SNV outside guide",
    "cntnap2b_very_large_deletion" = "Large deletion",
    "cntnap2b_3bp_insertion_pos1" = "Indel within guide",
    "cntnap2b_3bp_insertion_pos-2" = "Indel within guide",
```

```
    "cntnap2b_10bp_insertion_pos-2_1" = "Indel within guide",
    "cntnap2b_10bp_insertion_pos-2_2" = "Indel within guide",
    "cntnap2b_20bp_ins_pos-1" = "Indel within guide",
    "cntnap2b_1_bp_del_offtarget_2_bp_del_in_guide",
    "cntnap2b_amplicon_rc" = "Maps to negative strand",
    "cntnap2b_1_bp_del_offtarget_2_bp_del_in_guide" = "Indel overlaps guide")

# Plot seqs are ordered from top to bottom in alignment plot
pt_nms <- rev(names(plot_seqs))
aln_nms <- names(alns)

results <- lapply(seq_along(pt_nms), function(i){
  # For every row in the plot
  # classify alignments belonging to that row
  cig <- pt_nms[[i]]
  if (cig == "Excluded"){
    cats <- name_to_category[setdiff(names(name_to_category), aln_nms)]
  } else {
    # Which read names match this cigar label
    a_nms <- aln_nms[cig_labels == cig]
    # Get classification for these reads
    cats <- name_to_category[a_nms]
  }
  result <- data.frame(table(cats))
  result$y <- i
  result
})


results <- do.call(rbind, results)
# Add space to align to alignment plot
results$cats <- as.factor(results$cats)
results$y <- factor(results$y, levels = c(1:(max(results$y) + 1)))

# This colour palette is a smaller version of the palette used by CrispRVariants
hmcols<-colorRampPalette(c("gold","orange","orangered","red", "darkred"))(10)

q <- ggplot(results, aes(x= cats, y = y)) + geom_tile(aes(fill = Freq)) +
     theme_bw() + xlab(NULL) +  ylab(NULL) +
     scale_y_discrete(drop=FALSE) +
     scale_fill_gradientn(colours = hmcols, na.value = "white",
                          guide = "legend", breaks = c(1,2,3,4,5,10)) +
     theme(axis.text.y = element_blank(),
           axis.ticks.y = element_blank(),
           axis.text.x = element_text(angle = 45, hjust = 1, size = 5),
           plot.background=element_rect(fill = "transparent", colour = NA),
           plot.margin = grid::unit(c(1, 0.25, 0.5, 0), "lines"))

# Arrange the two plots together, so that the y-axes are equal
p2 <- ggplot2::ggplotGrob(p)
```

```
## Warning: Removed 33 rows containing missing values (geom_text).
```

```
p3 <- ggplot2::ggplotGrob(q)
p3$heights <- p2$heights

x <- gridExtra::grid.arrange(gridExtra::arrangeGrob(p2, p3, ncol = 2, widths = c(5,2)),
       newpage = FALSE)
```



Synthetic data set 2

We start with three subdirectories in the `simulation` folder: `crispresso`, `amplicondivider`, and `merged`.

Amplicon sequences were simulated by running the `simulate_mutations.R` script, then fastq reads were simulated from the amplicons by running `simulation_commands.sh`.

Analysis with *CRISPResso* was performed by running `crispresso_simulation_commands.sh`, and analysis with *AmpliconDIVider* first by merging with "'`then by running`amplicondivider_simulation_commands.`Commands for *CRISPRessoPooled* were generated by running`crispresso_pooled_simulation_commands.R`and run by sourcing the output file`crispresso_pooled_commands.sh"'.

For *CrispRVariants*, reads were mapped using *bwa mem* by running the `map_simulated.sh` script. Variant quantification with *CrispRVaraints* is detailed below.

```
library(BSgenome.Drerio.UCSC.danRer7)
library(CrispRVariants)
library(ggplot2)
library(reshape2)
library(rtracklayer)
```

```
danRer7 <- BSgenome.Drerio.UCSC.danRer7

guides <- import("../annotation/shah_guides.bed")
guides <- guides + 5

references <- getSeq(danRer7, guides)


parseCRISPResso <- function(results_dir){
  results_f <- file.path(results_dir, "Quantification_of_editing_frequency.txt")
  system(paste0("echo '\n' >>", results_f))

  f <- file(results_f)
  lns <- readLines(f)
  close(f)

  nhej <- lns[grep(".* NHEJ:", lns)]
  total <- lns[grep("Total", lns)]
  counts <- as.numeric(gsub(".*:([0-9]+)\ .*", "\\1", c(nhej, total)))
  result <- counts[1]/counts[2]*100
  c(result, counts[2])
}

# Parse CRISPResso pooled results
pooled_dirs <- list.files("../simulation/merged", recursive = TRUE,
                          pattern = "CRISPResso_on", include.dirs = TRUE,
                          full.names = TRUE)
condition <- gsub(".*CRISPResso_on_", "", pooled_dirs)
pooled_results <- sapply(pooled_dirs, function(x) parseCRISPResso(x)[1])
noff <- gsub(".*_([0-9]+)offtarget.*", "\\1", pooled_dirs)
nmut <- as.numeric(gsub(".*_([0-9]+)mut.*", "\\1", pooled_dirs))
nwt <- as.numeric(gsub(".*_([0-9]+)wt.*", "\\1", pooled_dirs))
pooled_results <- data.frame(Guide = condition, Truth = nmut/(nmut+nwt) * 100,
                             NOfftargets = noff, variable = "CRISPRessoPooled",
                             value = unname(pooled_results))

# Parse ampliconDIVider results
adiv_files <- list.files("../simulation/amplicondivider", full.names = TRUE)
adiv_results <- sapply(adiv_files, function(fn){
  tt <- read.table(fn, sep = "\t")[1,c(6)]*100
})
exclude <- adiv_results > 100
adiv_results <- adiv_results[!exclude]
print(sprintf("excluded %s", length(exclude)))
```

```
## [1] "excluded 240"
```

```
adiv_gd <- gsub("_.*","", basename(adiv_files)[!exclude])
adiv_noff <- gsub(".*_([0-9]+)offtarget.*", "\\1", adiv_files[!exclude])
adiv_nmut <- as.numeric(gsub(".*_([0-9]+)mut.*", "\\1", adiv_files[!exclude]))
adiv_nwt <- as.numeric(gsub(".*_([0-9]+)wt.*", "\\1",  adiv_files[!exclude]))
adiv_results <- data.frame(Guide = adiv_gd,
                           Truth = adiv_nmut/(adiv_nmut+adiv_nwt)*100,
```

```
                           NOfftargets = adiv_noff,
                           variable = "AmpliconDIVider",
                           value = unname(adiv_results))


# CrispRVariants and CRISPResso
base <- gsub(".fa|.gz","", list.files("../simulation", pattern = "*.fa"))
bams <- file.path("~/scratch", paste0(base, ".bam"))
noff <- as.integer(gsub(".*wt_([0-9]+)offtarget.*", "\\1", base))
frag_len <- as.integer(gsub(".*offtarget_([0-9]+)readlen.*", "\\1", base))
sim_guides <- gsub("(.*)_[0-9]+mut.*","\\1", base)
nmut <- as.numeric(gsub(".*_([0-9]+)mut.*", "\\1", base))
nwt <- as.numeric(gsub(".*_([0-9]+)wt.*", "\\1", base))
truth <-  nmut/(nmut+nwt) * 100

crispresso_dirs <- list.files("../simulation/crispresso", full.names = TRUE)

# Check they are in the same order
identical(unname(sapply(base, function(bb) grep(bb, crispresso_dirs))),c(1:length(base)))
```

```
## [1] TRUE
```

```
result <- lapply(seq_along(base), function(i){
            print(i)
            sim_guide <- sim_guides[i]
            guide <- guides[guides$name == sim_guide]
            reference <- references[guides$name == sim_guide][[1]]
            cset <- readsToTarget(bams[i], target.loc = 22,
                target = guide, reference = reference, collapse.pairs = TRUE,
                verbose = FALSE)
            crispresso <- parseCRISPResso(crispresso_dirs[i])
            c(sim_guide, noff[i], truth[i],
              mutationEfficiency(cset)[["Average"]], crispresso[[1]])
        })
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
```

```
## [1] 19
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
## [1] 25
## [1] 26
## [1] 27
## [1] 28
## [1] 29
## [1] 30
## [1] 31
## [1] 32
## [1] 33
## [1] 34
## [1] 35
## [1] 36
## [1] 37
## [1] 38
## [1] 39
## [1] 40
## [1] 41
## [1] 42
## [1] 43
## [1] 44
## [1] 45
## [1] 46
## [1] 47
## [1] 48
## [1] 49
## [1] 50
## [1] 51
## [1] 52
## [1] 53
## [1] 54
## [1] 55
## [1] 56
## [1] 57
## [1] 58
## [1] 59
## [1] 60
## [1] 61
## [1] 62
## [1] 63
## [1] 64
## [1] 65
## [1] 66
## [1] 67
## [1] 68
## [1] 69
## [1] 70
## [1] 71
## [1] 72
```

```
## [1] 73
## [1] 74
## [1] 75
## [1] 76
## [1] 77
## [1] 78
## [1] 79
## [1] 80
## [1] 81
## [1] 82
## [1] 83
## [1] 84
## [1] 85
## [1] 86
## [1] 87
## [1] 88
## [1] 89
## [1] 90
## [1] 91
## [1] 92
## [1] 93
## [1] 94
## [1] 95
## [1] 96
## [1] 97
## [1] 98
## [1] 99
## [1] 100
## [1] 101
## [1] 102
## [1] 103
## [1] 104
## [1] 105
## [1] 106
## [1] 107
## [1] 108
## [1] 109
## [1] 110
## [1] 111
## [1] 112
## [1] 113
## [1] 114
## [1] 115
## [1] 116
## [1] 117
## [1] 118
## [1] 119
## [1] 120
## [1] 121
## [1] 122
## [1] 123
## [1] 124
## [1] 125
## [1] 126
```

```
## [1] 127
## [1] 128
## [1] 129
## [1] 130
## [1] 131
## [1] 132
## [1] 133
## [1] 134
## [1] 135
## [1] 136
## [1] 137
## [1] 138
## [1] 139
## [1] 140
## [1] 141
## [1] 142
## [1] 143
## [1] 144
## [1] 145
## [1] 146
## [1] 147
## [1] 148
## [1] 149
## [1] 150
## [1] 151
## [1] 152
## [1] 153
## [1] 154
## [1] 155
## [1] 156
## [1] 157
## [1] 158
## [1] 159
## [1] 160
## [1] 161
## [1] 162
## [1] 163
## [1] 164
## [1] 165
## [1] 166
## [1] 167
## [1] 168
## [1] 169
## [1] 170
## [1] 171
## [1] 172
## [1] 173
## [1] 174
## [1] 175
## [1] 176
## [1] 177
## [1] 178
## [1] 179
## [1] 180
```

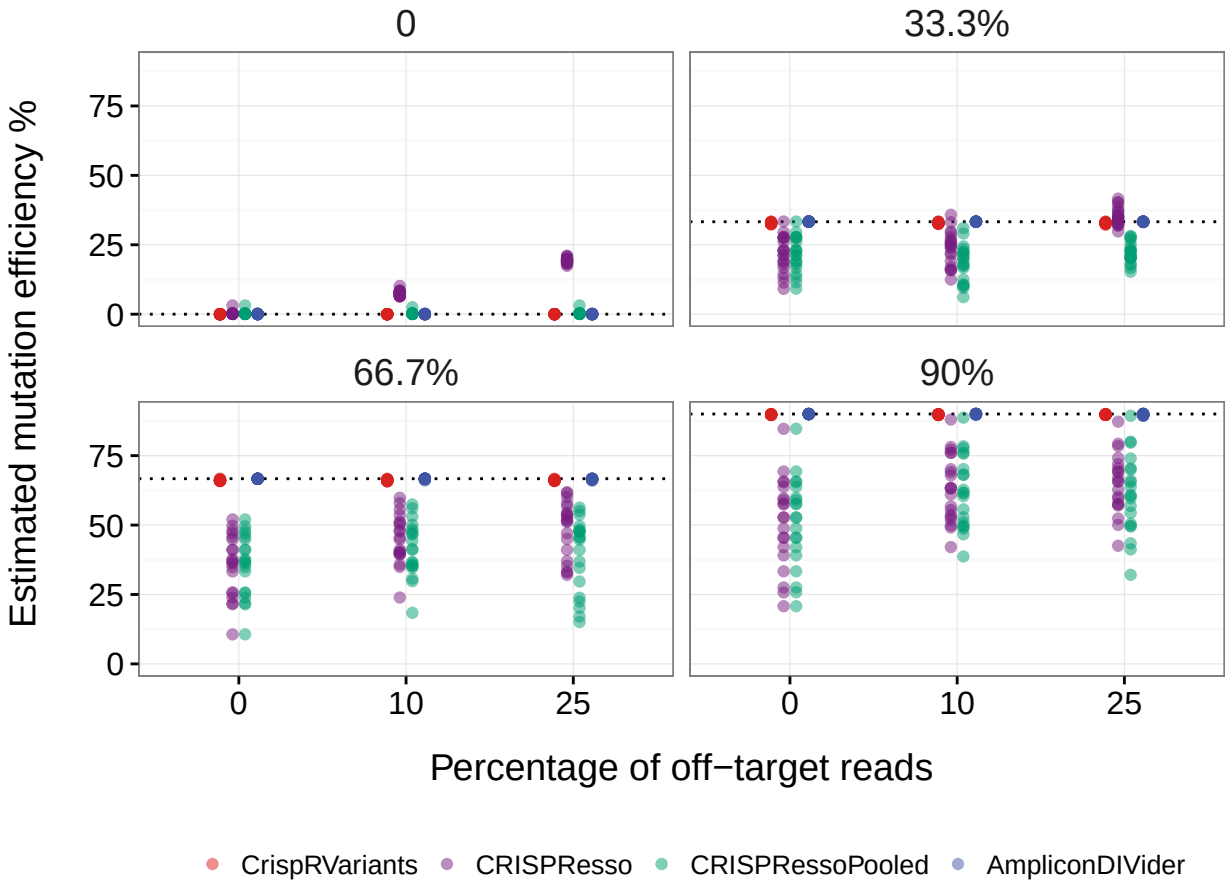
```
## [1] 181
## [1] 182
## [1] 183
## [1] 184
## [1] 185
## [1] 186
## [1] 187
## [1] 188
## [1] 189
## [1] 190
## [1] 191
## [1] 192
## [1] 193
## [1] 194
## [1] 195
## [1] 196
## [1] 197
## [1] 198
## [1] 199
## [1] 200
## [1] 201
## [1] 202
## [1] 203
## [1] 204
## [1] 205
## [1] 206
## [1] 207
## [1] 208
## [1] 209
## [1] 210
## [1] 211
## [1] 212
## [1] 213
## [1] 214
## [1] 215
## [1] 216
## [1] 217
## [1] 218
## [1] 219
## [1] 220
## [1] 221
## [1] 222
## [1] 223
## [1] 224
## [1] 225
## [1] 226
## [1] 227
## [1] 228
## [1] 229
## [1] 230
## [1] 231
## [1] 232
## [1] 233
## [1] 234
```

```
## [1] 235
## [1] 236
## [1] 237
## [1] 238
## [1] 239
## [1] 240
```

```
result <- data.frame(do.call(rbind, result))
colnames(result) <- c("Guide", "NOfftargets", "Truth", "CrispRVariants", "CRISPResso")
result <- melt(result, id.vars=c("Guide", "Truth", "NOfftargets"))
```

```
## Warning: attributes are not identical across measure variables; they will
## be dropped
```

```
result <- rbind(result, pooled_results, adiv_results)
result$NOfftargets <- factor(result$NOfftargets, levels = c(0, 33, 100))
levels(result$NOfftargets) <- c("0", "10", "25")
class(result$value) <- "numeric"
class(result$Truth) <- "factor"
truths <- c("0","33.3%", "66.7%", "90%")
levels(result$Truth) <- truths
levels(result$variable) <- c("CrispRVariants", "CRISPResso",
                             "CRISPRessoPooled", "AmpliconDIVider")
cols <- c("#D92120","#781C81","#009E73","#3F56A7")

tr <- data.frame(Truth = levels(result$Truth),
                 TrNum = as.numeric(gsub("%", "", levels(result$Truth))))

ggplot(result) +
       geom_hline(data = tr, aes(yintercept = TrNum), linetype = "dotted") +
       facet_wrap(~Truth, nrow = 2) +
       geom_point(aes(x=NOfftargets, y=value, colour=variable),
                  alpha = 0.5, position = position_dodge(width = 0.3)) +

       theme_bw() + xlab("Percentage of off-target reads") +
       ylab("Estimated mutation efficiency %") +
       theme(axis.text = element_text(size = 12),
             axis.title.y = element_text(margin = margin(0,20,0,0), size = 14),
             axis.title.x = element_text(margin = margin(15,0,10,0), size = 14),
             strip.text.x = element_text(size = 14),
             legend.key = element_blank(),
             legend.title = element_blank(),
             legend.position = "bottom",
             strip.background =  element_blank()) +
       scale_colour_manual(values = cols)
```



Supplementary Note 2: Sequencing errors and alignment uncertainty affect

variant count, placement and size

Analyses with Burger *et al* Sanger data

The Burger *et al* Sanger data is included in the github repository for this manuscript.

In this section, we use a pre-generated transcript database (txdb) to access structures of the transcripts overlapping the guide. The txdb was generated from a gtf file downloaded from Ensembl. For more information about generating a txdb, see the R Bioconductor package GenomicFeatures.

```
library("CrispRVariants")
library("ShortRead")
library("rtracklayer")
library("BSgenome.Drerio.UCSC.danRer7")
library("gdata")
library("GenomicFeatures")
library("grid")

ind <- "../idx/danRer7.fa"
```

```
# Import guides, add 5 bases to each end
gd <- rtracklayer::import("../annotation/Burger_Sanger_guides.bed")
gdl <- resize(gd, width(gd) + 10, fix = "center")
names(gdl) <- gdl$name

# Get the reference sequences from the genome
danRer7 <- BSgenome.Drerio.UCSC.danRer7
refs <- getSeq(danRer7, gdl)
names(refs) <- names(gdl)

txdb <- loadDb("~/zebrafish_txdb.sqlite")

ab1ToFastq <- function(raw, fdir, recall = TRUE){
  # List files ending in .ab1 in the raw directory
  dir.create(fdir, showWarnings = FALSE)
  ab1s <- dir(raw, pattern="ab1$", recursive = TRUE)
  # Get sequence names from filenames
  sns <- gsub(".ab1","",basename(ab1s))

  if (all(dirname(ab1s) == ".")){
    # No subdirectory
    fqs <- paste0(basename(dirname(fastq)), ".fastq")
  } else {
    # Name after subdirectory
    fqs  <- paste0(gsub("[\ |\\/]", "_", dirname(ab1s)), ".fastq")
  }
  fqs <- file.path(fdir, fqs)
  ab1s <- file.path(raw, ab1s)

  # Convert to fastq, optionally recalling bases
  dummy <- mapply(function(u,v,w,rc) {
    print(c(u,v,w))
    suppressWarnings(CrispRVariants::abifToFastq(u,v,w, recall = rc))
  }, sns, ab1s, fqs, recall)

  fqs <- unique(fqs)
  fqs
}


runMapping <- function(fqs, bdir, ind){
  # Map a list of fastq files with bwa mem, sort and index
  dir.create(bdir, showWarnings = FALSE)
  bms <- gsub(".fastq$",".bam",basename(fqs))
  sbms <- file.path(bdir, gsub(".bam","_s",bms))
  for(i in 1:length(fqs)) {
    cmd <- paste0("bwa mem ", ind, " ",
                  fqs[i]," | samtools view -Sb - > ", bms[i])
    system(cmd)
      indexBam(sortBam(bms[i],sbms[i]))
      unlink(bms[i])
  }
  sbms <- unique(paste0(sbms, ".bam"))
```

```
  sbms
}
```

Sequencing errors in *tbx16*

```
raw <- "../Burger_Sanger_data/spt_ccA_F1/ab1"
fastq <- "../Burger_Sanger_data/spt_ccA_F1/fastq"
bam <- "../Burger_Sanger_data/spt_ccA_F1/bam"

fqs <- ab1ToFastq(raw, fastq)
```

```
## [1] "AB1025"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_11_ph/AB1025.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_11_ph.fastq"
## [1] "AB1026"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_11_ph/AB1026.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_11_ph.fastq"
## [1] "AB1027"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_11_ph/AB1027.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_11_ph.fastq"
## [1] "AB1028"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_11_ph/AB1028.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_11_ph.fastq"
## [1] "AB1029"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_11_ph/AB1029.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_11_ph.fastq"
## [1] "AB1030"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_11_ph/AB1030.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_11_ph.fastq"
## [1] "AB1031"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_11_ph/AB1031.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_11_ph.fastq"
## [1] "AB1032"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_11_ph/AB1032.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_11_ph.fastq"
## [1] "AB1033"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_12_ph/AB1033.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_12_ph.fastq"
## [1] "AB1034"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_12_ph/AB1034.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_12_ph.fastq"
## [1] "AB1035"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_12_ph/AB1035.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_12_ph.fastq"
## [1] "AB1036"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_12_ph/AB1036.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_12_ph.fastq"
## [1] "AB1037"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_12_ph/AB1037.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_12_ph.fastq"
## [1] "AB1038"
```

```
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_12_ph/AB1038.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_12_ph.fastq"
## [1] "AB1039"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_12_ph/AB1039.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_12_ph.fastq"
## [1] "AB1040"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_12_ph/AB1040.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_12_ph.fastq"
## [1] "AB1049"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_14_ph/AB1049.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph.fastq"
## [1] "AB1050"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_14_ph/AB1050.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph.fastq"
## [1] "AB1051"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_14_ph/AB1051.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph.fastq"
## [1] "AB1052"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_14_ph/AB1052.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph.fastq"
## [1] "AB1053"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_14_ph/AB1053.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph.fastq"
## [1] "AB1054"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_14_ph/AB1054.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph.fastq"
## [1] "AB1055"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_14_ph/AB1055.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph.fastq"
## [1] "AB1056"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_14_ph/AB1056.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph.fastq"
## [1] "AB1057"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_14_ph/embryo 8/AB1057.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph_embryo_8.fastq"
## [1] "AB1058"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_14_ph/embryo 8/AB1058.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph_embryo_8.fastq"
## [1] "AB1059"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_14_ph/embryo 8/AB1059.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph_embryo_8.fastq"
## [1] "AB1060"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_14_ph/embryo 8/AB1060.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph_embryo_8.fastq"
## [1] "AB1061"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_14_ph/embryo 8/AB1061.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph_embryo_8.fastq"
## [1] "AB1062"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_14_ph/embryo 8/AB1062.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph_embryo_8.fastq"
## [1] "AB1063"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_14_ph/embryo 8/AB1063.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph_embryo_8.fastq"
## [1] "AB1064"
```

```
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_14_ph/embryo 8/AB1064.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph_embryo_8.fastq"
## [1] "IM2009"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_5_wt/IM2009.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_5_wt.fastq"
## [1] "IM2010"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_5_wt/IM2010.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_5_wt.fastq"
## [1] "IM2011"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_5_wt/IM2011.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_5_wt.fastq"
## [1] "IM2012"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_5_wt/IM2012.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_5_wt.fastq"
## [1] "IM2013"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_5_wt/IM2013.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_5_wt.fastq"
## [1] "IM2014"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_5_wt/IM2014.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_5_wt.fastq"
## [1] "IM2015"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_5_wt/IM2015.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_5_wt.fastq"
## [1] "IM2016"
## [2] "../Burger_Sanger_data/spt_ccA_F1/ab1/spt_ccA_F1_5_wt/IM2016.ab1"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_5_wt.fastq"
```

```
print(fqs)
```

```
## [1] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_11_ph.fastq"
## [2] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_12_ph.fastq"
## [3] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph.fastq"
## [4] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_14_ph_embryo_8.fastq"
## [5] "../Burger_Sanger_data/spt_ccA_F1/fastq/spt_ccA_F1_5_wt.fastq"
```

```
bms <- runMapping(fqs, bam, ind)
nms <- gsub(".bam", "", gsub("_", " ", basename(bms)))

guide <- gdl["spt_ccA"]
ref <- refs["spt_ccA"]

cset <- readsToTarget(bms, target = guide, reference = ref,
            names = nms, target.loc = 22, verbose = FALSE)

print(cset)
```

```
## CrisprSet object containing 5 CrisprRun samples
## Target location:
## GRanges object with 1 range and 2 metadata columns:
##           seqnames              ranges strand |        name     score
##              <Rle>           <IRanges>  <Rle> | <character> <numeric>
##   spt_ccA     chr8 [54029513, 54029545]      + |     spt_ccA         0
##   -------
```

```
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
## [1] "Most frequent variants:"
##             spt ccA F1 11 ph s spt ccA F1 12 ph s spt ccA F1 14 ph s
## no variant                   0                  0                  0
## -1:29I,4:8I                  2                  2                  4
## -1:1I                        5                  4                  0
## -6:9D,5:3I                   0                  0                  0
## -1:30I,4:8I                  0                  1                  0
## -1:28I,4:8I                  0                  0                  0
##             spt ccA F1 14 ph embryo 8 s spt ccA F1 5 wt s
## no variant                            1                 0
## -1:29I,4:8I                           3                 2
## -1:1I                                 1                 0
## -6:9D,5:3I                            0                 5
## -1:30I,4:8I                           0                 0
## -1:28I,4:8I                           1                 0
```

```
plotVariants(cset,
             plotAlignments.args = list(max.insertion.size = 50,
                                        legend.cols = 2),
             plotFreqHeatmap.args = list(
               legend.key.height = grid::unit(1, "lines")))
```


```
## TableGrob (2 x 1) "arrange": 2 grobs
##   z     cells    name                grob
## 1 1 (1-1,1-1) arrange rect[GRID.rect.3936]
## 2 2 (2-2,1-1) arrange      gtable[arrange]
```

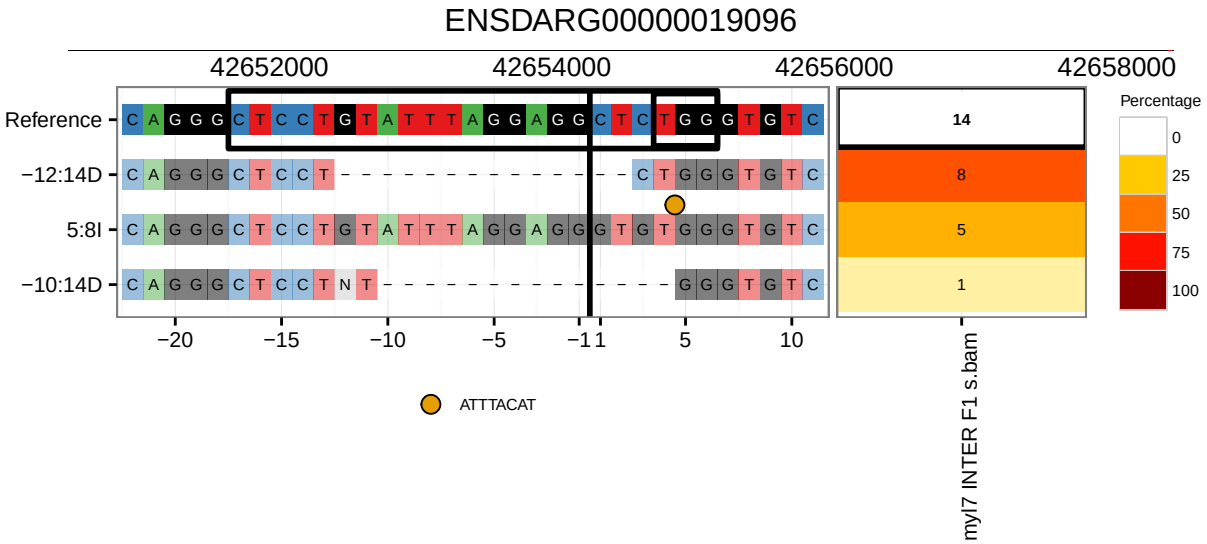
Alignment uncertainty in *myl7*

```
raw <- "../Burger_Sanger_data/myl7_INTER_F1/ab1"
fastq <- "../Burger_Sanger_data/myl7_INTER_F1/fastq"
bam <- "../Burger_Sanger_data/myl7_INTER_F1/bam"
```

```
guide <- gdl["myl7_5"]
ref <- refs["myl7_5"]

ab1ToPlot <- function(raw, fastq, bam, guide, ref, recall){
  fqs <- ab1ToFastq(raw, fastq, recall = recall)
  bms <- runMapping(fqs, bam, ind)
  nms <- gsub("_s.bam", "", gsub("_", " ", basename(bms)))

  cset <- readsToTarget(bms, target = guide, reference = ref,
            names = nms, target.loc = 22, verbose = FALSE)

  print(cset)
  plotVariants(cset, txdb = txdb)

  # Remove the fastq and bam files so the reads are
  # not duplicated at the next function call
  unlink(fastq, recursive = TRUE)
  unlink(bam, recursive = TRUE)
}

# First extract ab1 files without base recalibration
dummy <- ab1ToPlot(raw, fastq, bam, guide, ref, recall = FALSE)
```

```
## [1] "CH141"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH141.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH142"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH142.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH143"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH143.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH144"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH144.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH146"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH146.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH147"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH147.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH148"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH148.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH149"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH149.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH150"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH150.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH151"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH151.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
```

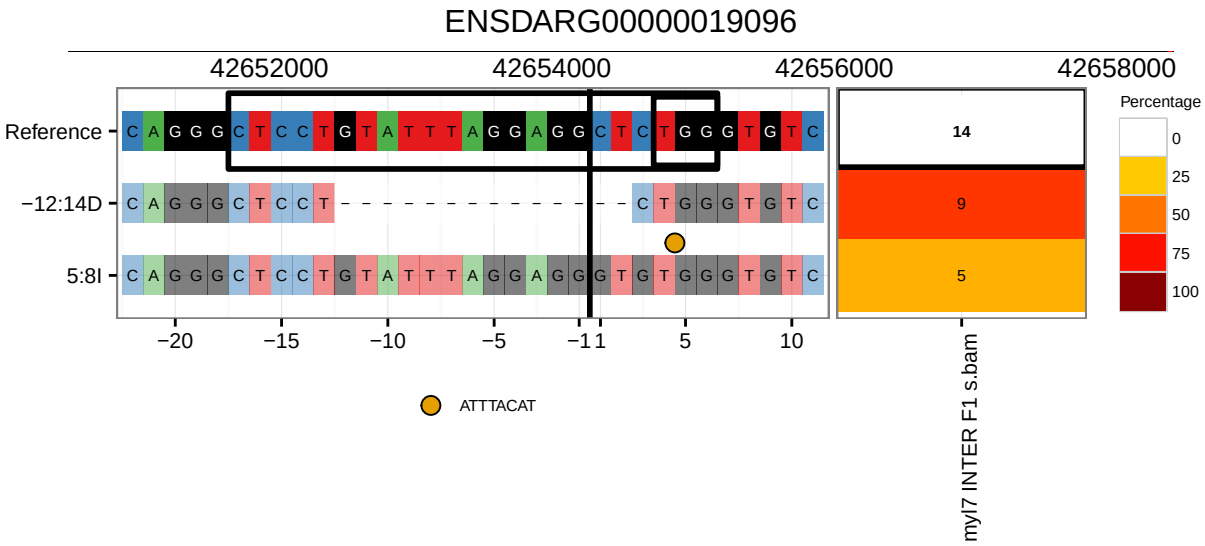
```
## [1] "CH152"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH152.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH154"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH154.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH155"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH155.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH156"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH156.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## CrisprSet object containing 1 CrisprRun samples
## Target location:
## GRanges object with 1 range and 2 metadata columns:
##          seqnames              ranges strand |        name     score
##             <Rle>           <IRanges>  <Rle> | <character> <numeric>
##   myl7_5     chr8 [42658358, 42658390]      - |      myl7_5         0
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
## [1] "Most frequent variants:"
##         myl7 INTER F1 s.bam
## -12:14D                   8
## 5:8I                      5
## -10:14D                   1

## 'select()' returned 1:many mapping between keys and columns
## 'select()' returned 1:many mapping between keys and columns
```




```
# Then with recalibration
dummy <- ab1ToPlot(raw, fastq, bam, guide, ref, recall = TRUE)
```

```
## [1] "CH141"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH141.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
```

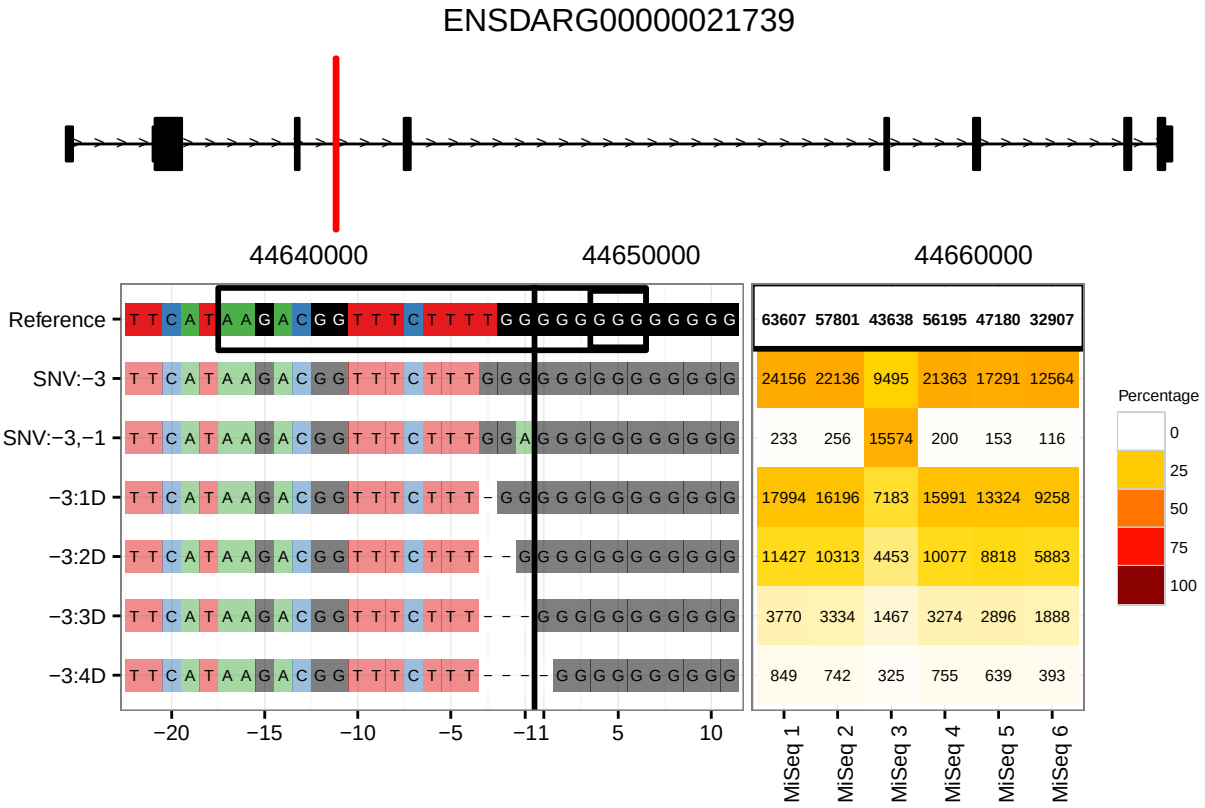
```
## [1] "CH142"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH142.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH143"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH143.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH144"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH144.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH146"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH146.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH147"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH147.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH148"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH148.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH149"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH149.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH150"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH150.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH151"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH151.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH152"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH152.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH154"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH154.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH155"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH155.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## [1] "CH156"
## [2] "../Burger_Sanger_data/myl7_INTER_F1/ab1/CH156.ab1"
## [3] "../Burger_Sanger_data/myl7_INTER_F1/fastq/myl7_INTER_F1.fastq"
## CrisprSet object containing 1 CrisprRun samples
## Target location:
## GRanges object with 1 range and 2 metadata columns:
##          seqnames              ranges strand |        name     score
##             <Rle>           <IRanges>  <Rle> | <character> <numeric>
##   myl7_5     chr8 [42658358, 42658390]      - |      myl7_5         0
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
## [1] "Most frequent variants:"
##         myl7 INTER F1 s.bam
## -12:14D                   9
## 5:8I                      5

## 'select()' returned 1:many mapping between keys and columns
## 'select()' returned 1:many mapping between keys and columns
```



Analyses with Burger *et al* MiSeq data

The Burger *et al* MiSeq data is available from ArrayExpress with accession number E-MTAB-4143. Here we assume that the bam files have been download into a directory named "Burger_MiSeq_data". These are paired-end 250bp reads.

```
bam_dir <- "../Burger_MiSeq_data"
bams <- list.files(bam_dir, pattern = "*.bam$", full.names = TRUE)
target <- rtracklayer::import("../annotation/Burger_MiSeq_guides.bed")
target <- target[target$name == "xirp1_off2"]
target <- target + 5
reference <- getSeq(danRer7, target)
# Note: getSeq returns a DNAStringSet, we must select the first
# sequence to transform it into a DNAString

sample_names <- paste("MiSeq", c(1:6))

# Set collapse.pairs = TRUE to count pairs correctly
cset <- readsToTarget(bams, target = target, reference = reference[[1]],
                      target.loc = 22, names = sample_names,
                      collapse.pairs = TRUE, verbose = FALSE)
plotVariants(cset, txdb = txdb, row.ht.ratio = c(1,2), col.wdth.ratio = c(1.5,1),
             plotAlignments.args = list(min.freq = 1),
             plotFreqHeatmap.args = list(min.freq = 1))
```




```
## TableGrob (2 x 1) "arrange": 2 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange  gtable[layout]
## 2 2 (2-2,1-1) arrange gtable[arrange]
```

Supplementary Note 8: Chimeric alignments include genuine variants and sequencing errors

Here we use the data from Burger *et al* and Cho *et al*. The data from Cho *et al* is available from the DNA Data Bank of Japan under accession DRA001195. We assume the Cho off-target data (i.e. not exome data) has been downloaded, extracted to fastq and mapped, and that the mapped bam files are located in a directory named "Cho_data". The Cho data is for human cells. We have reformatted the metadata describing the Cho samples (Supplementary Table 2 intheir paper) and include this in the annotation folder.

Functions used in chimera analyses

read_alns

This is a wrapper for GenomicAlignments::readGAlignments setting some parameters

```
read_alns <- function(fname){
  readGAlignments(fname, param = ScanBamParam(what = c("seq", "flag")),
     use.names = TRUE)
}
```

get_chimeras

```
get_chimeras <- function(alns, guides, expand.guide = 20){
  # Get the chimeric reads
  ch_idxs <- CrispRVariants::findChimeras(alns, by.flag = TRUE)
  ch <- alns[ch_idxs]

  # Check that the chimeric region is near the guide
  ex_gd <- guides + expand.guide
  # Exclude reads where the guide + some flanking sequence is contained
  # within a chimeric segment
  guide_within <- subjectHits(findOverlaps(ex_gd, ch,
                              ignore.strand = TRUE, type = "within"))

  ordrd <- unique(guide_within[order(guide_within)])
  non_ch <- names(ch) %in% names(ch)[ordrd]
  ch_idxs <- ch_idxs[!non_ch]
  ch <- alns[ch_idxs]

  # Get combinations of cigar strings for chimeric alignment sets
  partition <- cumsum(rle(names(ch))$lengths)
  cigs <- relist(cigar(ch), PartitioningByEnd(partition))
  cigs <- lapply(cigs, paste, collapse = ",")
  uniq_combs <- ! duplicated(cigs)

  # Return one read set per chimera combination
  ch <- unlist(relist(ch, PartitioningByEnd(partition))[as.vector(uniq_combs)])
  ch
}
```

select_chimeras

This function filters candidate chimeric reads by maximum mapped range and number of chromosomes spanned (all chimeric segments must map to a single chromosome).

```
select_chimeras <- function(alns){
    partition <- cumsum(rle(names(alns))$lengths)
    alnsl <- relist(alns, PartitioningByEnd(partition))

    # Remove groups spanning multiple chromosomes.
    # This will exclude insertions mapped to another chr, these are rare
    sq_nms <- unique(seqnames(alnsl))
    keep <- elementLengths(sq_nms) == 1
    alnsl <- alnsl[keep]

    singlechr_len <- length(unlist(alnsl))
    sq_nms <- sq_nms[keep]

    # Get ranges
    gp_rngs <- GRanges(as.character(sq_nms), IRanges(min(start(alnsl)), max(end(alnsl))))

    # Remove alns spanning more than 1000, these are probably pcr errors
```

```
    keep <- width(gp_rngs) <= 1000
    alnsl <- alnsl[keep]
    gp_rngs <- gp_rngs[keep]
    result <- list(alns = alnsl, gp_rngs = gp_rngs)
    result
}
```

dist_to_cut

Finds the minimum distance between any member of a chimeric read set and a CRISPR cut site.

```
dist_to_cut <- function(alns_list, cut_sites){
    # Accepts a list of alignments split by chimeric group
    all_alns <- as(unlist(alns_list), "GRanges")
    dists <- distanceToNearest(all_alns, cut_sites, ignore.strand = TRUE)
    dists <- min(relist(mcols(dists)$distance, alns_list))
    as.vector(dists)
}
```

make_violin_plot

This makes violin plots showing the distibution of on- and off-target chimeric reads.

```
make_violin_plot <- function(dat){
    p <- ggplot(dat, aes(x = cond, y = dist)) + geom_violin(fill = "gray") +
        theme_bw() + ylab("Distance to nearest cut site (bases)") +
        theme(axis.title.x=element_blank())
    return(p)
}
```

mappedRangeByTarget

This function filters chimeric read alignment sets by the aligned range. The aligned range must be nearly identical to the PCR primers for the amplicon. This is done to try and avoid PCR chimeras. We only perform this step for the Burger data, as we do not have sufficient information to do the same for the Cho data. Returns a list of chimeras for each PCR range.

```
mappedRangeByTarget <- function(alns, targets, maxgap = 5){
  # Only keep reads where the range of the aligned segments is
  # nearly equal to the primer range.  Note this excludes segments
  # where an insertion maps outside of the amplicon, but endpoints will be retained

  temp <- select_chimeras(alns)
  alnsl <- temp$alns
  gp_rngs <- temp$gp_rngs

  hits <- findOverlaps(gp_rngs, targets + maxgap, type = "within")
  wdth_diff <- abs(width(targets[subjectHits(hits)]) - width(gp_rngs[queryHits(hits)]))
  hits <- hits[wdth_diff <= 2*maxgap]

  result <- vector("list", length(targets))
```

```
  splits <- split(queryHits(hits), subjectHits(hits))

  for (nm in names(splits)){
    result[as.integer(nm)] <- unlist(alnsl[splits[[nm]]])
  }
  if ("name" %in% names(mcols(targets))) names(result) <- targets$name

  return(result)
}
```

getChByTgt

Select chimeras matching each PCR primer range, only used for Burger data.

```
getChByTgt <- function(bams, guides, target_ranges){
    alns_by_tgt <- bplapply(bams, function(bam){
      alns <- read_alns(bam)
      # Cut down size by first selecting chimeric reads that overlap the primer ranges
      # (redundant with filtering in mappedRangeByTarget)
      ch_idxs <- CrispRVariants::findChimeras(alns, by.flag = TRUE)
      alns <- alns[ch_idxs]
      hits_pcr <- names(alns)[queryHits(findOverlaps(alns, target_ranges))]
      alns <- alns[names(alns) %in% hits_pcr]

      # Now exclude guides completely contained within one segment
      alns <- get_chimeras(alns, guides)

      result <- mappedRangeByTarget(alns, targets = target_ranges)
      names(result) <- target_ranges$name
      result
      }, BPPARAM = BiocParallel::MulticoreParam(6))
    alns_by_tgt
}
```

Chimera analysis with Cho data

```
library("BiocParallel")
library("BSgenome.Hsapiens.UCSC.hg19")
library("CrispRVariants")
library("gdata")
library("GenomicRanges")
library("GenomicAlignments")
library("ggplot2")
library("gridExtra")
library("Rsamtools")
library("rtracklayer")

hg19 <- BSgenome.Hsapiens.UCSC.hg19
cho <- gdata::read.xls("../annotation/Cho_Table2_reformatted.xls", header = TRUE)
guides <- GenomicRanges::GRanges(cho$chromosome,
```

```
                                IRanges(cho$start + 1, width = 23), strand = cho$strand)
guide_seqs <- getSeq(hg19, guides)
cho_guides <- as(cho$guide, "DNAStringSet")
all.equal(as.character(guide_seqs), as.character(cho_guides))
```

```
## [1] TRUE
```

```
is_ontarget <- grepl("OnTarget", cho$X)
#cut_sites <- narrow(guides, start = 17, end = 17)
cut_sites <- resize(resize(guides, 6, fix = "end"), 1, fix = "start")

cho_bam_files <- list.files("../Cho_data", pattern = "*.bam", full.names = TRUE)
print(cho_bam_files)
```

```
## [1] "../Cho_data/DRR014240_s.bam" "../Cho_data/DRR014241_s.bam"
## [3] "../Cho_data/DRR014249_s.bam"
```

```
alns <- lapply(cho_bam_files, read_alns)
alns <- lapply(alns, get_chimeras, guides = guides)
alns <- do.call(c, alns)

temp <- select_chimeras(alns)
alnsl <- temp$alns
gp_rngs <- temp$gp_rngs


# Keep alignments that only overlap one guide, divide into on- and off-target
hits <- findOverlaps(gp_rngs, guides)
unq_hits <- ! duplicated(queryHits(hits)) & ! duplicated(queryHits(hits), fromLast = TRUE)

unq_on <- unq_hits & is_ontarget[subjectHits(hits)]
unq_off <- unq_hits & !is_ontarget[subjectHits(hits)]

cho_on <- alnsl[queryHits(hits)[unq_on]]
cho_off <- alnsl[queryHits(hits)[unq_off]]

cho_off_dists <- dist_to_cut(cho_off, cut_sites)
cho_on_dists <- dist_to_cut(cho_on, cut_sites)

cho_dat <- data.frame(dist = c(cho_on_dists, cho_off_dists),
  cond = rep(c("On target","Off target"),
             c(length(cho_on_dists), length(cho_off_dists))))

print(sprintf("On target %s \n Off target %s\n",
              length(cho_on_dists), length(cho_off_dists)))
```

```
## [1] "On target 1142 \n Off target 158\n"
```

```
p1 <- make_violin_plot(cho_dat)
p1 <- p1 + ggtitle("Cho")
```

Chimera analysis with Burger data

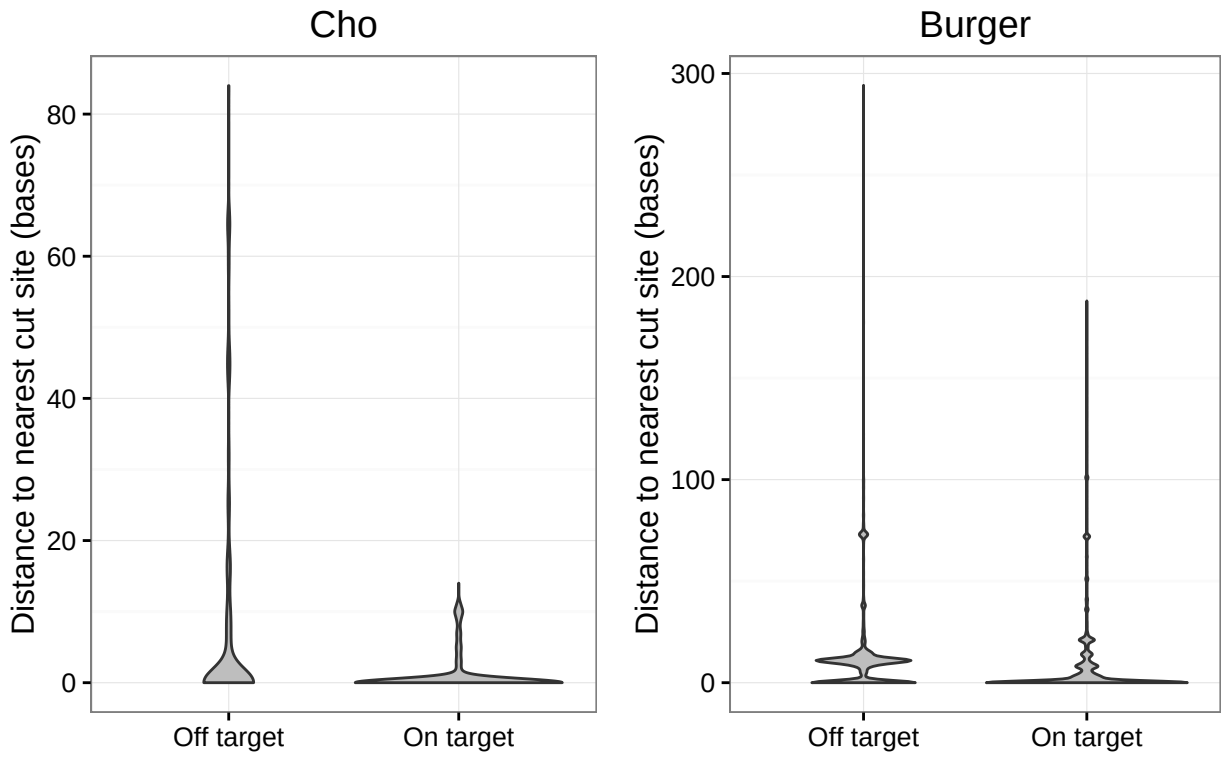
```
bam_dir <- "../Burger_MiSeq_data"
bams <- list.files(bam_dir, pattern = "*.bam$", full.names = TRUE)
guides <- rtracklayer::import("../annotation/Burger_MiSeq_guides.bed")
pcr_ranges <- import(file.path("../annotation/Burger_MiSeq_primer_ranges.bed"))
gd_to_primer <- read.table("../annotation/Burger_MiSeq_layout.txt", sep = "\t",
                           header = TRUE)

# In this experiment the lcr guides were multiplexed and the hand2 guides share
# PCR primers, so we'll remove these.
guides <- guides[!grepl("hand2_ccA$|hand2_ccB$|lcr", guides$name)]
pcr_ranges <- pcr_ranges[!grepl("hand2$|lcr", pcr_ranges$name)]
gd_to_primer <- gd_to_primer[!grepl("hand2$|lcr", gd_to_primer$primer), ]

cut_sites <- narrow(guides, start = 17, end = 17)
is_ontarget <- ! grepl("off", guides$name)

ch_by_primer <- getChByTgt(bams, guides, pcr_ranges)

ch_by_gd <- apply(gd_to_primer, 1, function(rw){
  fn <- rw["primer"]
  gd <- rw[["guide"]]
  gidx <- guides$name == gd
  guide <- guides[gidx]
  cut_site <- cut_sites[gidx]
  idxs <- as.numeric(strsplit(rw["samples"], ",")[[1]])
  # Pull out chimeras for this guide for the correct samples
  all_alns <- lapply(idxs, function(idx) ch_by_primer[[idx]][[gd]])
  all_dists <- lapply(all_alns, function(alns, guides){
    if (length(alns) == 0) return(list())
    ch <- get_chimeras(alns, guides)
    ch <- split(ch, names(ch))
    dists <- dist_to_cut(ch, cut_site)
    }, guides = guide)
  dists <- do.call(c, all_dists)
  dists
})

burger_on_target <- unlist(ch_by_gd[is_ontarget], use.names = FALSE)
burger_off_target <- unlist(ch_by_gd[!is_ontarget], use.names = FALSE)

dat <- data.frame(dist = c(burger_on_target, burger_off_target),
cond = rep(c("On target","Off target"), c(length(burger_on_target),
           length(burger_off_target))))

print(sprintf("On target %s \n Off target %s\n", length(burger_on_target),
       length(burger_off_target)))
```

```
## [1] "On target 11386 \n Off target 18252\n"
```

```
p2 <- make_violin_plot(dat)
p2 <- p2 + ggtitle("Burger")
```

Combine chimera plots

```
gridExtra::grid.arrange(p1,p2, ncol=2)
```




```
sessionInfo()
```

```
## R version 3.2.2 (2015-08-14)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 14.04.4 LTS
##
## locale:
##  [1] LC_CTYPE=en_CA.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_CA.UTF-8        LC_COLLATE=en_CA.UTF-8
##  [5] LC_MONETARY=en_CA.UTF-8    LC_MESSAGES=en_CA.UTF-8
##  [7] LC_PAPER=en_CA.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_CA.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
##  [1] grid      stats4    parallel  stats     graphics  grDevices utils
##  [8] datasets  methods   base
##
## other attached packages:
```

```
##  [1] BSgenome.Hsapiens.UCSC.hg19_1.4.0  ShortRead_1.28.0
##  [3] GenomicAlignments_1.6.3            SummarizedExperiment_1.0.2
##  [5] Rsamtools_1.22.0                   scales_0.3.0.9000
##  [7] reshape2_1.4.1                     gridExtra_2.0.0
##  [9] GenomicFeatures_1.22.8             AnnotationDbi_1.32.3
## [11] Biobase_2.30.0                     gdata_2.17.0
## [13] BSgenome.Drerio.UCSC.danRer7_1.4.0 BSgenome_1.38.0
## [15] rtracklayer_1.30.1                 GenomicRanges_1.22.3
## [17] GenomeInfoDb_1.6.2                 Biostrings_2.38.3
## [19] XVector_0.10.0                     IRanges_2.4.6
## [21] S4Vectors_0.8.6                    BiocGenerics_0.16.1
## [23] BiocParallel_1.4.3                 CrispRVariants_0.99.3
## [25] ggplot2_2.0.0.9001
##
## loaded via a namespace (and not attached):
##  [1] gtools_3.5.0         lattice_0.20-33      colorspace_1.2-6
##  [4] htmltools_0.3        yaml_2.1.13          XML_3.98-1.3
##  [7] DBI_0.3.1            RColorBrewer_1.1-2   lambda.r_1.1.7
## [10] plyr_1.8.3           stringr_1.0.0        zlibbioc_1.16.0
## [13] munsell_0.4.3        gtable_0.1.2.9000    futile.logger_1.4.1
## [16] hwriter_1.3.2        evaluate_0.8         labeling_0.3
## [19] latticeExtra_0.6-26  knitr_1.12.3         biomaRt_2.26.1
## [22] httpuv_1.3.3         Rcpp_0.12.2          xtable_1.8-0
## [25] formatR_1.2.1        sangerseqR_1.6.0     mime_0.4
## [28] digest_0.6.9         stringi_1.0-1        shiny_0.13.0
## [31] tools_3.2.2          bitops_1.0-6         magrittr_1.5
## [34] RCurl_1.95-4.7       RSQLite_1.0.0        futile.options_1.0.0
## [37] rmarkdown_0.9.2      R6_2.1.1
```