CrispRVariants Reference Manual

Helen Lindsay

4 February 2016

Contents

About this guide

This guide collects example use cases for CrispRVariants from several studies. For simplicity, we include here only the relevant code snippets for using CrispRVariants and direct users to the individual study repositories for the full data analysis pipelines.

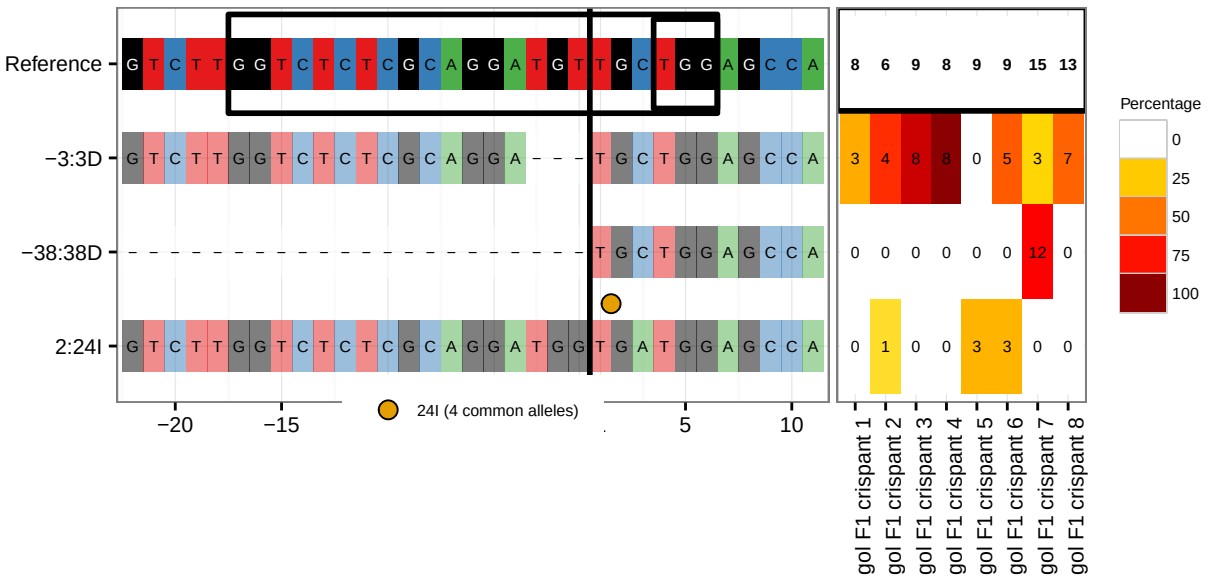
Changing the appearance of plots

Note that arguments for CrispRVariants::plotAlignments described below can be passed to CrispRVariants::plotVariants as a list, e.g. `plotAlignments.args = list(axis.text.size = 14)`. Similarly, arguments for CrispRVariants::plotFreqHeatmap are passed through plotVariants via `plotFreqHeatmap.args`.

Filtering data in plotVariants

The data used in `plotAlignments` and `plotFreqHeatmap` can be filtered by either frequency via `min.freq`, count via `min.count`, or to show a set number of alleles sorted by frequency, via `top.n`. Within `plotVariants`, these filtering options need to be set for both `plotAlignments" and`plotFreqHeatmap"'. We also add space to the bottom of the plot to prevent clipping of the labels.

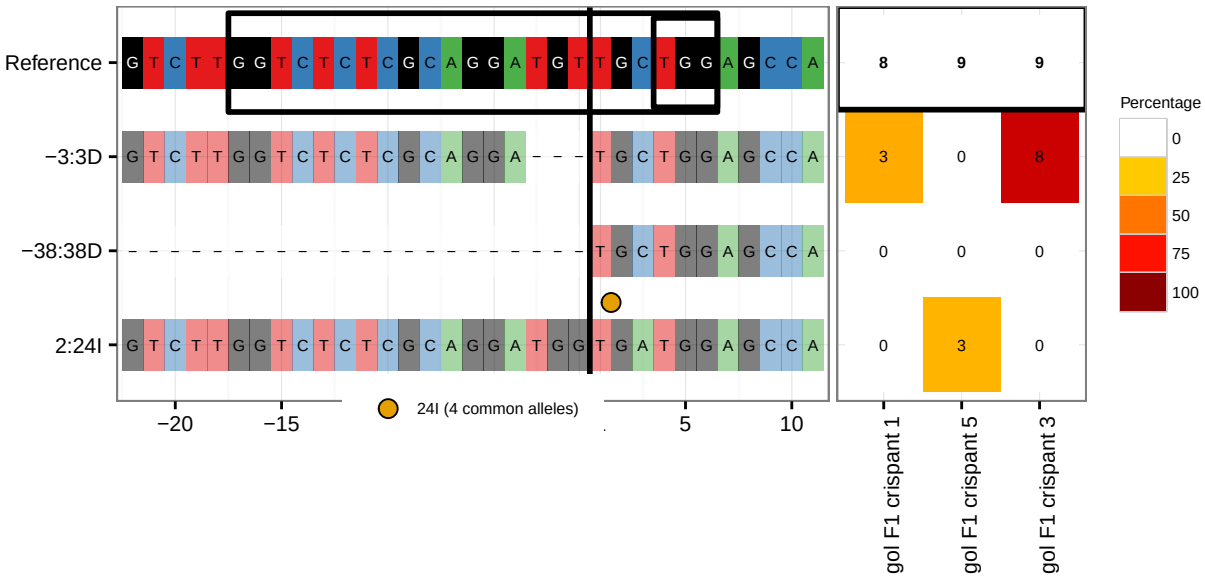
```
library(GenomicFeatures)
plotVariants(gol, plotAlignments.args = list(top.n = 3),
             plotFreqHeatmap.args = list(top.n = 3),
             left.plot.margin = ggplot2::unit(c(0.1,0,5,0.2), "lines"))
```




```
## NULL
```

At present, filtering by sample (column) is possible for `plotFreqHeatmap` via the `order` parameter (which can also be used to reorder columns), but not 'plotAlignments.

```
plotVariants(gol, plotAlignments.args = list(top.n = 3),
             plotFreqHeatmap.args = list(top.n = 3, order = c(1,5,3)),
             left.plot.margin = ggplot2::unit(c(0.1,0,5,0.2), "lines"))
```




```
## NULL
```

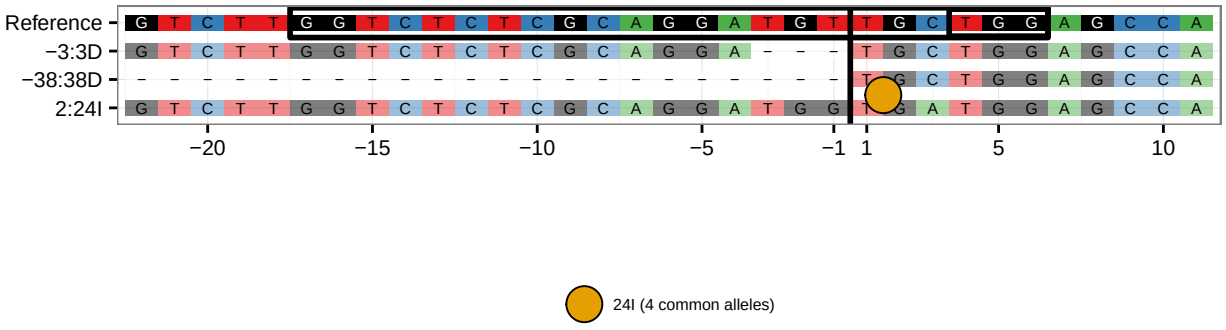
plotAlignments

```
library("CrispRVariants")
data(gol_clutch1)
```

Insertion symbols

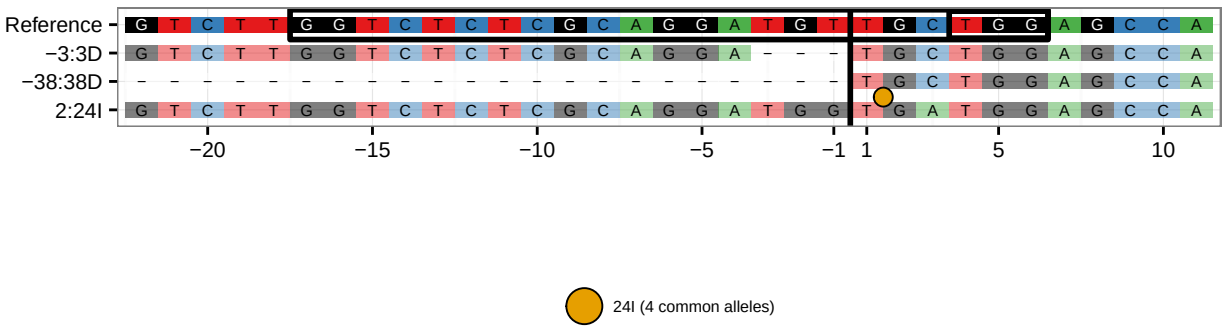
The symbols indicating insertions are controlled by four parameters. `ins.size` (default 3) controls the size of the symbols within the plot area.

```
plotAlignments(gol, top.n = 3, ins.size = 6)
```



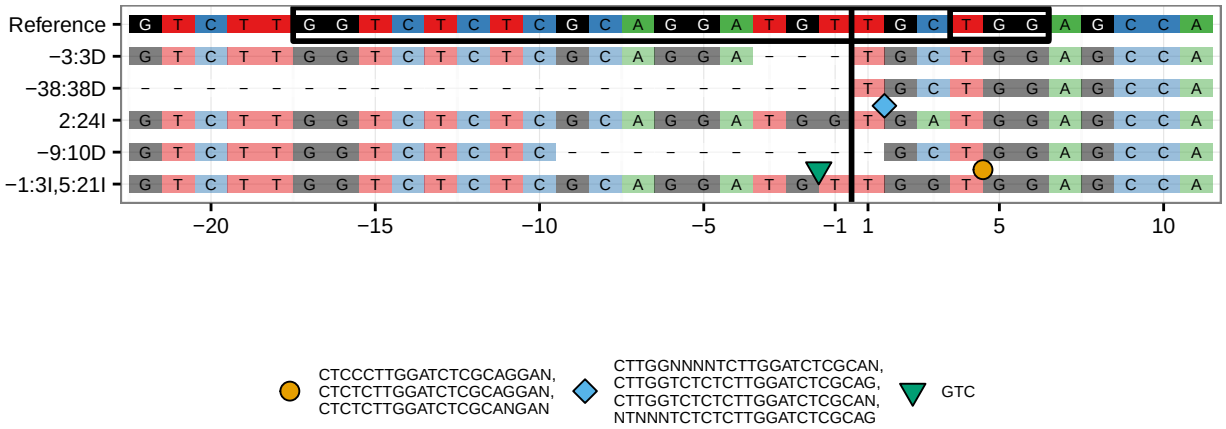
By default the symbols in the legend are the same size as those in the plot, but this can be controlled separately with `legend.symbol.size`.

```
plotAlignments(gol, top.n = 3, legend.symbol.size = 6)
```



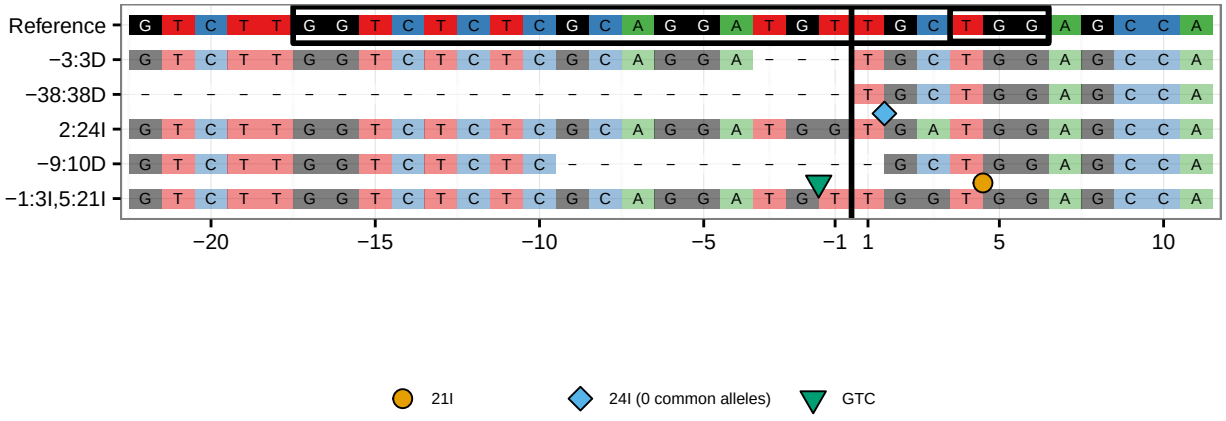
As long sequences can make the plot difficult to read, by default only the length of insertions greater then 20bp is shown. This can be changed with the `max.insertion.size` parameter. If there is more than one allele, the number of (frequent) alleles is indicated.

```
plotAlignments(gol, top.n = 5, max.insertion.size = 25)
```



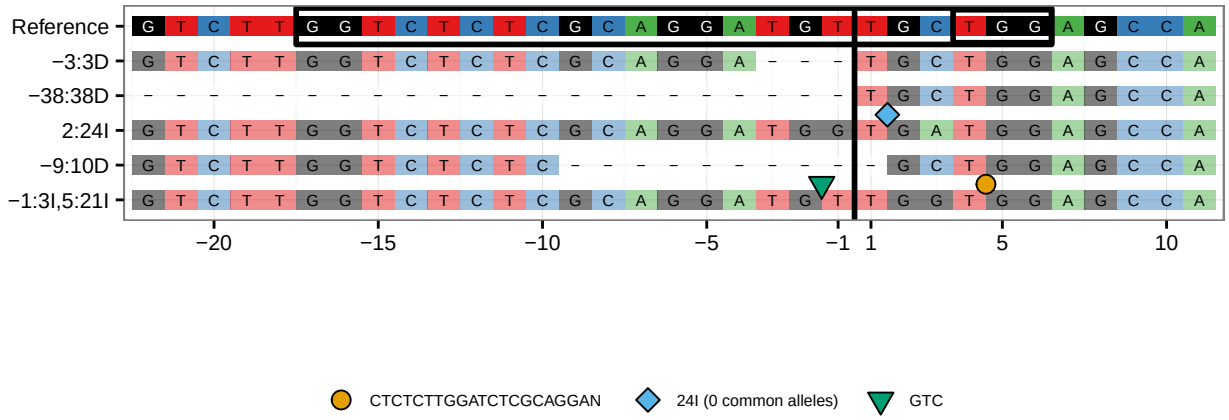
Finally, the parameter `min.insertion.freq` (default 5%) controls how many alleles are displayed at each insertion locus. In large data sets, there will be a substantial proportion of reads with sequencing errors, and we may only wish to display the most common sequences.

```
# Here we set a fairly high value of 50% for min.insertion.freq
# As ambiguous nucleotides occur frequently in this data set,
# there are no alleles passing this cutoff.
plotAlignments(gol, top.n = 5, min.insertion.freq = 50)
```



`max.insertion.size` and `min.insertion.freq` can be combined. In this case, alleles longer than `max.insertion.size` but less frequent than `min.insertion.freq` will be collapsed.

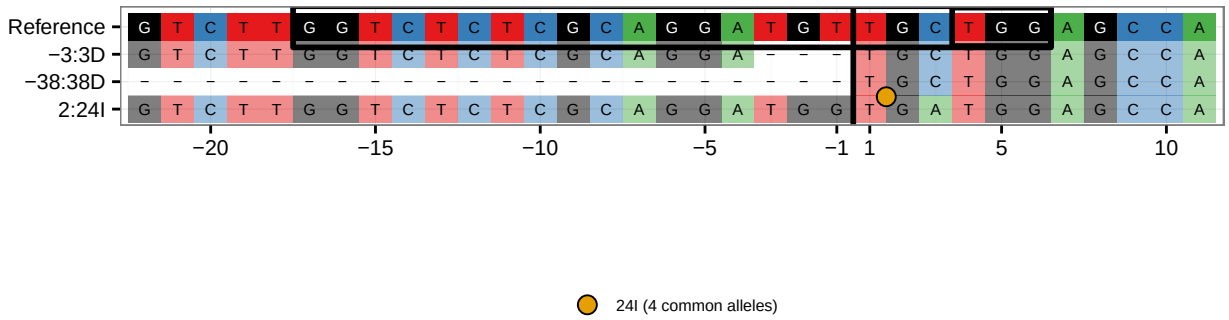
```
plotAlignments(gol, top.n = 5, max.insertion.size = 25, min.insertion.freq = 50)
```



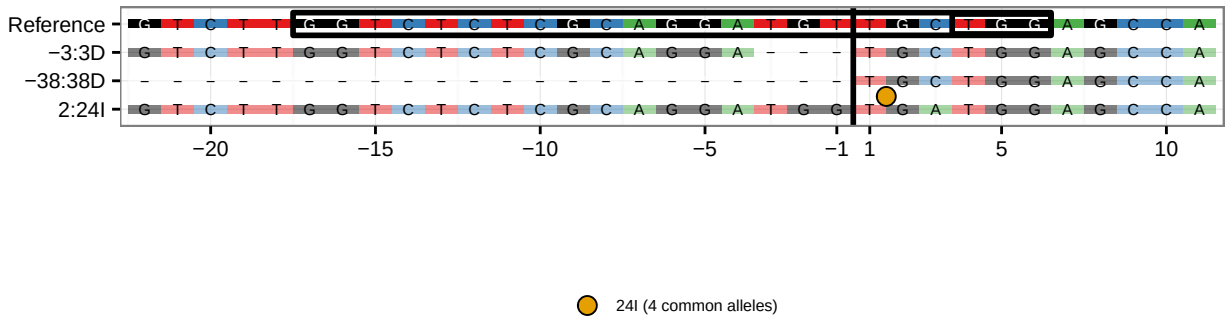
Whitespace between rows

The space between rows is controlled with the `tile.height` parameter (default 0.55). Values closer to 0 increase the space between rows, whilst values closer to 1 decrease the space between rows.

```
# No white space between rows
plotAlignments(gol, top.n = 3, tile.height = 1)
```



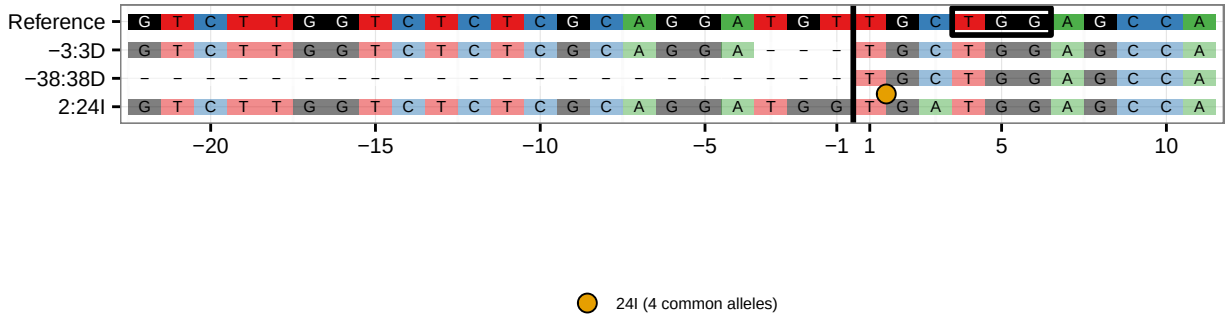

```
# More white space between rows
plotAlignments(gol, top.n = 3, tile.height = 0.3)
```



Box around guide

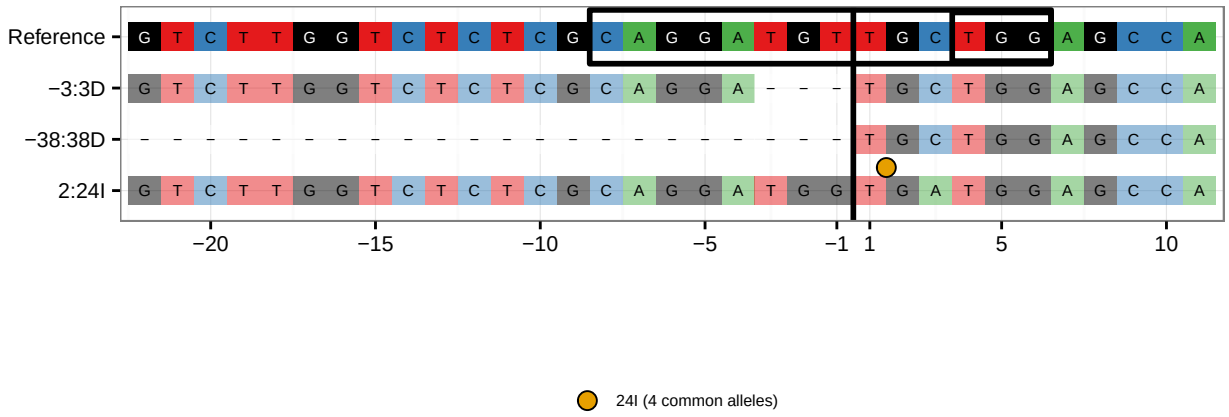
The black box around the guide sequence can be removed by setting `highlight.guide = FALSE`.

```
plotAlignments(gol, top.n = 3, highlight.guide = FALSE)
```



By default, the box around the guide is drawn from 17 bases upstream of the `target.loc` to 6 bases downstream. For experiments with a truncated guide, or other non-standard guide location, the box must be manually specified. The guide location can be altered by setting the `guide.loc` parameter. This can be either an IRanges::IRanges or GenomicRanges::GRanges object.

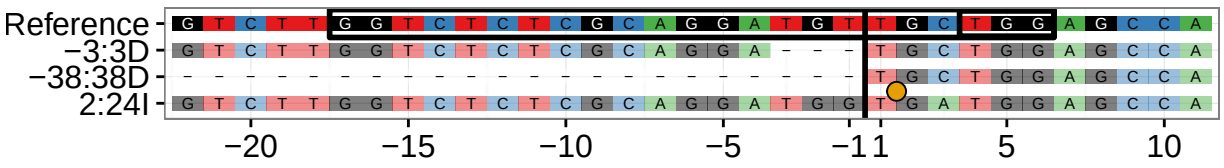
```
library(IRanges)
guide <- IRanges::IRanges(15,28)
plotAlignments(gol, top.n = 3, guide.loc = guide)
```

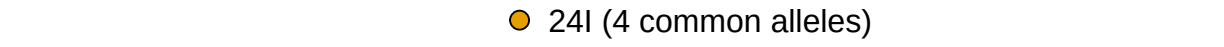


Text sizes

The text within the alignments is controlled by `plot.text.size` (default 0), and can be removed completely by setting `plot.text.size = 0`. The axis labels and legend labels are controlled with `axis.text.size` (default 8) and `legend.text.size` (default 6) respectively. The number of columns in the legend is controlled by `legend.cols` (default 3).

```
# Here we increase the size of the axis labels and make
# two columns for the legend
plotAlignments(gol, top.n = 3, axis.text.size = 12,
               legend.text.size = 12, legend.cols = 2)
```

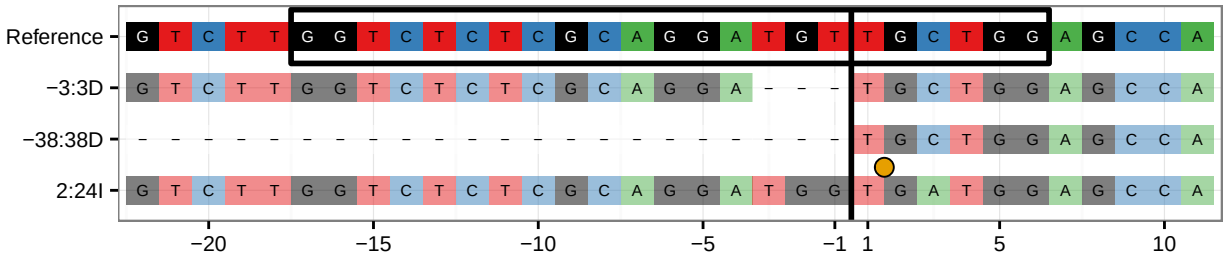




Box around PAM

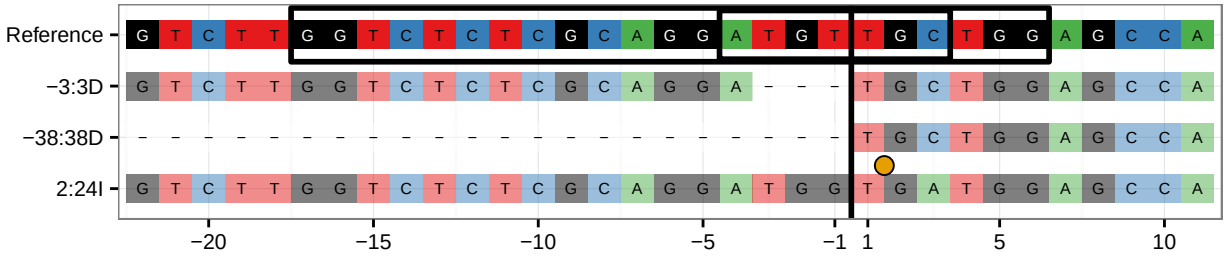
The argument `highlight.pam` determines whether a box around the PAM should be drawn.

```
# Don't highlight the PAM sequence
plotAlignments(gol, top.n = 3, highlight.pam = FALSE)
```



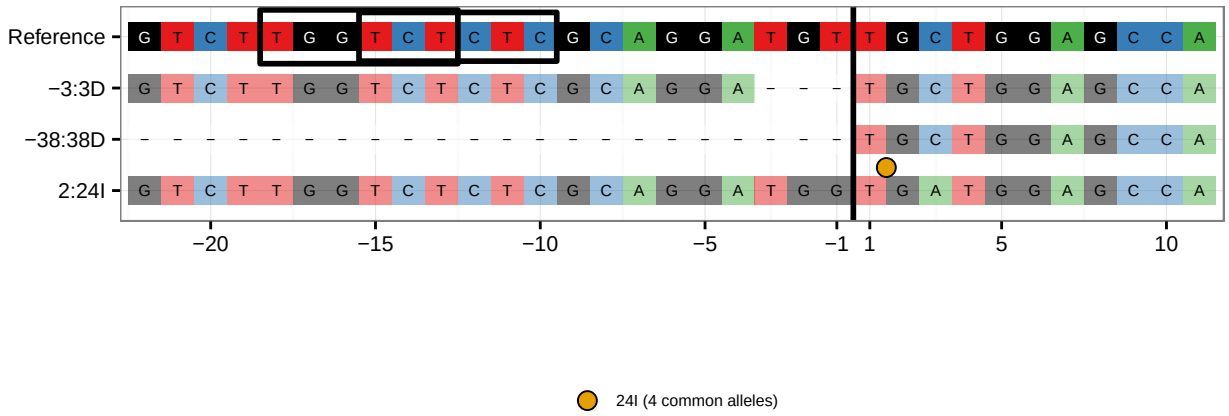
By default this box is drawn 3 nucleotides downstream of the `target.loc`. Other applications might require a different region highlighted. This can be achieved by explicitly setting the start and end positions of the box, with respect to the reference sequence.

```
# Highlight 3 bases upstream to 3 bases downstream of the target.loc
plotAlignments(gol, top.n = 3, pam.start = 19, pam.end = 25)
```



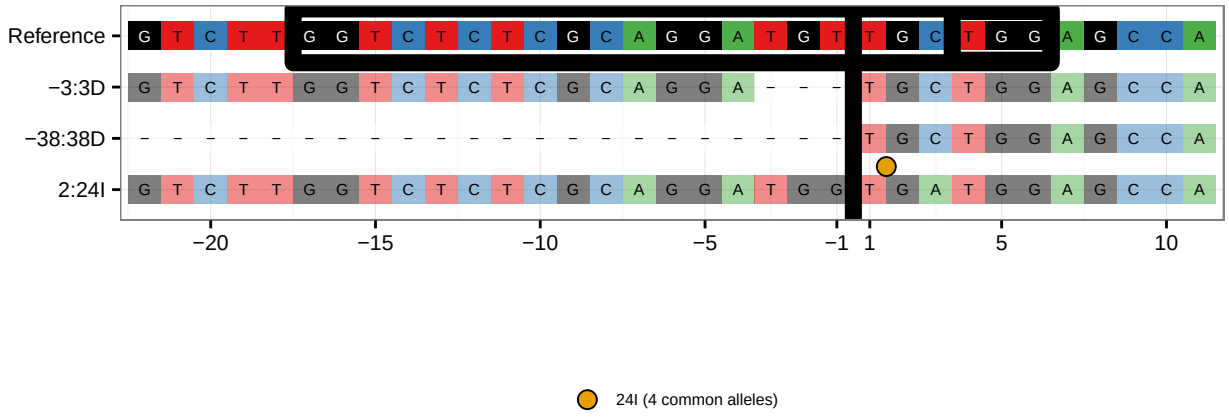
The boxes around the guide and the PAM can both be changed to arbitrary locations, however note that the guide box is specified by a ranges object whilst the PAM box is specified by start and end coordinates. Both coordinates are with respect to the start of the reference sequence. The box around the guide is slightly wider than the box around the PAM.

```
plotAlignments(gol, top.n = 3, guide.loc = IRanges(5,10),
               pam.start = 8, pam.end = 13)
```



The thickness of the lines showing the cut site, the guide and the PAM are controlled with `line.weight` (default 1).

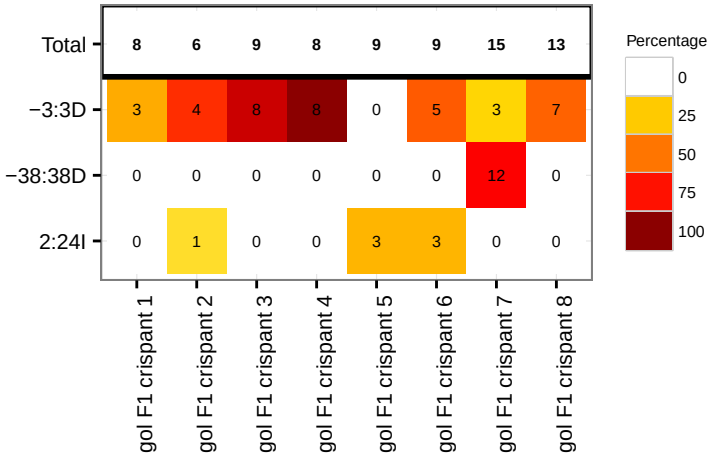
```
plotAlignments(gol, top.n = 3, line.weight = 3)
```



plotFreqHeatmap

Here is the result of calling `plotFreqHeatmap` with default values, showing the three most common variant alleles.

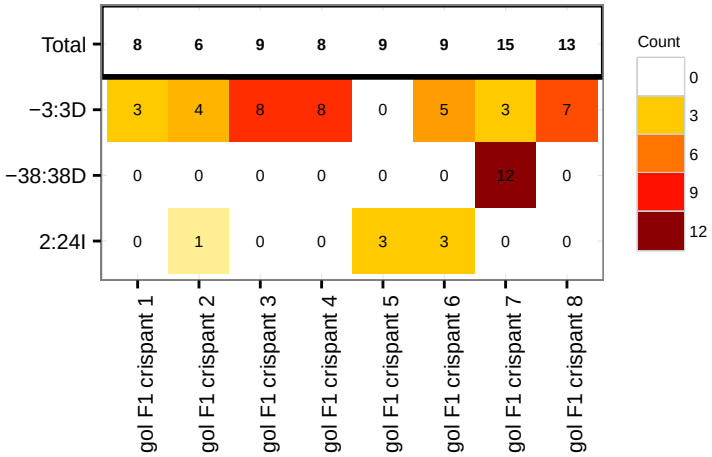
```
plotFreqHeatmap(gol, top.n = 3)
```



Controlling the data plotted

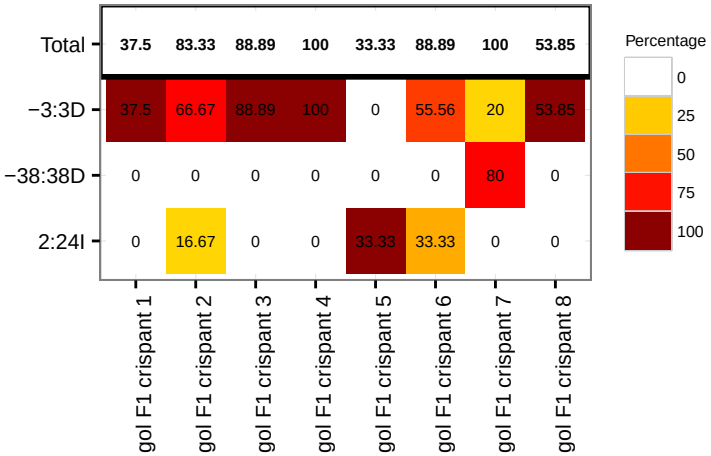
The tiles may be coloured by either the percentage of the column totals (default), or by the counts, by setting `as.percent = FALSE`. The column headers show the total number of sequences in the data. Typically, rare variants are excluded, so the displayed variants do not add up to the column totals.

```
plotFreqHeatmap(gol, top.n = 3, as.percent = FALSE)
```



When calling `plotFreqHeatmap.CrisprSet`, the data can be displayed as percentages instead of raw counts by setting `type = "proportions"` instead of the default `type = "counts"`.

```
plotFreqHeatmap(gol, top.n = 3, type = "proportions")
```



%### Controlling the appearance of the text % %x.axis.title = NULL, x.size = 6, y.size = 8, x.angle = 90, %plot.text.size = 2, line.width = 1, %x.hjust = 1, , x.labels = NULL,

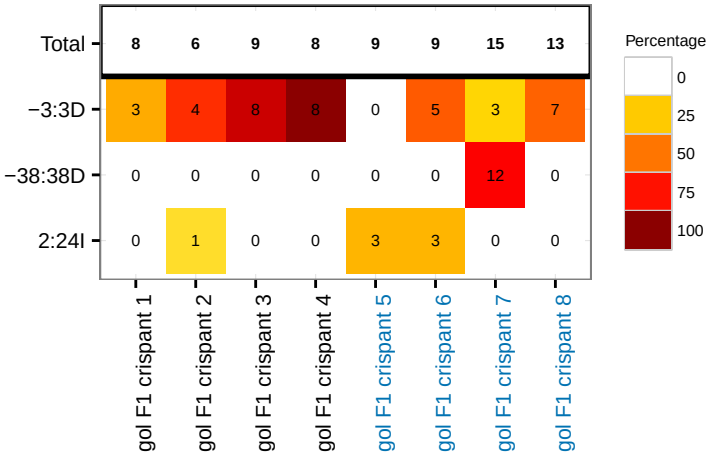
Changing colours of x-labels

The x-labels can be coloured by experimental group. To do this, a grouping vector must be supplied by setting parameter `group`. Columns are ordered according to the levels of the group. There should be one group value per column in the data.

```
ncolumns <- ncol(variantCounts(gol))
ncolumns
```

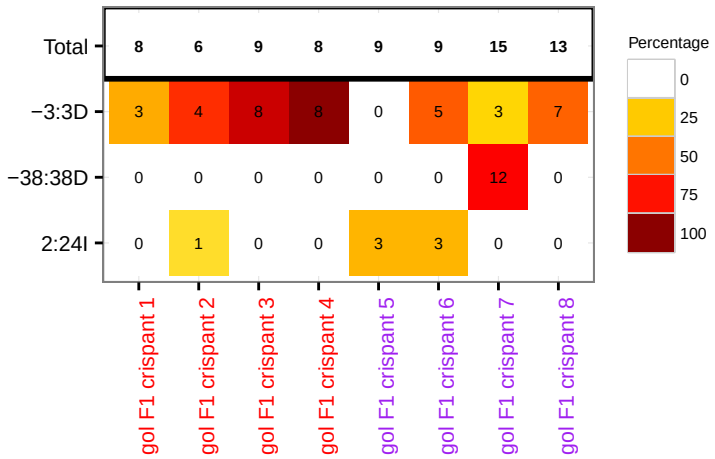
```
## [1] 8
```

```
grp <- rep(c(1,2), each = ncolumns/2)
plotFreqHeatmap(gol, top.n = 3, group = grp)
```



The default colours are designed to be readable on a white background and colour-blind safe. These can be changed by supplying a vector of colours for each level of the group. Colours must be supplied if there are more than 7 experimental groups.

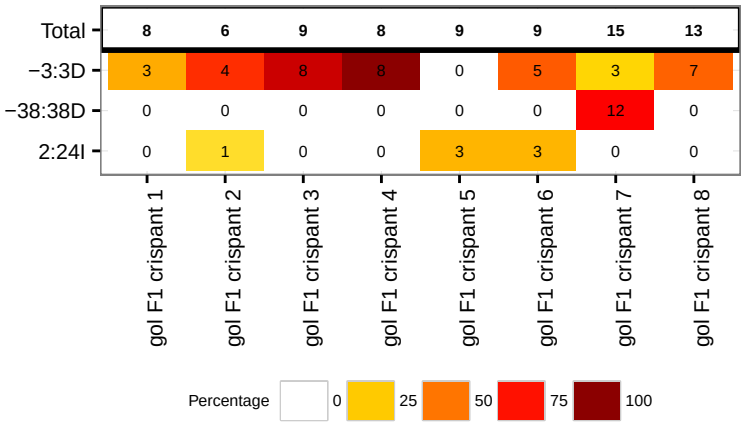
```
grp_clrs <- c("red", "purple")
plotFreqHeatmap(gol, top.n = 3, group = grp, group.colours = grp_clrs)
```



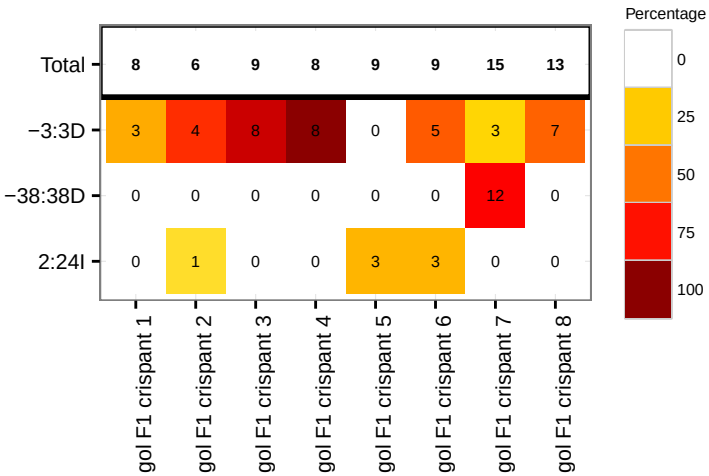
Controlling the appearance of the legend

The legend position is controlled via `legend.position`, which is passed to `ggplot2::theme`. Similarly `legend.key.height` controls the height of the legend. See the ggplot docs for more information.

```
plotFreqHeatmap(gol, top.n = 3, legend.position = "bottom")
```



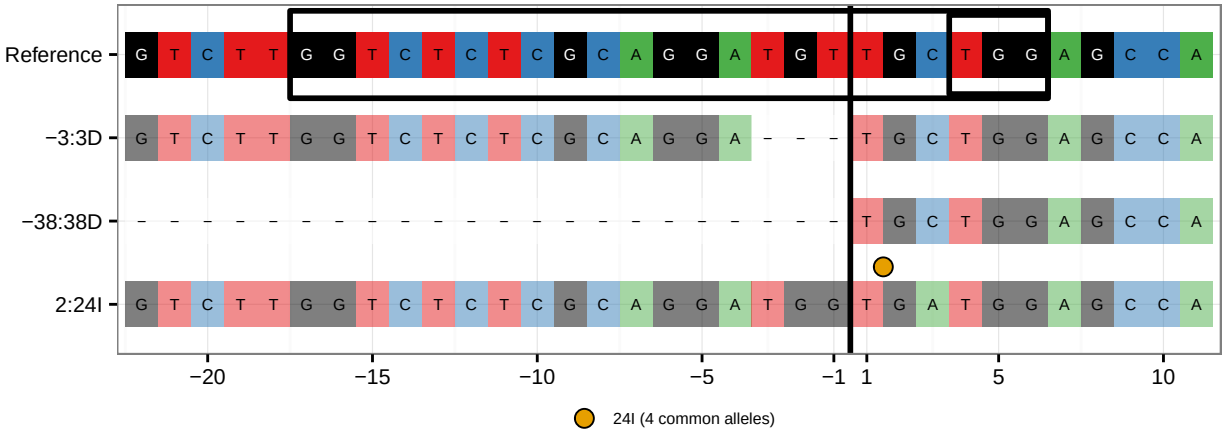

```
plotFreqHeatmap(gol, top.n = 3,
                legend.key.height = ggplot2::unit(1.5, "lines"))
```



Other modifications

`plotAlignments` and `plotFreqHeatmap` both return `ggplot` objects, which can be adjusted via `theme()`. For example, to decrease the space between the legend and the plot:

```
p <- plotAlignments(gol, top.n = 3)
p + theme(legend.margin = ggplot2::unit(0, "cm"))
```



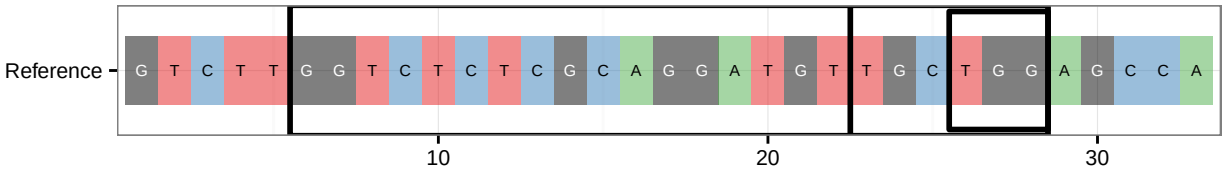
Using CrispRVariants plotting functions independently

The CrispRVariants plotting functions are intended to be used within a typical CrispRVariants pipeline, where the correct arguments are extracted from a CrisprSet object. However, with some data formatting, it is also possible to use these functions with standard R objects.

Plot the reference sequence

Processing large data with CrispRVariants requires some time. It can be useful to first plot the reference sequence to check that the intended target location is specified. Here we use the reference sequence from the *gol* data set included in CrispRVariants. Any Biostrings::DNAString can be used. Note that CrispRVariants::plotAlignments accepts elliptical arguments in its signature, so non-signature arguments must be supplied by name. The code below shows the minimum arguments required for running CrispRVariants::plotAlignments.

```
# Get a reference sequence
library("CrispRVariants")
data(gol_clutch1)
ref <- gol$ref

#Then to make the plot:
plotAlignments(ref, alns = NULL, target.loc = 22, ins.sites = data.frame())
```



Plot pairwise alignments using CrispRVariants

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

For this we use the synthetic data set, which is included in the CrispRVariants package as raw and mapped reads. The corresponding guide is available in bed format.

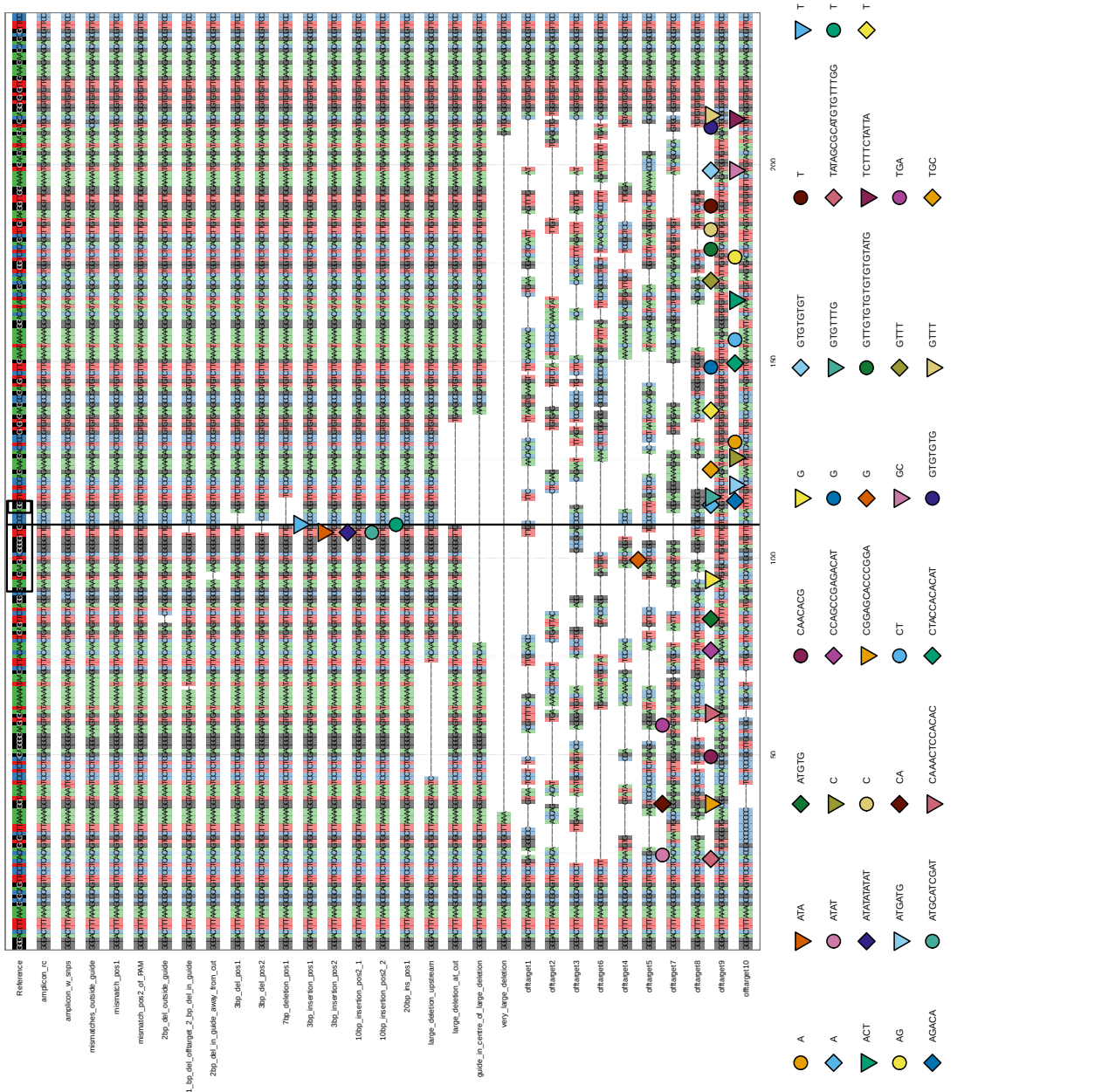
```
library("Biostrings")
library("CrispRVariants")
library("ShortRead")

# This function converts pairwise alignments to reference-based alignments,
# and produces a table of insertion locations.
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)
}


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 <- CrispRVariants::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 = 4,
               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
ATA
ATGTG
CAACACG
G
GTGTGTGT
T
T
A
ATAT
C
CCAGCCGAGACAT
G
GTGTTTG
TATAGCGCATGTGTTTTGG
T
ACT
ATATATATAT
C
CGGAGCACCCGGA
G
GTTGTGTGTGTGTGTATG
TCTTTTCTATTA
T
AG
ATGATG
CA
CT
GC
GTTT
TGA
AGACA
ATGCATCGAT
CAAACTCCACAC
CTACCACACAT
GTGTGTG
GTTT
TGC

Customise CrispRVariants::plotVariants

This example shows some of the internal data stored in a CrisprSet object, which can be accessed to create a customised plot.

CrispRVariants::plotFreqHeatmap by default groups reads by experimental condition, where one bam file represents one experimental condition. 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)
```