gRog: An introduction

Zachary Skidmore

2015-06-18

Introduction

Intuitively visualizing and interpreting data from high-throughput genomic technologies continues to be challenging. Graphical R observations of genomics (gRog) attempts to alleviate this burden by providing highly customizable publication-quality graphics focused primarily on a cohort level (i.e., multiple samples/patients). gRog attempts to maintain a high degree of flexibility while leveraging the abilities of ggplot2 and bioconductor to achieve this goal.

Functions

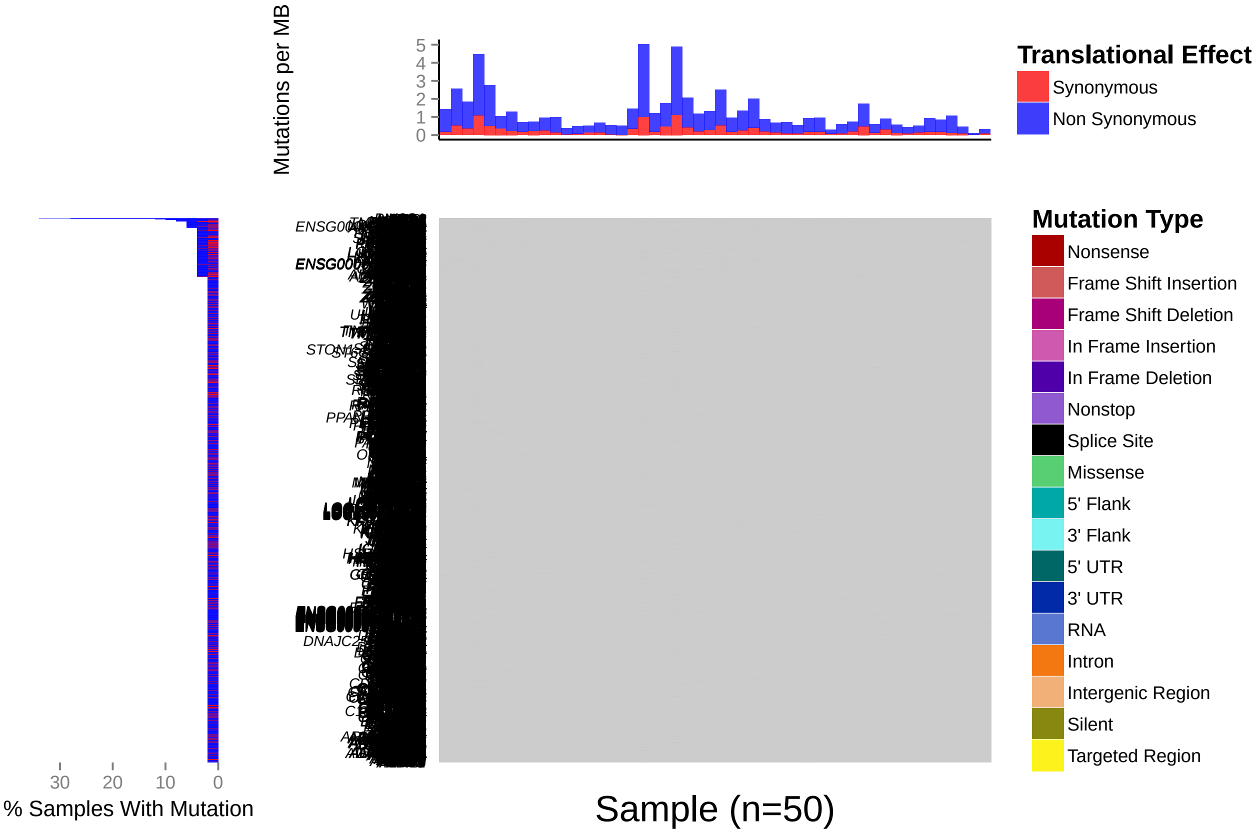
mutSpec

`mutSpec` provides a method of visualizing the mutational landscape of a cohort. The input to `mutSpec` consists of a data frame derived from either a .maf (version 2.4) file or a file in MGI annotation format (obtained from The Genome Modeling System). `mutspec` will display the mutation occurrence and type in the main panel while showing the mutation rate and the percentage of samples with a mutation in the side panels. Conflicts arising from multiple mutations in the same gene/sample cell are resolved by a hierarchical removal of mutations keeping the most deleterious as defined by the order of the "mutation type" legend. This hierarchical removal occurs only in the main panel.

`BRCA_MAF` is a truncated MAF file consisting of 50 samples from the TCGA project corresponding to Breast invasive carcinoma (complete data). Using this dataset we can view the default behavior of `mutSpec`:

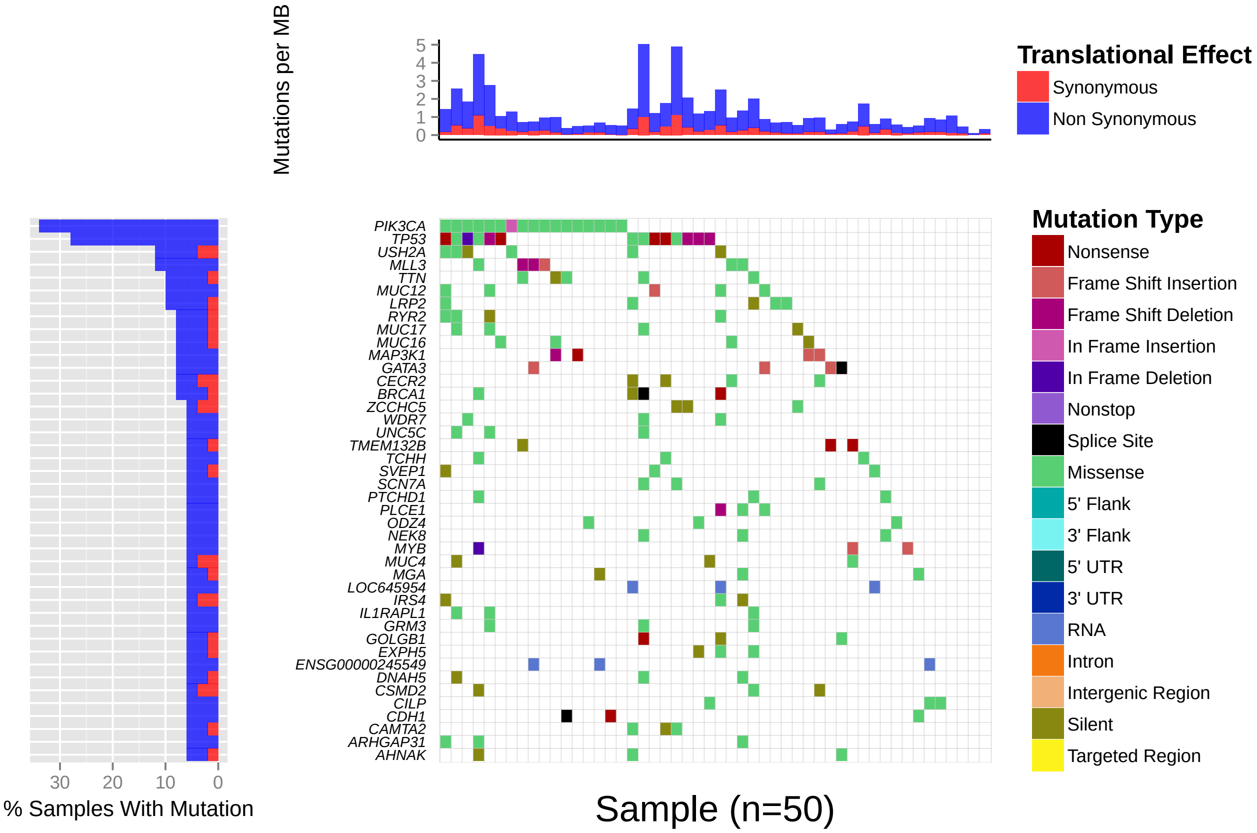
```
library(GGgenome)
```

```
mutSpec(brcaMAF)
```



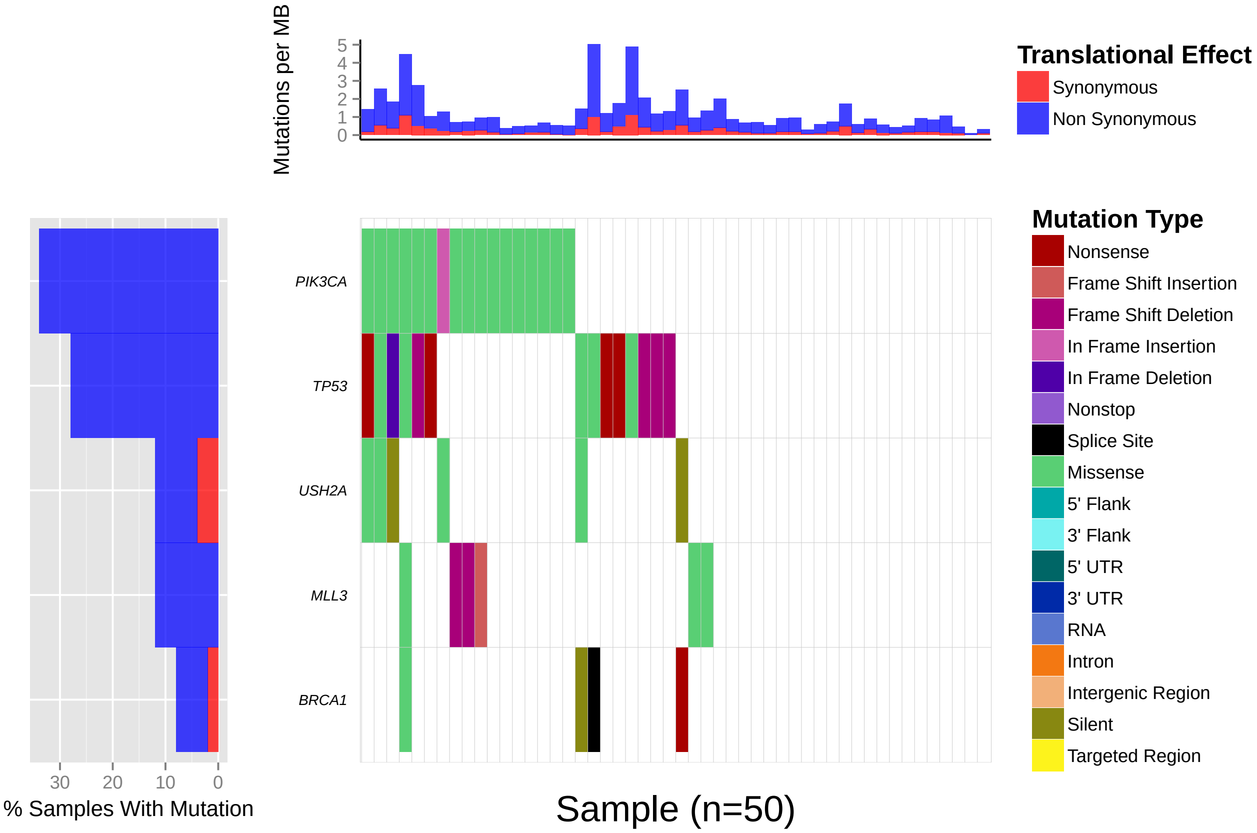
This type of view is of limited use given the large number of genes. Often it is beneficial to reduce the number of cells in the plot by limiting the number of genes plotted. There are two ways to do this, the `recurrence_cutoff` parameter will remove genes from the data which do not have at least x number of samples mutated. For example the `BRCA_MAF` data set contains 50 samples, to plot genes with mutations present in at least 3 (6%) of samples:

```
mutSpec(brcaMAF, recurrence_cutoff=3)
```



Alternatively one can select genes of interest using the `genes` parameter. For example, if it was desirable to plot only “PIK3CA”, “TP53”, “USH2A”, “MLL3”, AND “BRCA1”:

```
mutSpec(brcaMAF, genes=c("PIK3CA", "TP53", "USH2A", "MLL3", "BRCA1"))
```

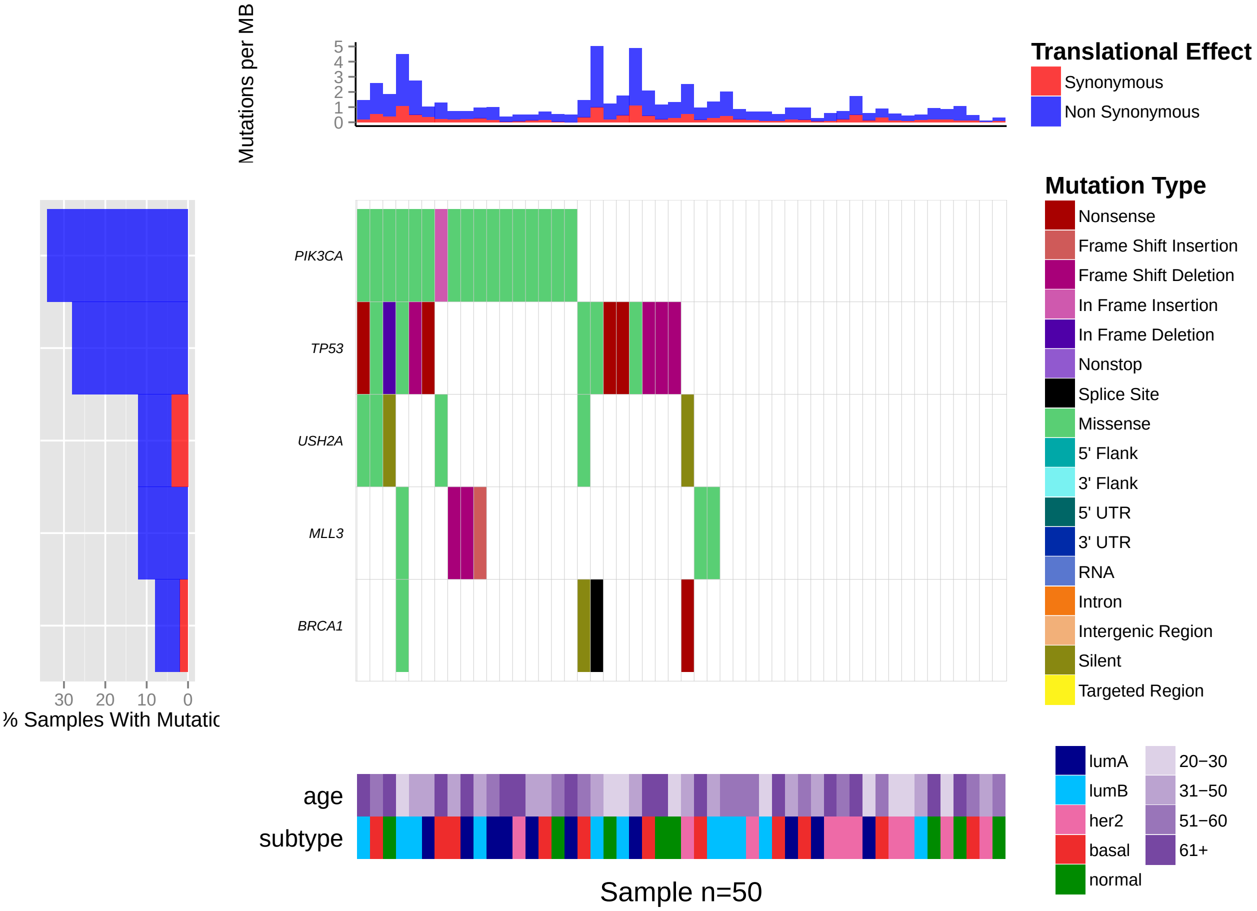


It is important to note that the Mutation Burden plot does not change during these subsets, this is calculated directly from the input via the formula: $mutations\ in\ sample/coverage\ space * 1000000$ by default the coverage space defaults to the size in base pairs of the "SeqCap EZ Human Exome Library v2.0". This default can be changed via the parameter `coverage_space`. This calculation is only meant to be a rough estimate as actual coverage space can vary from sample to sample, for a more accurate calculation the user has the option to supply an optional argument `mutBurden` giving the users own calculation for each sample, this should be a data frame with columns 'sample', 'mut_burden' taking the following form:

sample	mut_burden
TCGA-A1-A0SO-01A-22D-A099-09	2.43899950843364
TCGA-A2-A0EU-01A-22W-A071-09	2.72352675516873
TCGA-A2-A0ER-01A-21W-A050-09	1.8710163696741
TCGA-A2-A0EN-01A-13D-A099-09	1.54831599746482
TCGA-A1-A0SI-01A-11D-A142-09	2.54618151195248
TCGA-A2-A0D0-01A-11W-A019-09	2.01022626591171
TCGA-A2-A0D0-01A-11W-A019-09	1.93174341475855
TCGA-A1-A0SI-01A-11D-A142-09	1.99733102468375
TCGA-A2-A0CT-01A-31W-A071-09	1.66900256749179
TCGA-A2-A04U-01A-11D-A10Y-09	1.98200203310719

In addition to specifying the mutation burden the user also has the ability to plot additional clinical data. The clinical data supplied should be a data frame in "long" format with column names "sample", "variable", "value". It is recommended to use the `melt` function in the package reshape2 to coerce data into this format. Here we add clinical data to be plotted and specify a custom order and colours for variables putting these values in two columns within the legend:

```r
# create fake clinical data
subtype <- c('lumA', 'lumB', 'her2', 'basal', 'normal')
subtype <- sample(subtype, 50, replace=TRUE)
age <- c('20-30', '31-50', '51-60', '61+')
age <- sample(age, 50, replace=TRUE)
sample <- as.character(unique(brcaMAF$Tumor_Sample_Barcode))
clinical <- as.data.frame(cbind(sample, subtype, age))

# melt the data
library(reshape2)
clinical <- melt(clinical, id.vars=c('sample'))

# Run mutSpec
mutSpec(brcaMAF, clinDat=clinical, clin.var.colour=c('lumA'='blue4', 'lumB'='deepskyblue',
'her2'='hotpink2', 'basal'='firebrick2', 'normal'='green4', '20-30'='#ddd1e7',
'31-50'='#bba3d0', '51-60'='#9975b9', '61+'='#7647a2'),
genes=c("PIK3CA", "TP53", "USH2A", "MLL3", "BRCA1"), clin.legend.col=2, clin.var.order=c('lumA',
'lumB', 'her2', 'basal', 'normal', '20-30', '31-50', '51-60', '61+'))
```

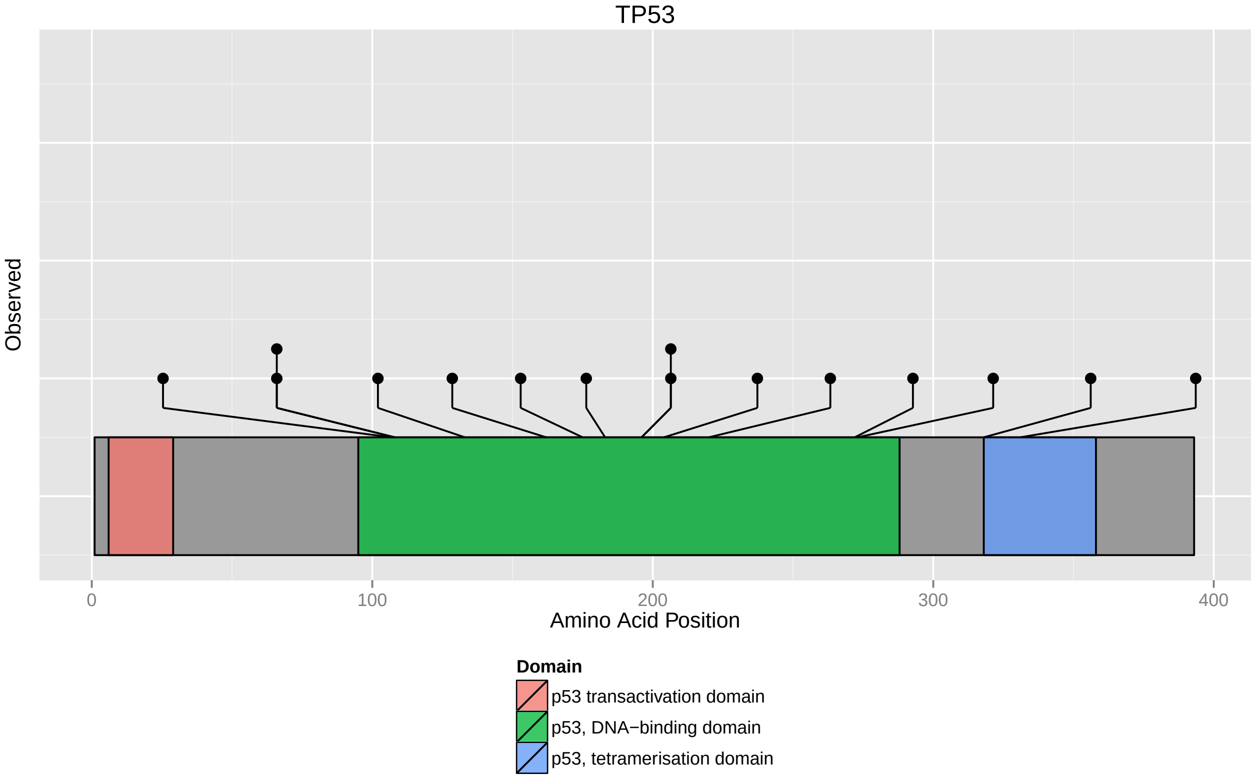


Occasionally there may be samples not represented within the .maf file (due to a lack of mutations). It may still be desirable to plot these samples. To accomplish this simply add the samples into the appropriate column before loading the data and leave the rest of the columns as NA. Additionally it may be desireable to plot data not in a standard format. If this is the case it is recommended to set the `file_type` parameter to 'MGI' and name columns as 'sample', 'gene_name', and 'trv_type'. Valid levels for 'trv_type' are: "nonsense", "frame_shift_del", "frame_shift_ins", "splice_site_del", "splice_site_ins", "splice_site", “nonstop”, “in_frame_del”, “in_frame_ins”, “missense”, “splice_region”, “5_prime_flanking_region”, “3_prime_flanking_region”, “3_prime_untranslated_region”, “5_prime_untranslated_region”, “rna”, “intronic”, and “silent”.

lolliplot

`lolliplot` provides a method for visualizing mutation hotspots on a transcript framework. The input consist of a data frame with columns ‘transcript_name’, ‘gene’ and ‘amino_acid_change’ giving the ensembl transcript id, gene name, and the amino acid change respectivley. `lolliplot` queries various online databases to extract meta data for the transcript framework and as such needs an active internet connection. `lolliplot` also assumes the species to be H.sapiens, this assumption can be changed via the parameter `taxId` which takes a UniProt Taxonomic identifier.

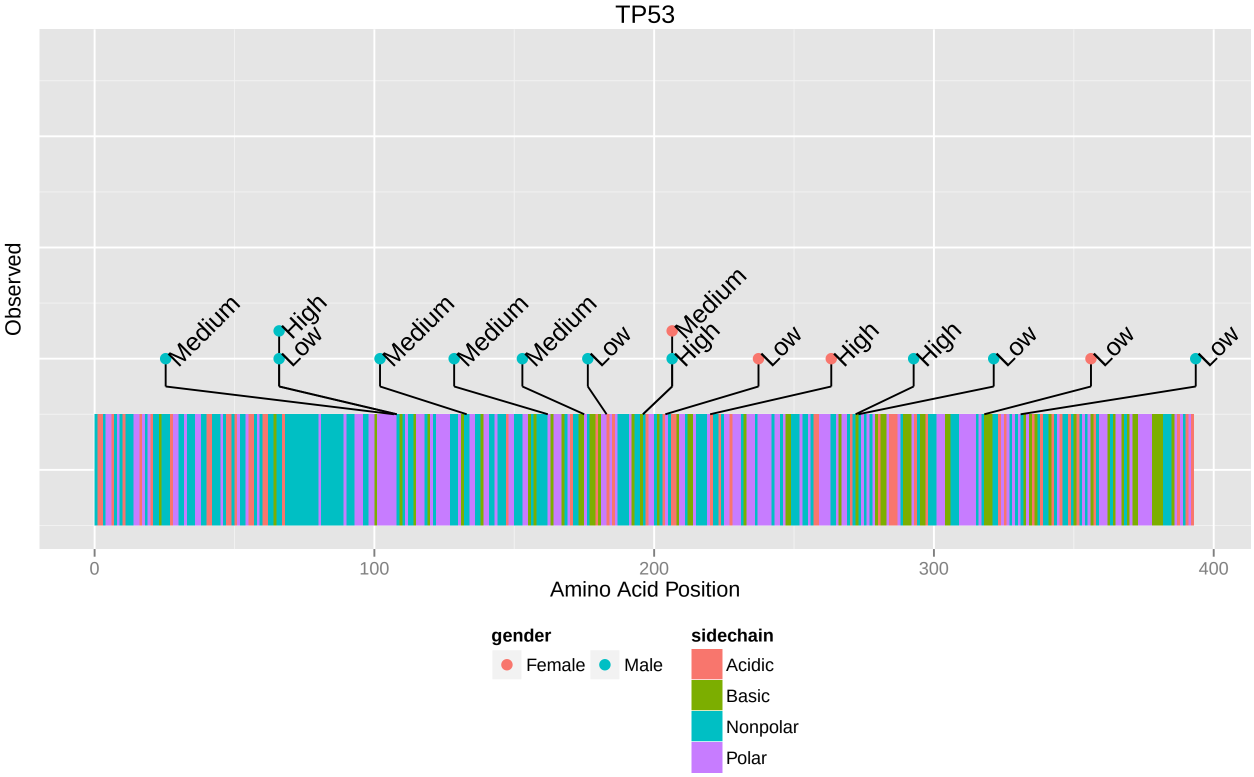
```r
# Create input data
data <- brcaMAF[brcaMAF$Hugo_Symbol == 'TP53',c('Hugo_Symbol', 'amino_acid_change_WU')]
data <- as.data.frame(cbind(data, 'ENST00000269305'))
colnames(data) <- c('gene', 'amino_acid_change', 'transcript_name')

# Call lolliplot
lolliplot(data)
```



In an effort to maintain a high degree of flexibility the user has the option of selecting columns on which to fill and label. The parameters `fill_value` and `label_column` allow this behavior by taking column names on which to fill and label respectively. Additionally one can plot the amino acid sidechain information in lieu of protein domains.

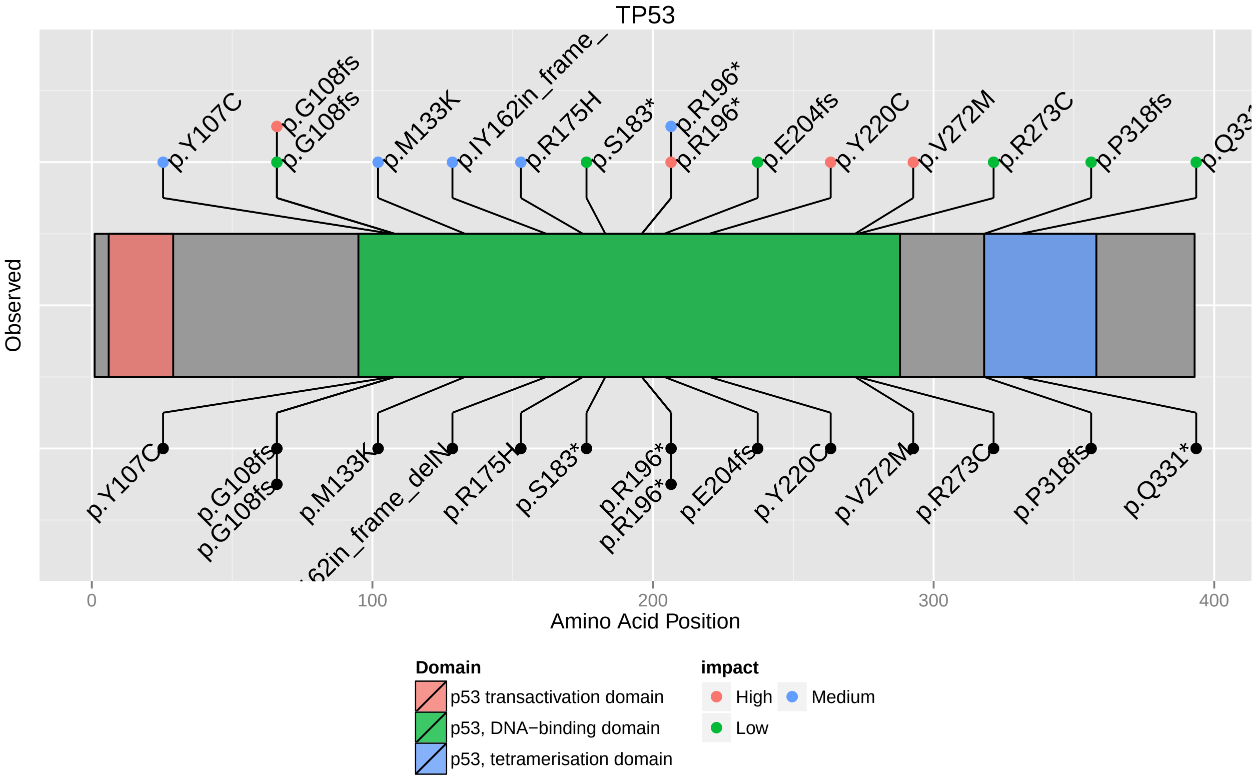
```r
# Add additional columns to the data
data$gender <- sample(c("Male", "Female"), 15, replace=T)
```

```
data$impact <- sample(c("Low", "Medium", "High"), 15, replace=T)

lolliplot(data, fill_value='gender', label_column='impact', plot_sidechain=TRUE)
```



The user has the option of plotting an additional track on the area underneath the gene track via the parameter `y`. Input for this additional layer consists of a data frame with column names ‘transcript_name’ and ‘amino_acid_change’ in p. notation. `lolliplot` will capture the input corresponding to the input given in `x` and plot the subsequent data. Additional fill and label columns are allowed however they must match those variables given in `fill_value` and `label_column`.

```
# create additional data
data2 <- data[,2:4]

lolliplot(data, y=data2, fill_value='impact', label_column='amino_acid_change')
```



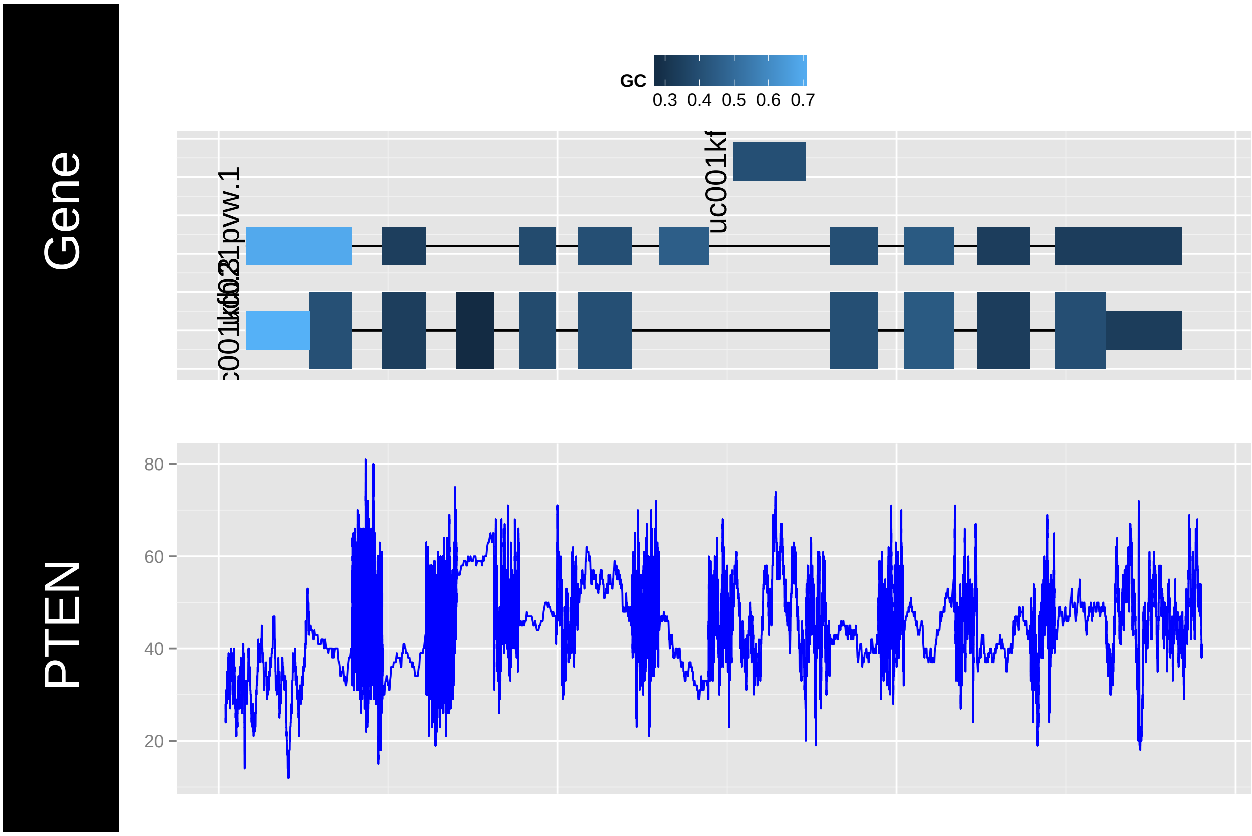
`lolliplot` uses a force field model from the package FField to repulse and attract data in an attempt to achieve a reasonable degree of seperation between points. Suitable defaults have been set. However, on occasion the user may need to manually adjust the force field parameters. This can be done for both upper and lower tracks via `rep.fact`, `rep.dist.lmt`, `attr.fact`, `adj.max`, `adj.lmt`, `iter.max` please see documentation for FField::FFieldPtRep for a complete description of these parameters.

genCov

genCov provides a methodology for viewing coverage information in relation to a gene track. It takes a list of data frames with each data frame containing columns “end” and “cov” corresponding to the region of interest. Additional required arguments are a Granges object specifying the region of interest, a BSgenome, and a txdb object containing transcription metadata (see the package Granges for more information). genCov will plot a genomic features track and align coverage data in the list to the plot:

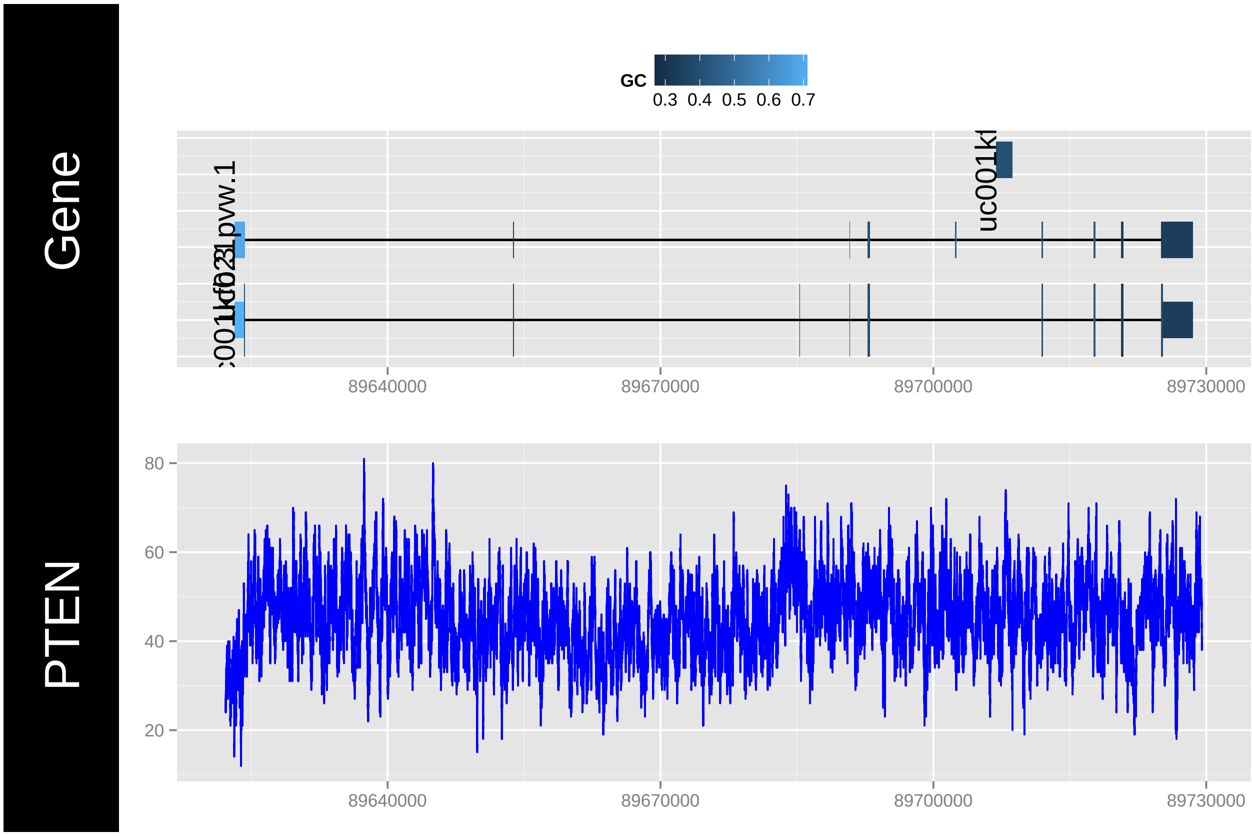
```
# need transcript data for reference
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

# need a biostrings object for reference
library(BSgenome.Hsapiens.UCSC.hg19)
genome <- BSgenome.Hsapiens.UCSC.hg19

# need Granges object
gr <- GRanges(seqnames=c("chr10"), ranges=IRanges(start=c(89622195), end=c(89729532)), strand=strand(c(

# save the data as a named list
data <- list("PTEN" = ptenCOV)
```

```r
# Plot the graphic
genCov(data, txdb, gr, genome)
```



By default genCov will perform a log compression of genomic space for each feature type,‘Intron’,‘CDS’,‘UTR’. The degree of compression can be set via the parameter `base` which will perform the appropriate log compression for the features specified in `transform`. The user can turn off this behavior by setting transform to NULL. Defaults to log-10 compression for intronic space, and log-2 compression for CDS and UTR.

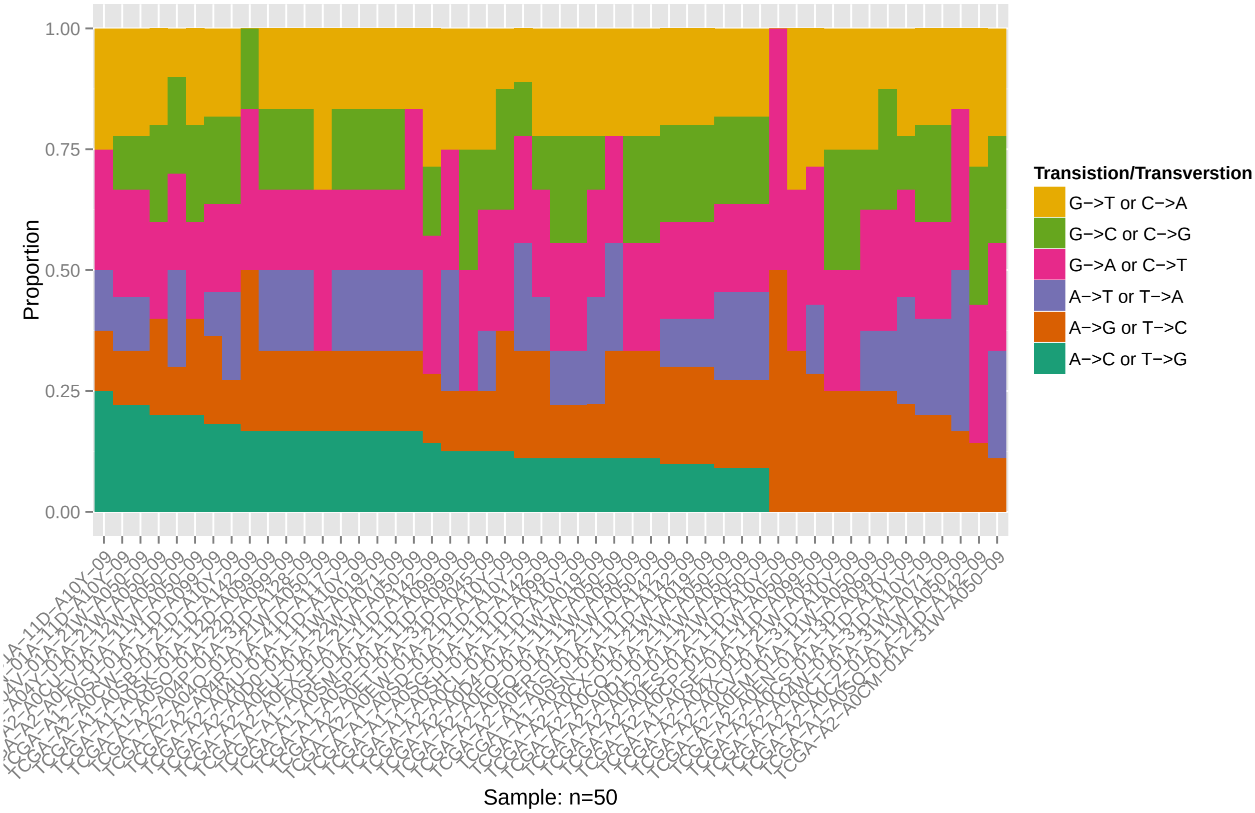
```r
genCov(data, txdb, gr, genome, transform=NULL)
```



TvTi

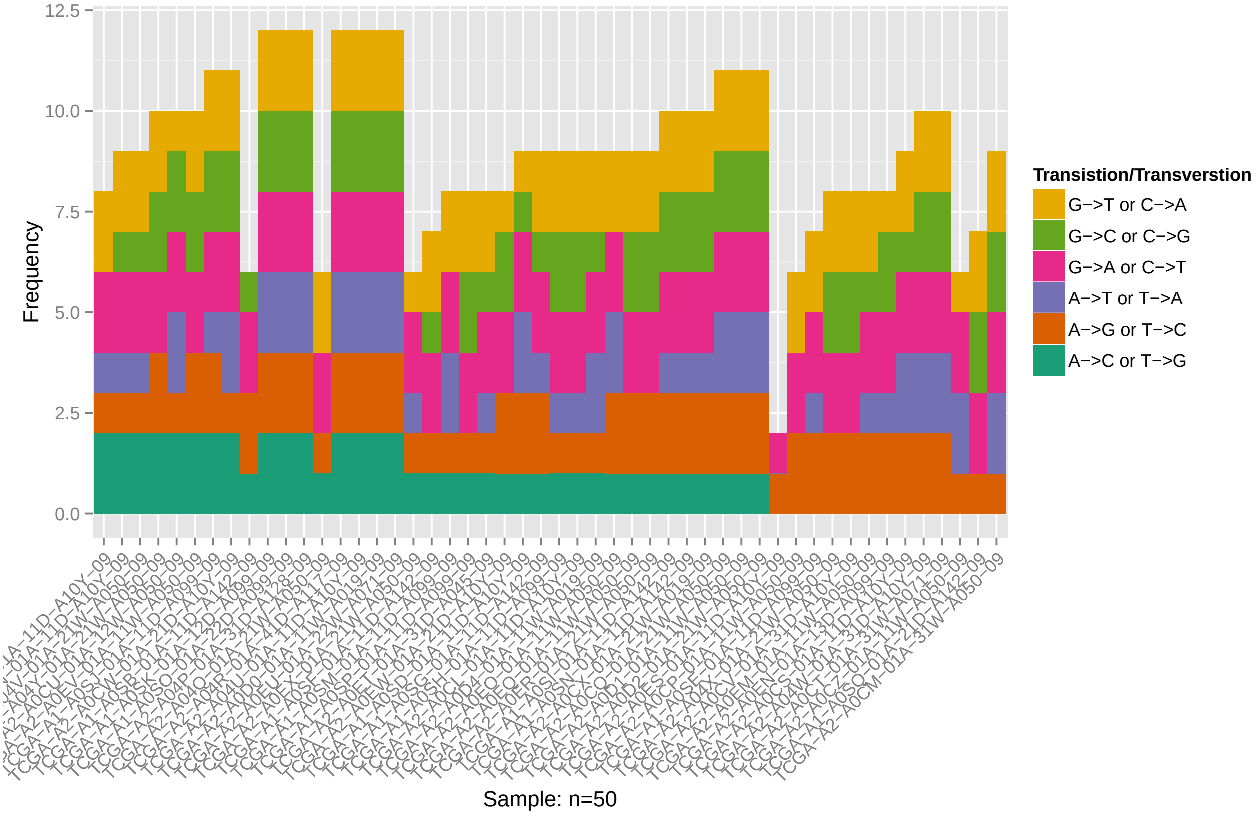
`TvTi` provides a framework for visuzlizing transversions and transitions for a cohort. Input consists of a data frame with column names “Tumor_Sample_Barcode”, “Reference_Allele”, “Tumor_Seq_Allele1”, and “Tumor_Seq_Allele2” for a .maf file. Alternativley the user can set the `file_type` parameter to “MGI” and supply columns “sample”, “reference”, and “variant”.

```
TvTi(brcaMAF)
```



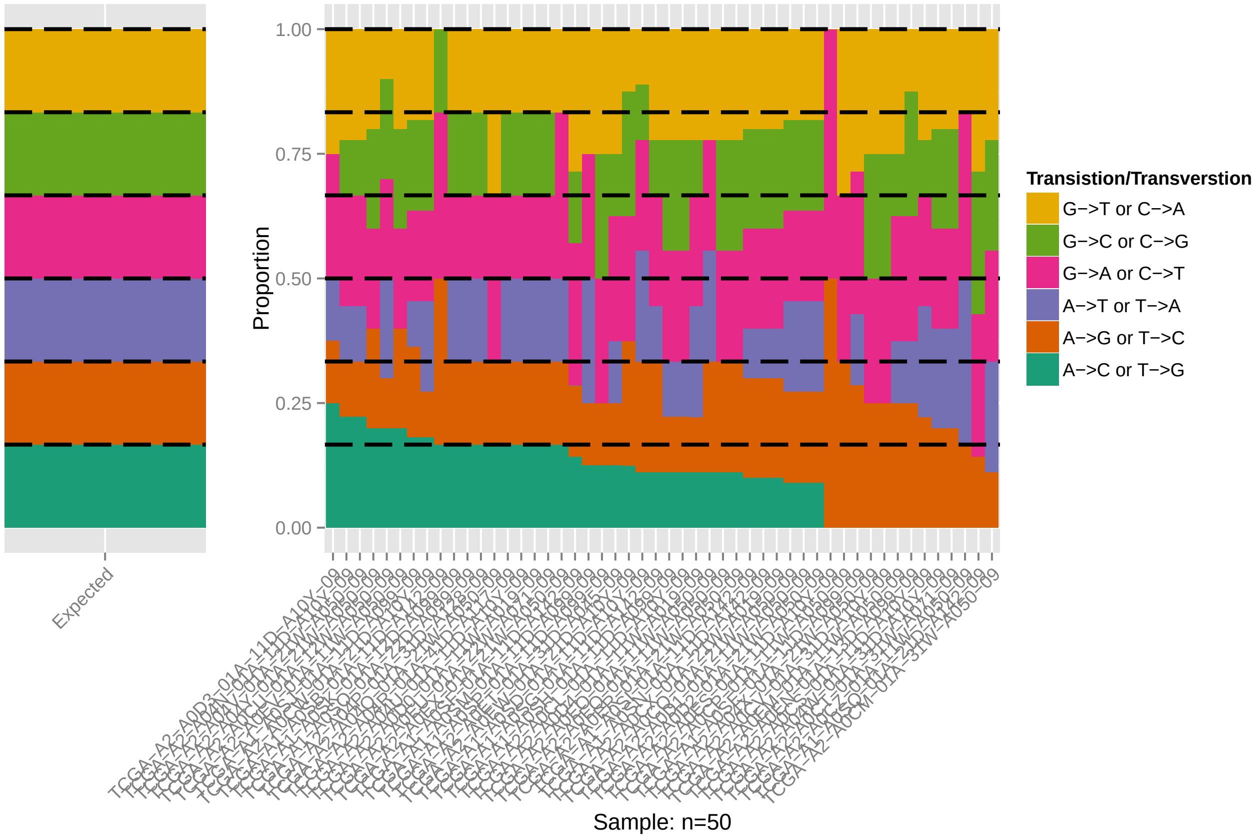
By default `TvTi` will plot the propotion of each transition/transversion seen in each sample. This can be overwritten to plot the frequency seen via the parameter `type`.

```
TvTi(brcaMAF, type='Frequency')
```



If there are prior expectations about the transition/transversion rate that data can be specified in y which takes a named vector with names corresponding to each transition/transversion type.

```
expec <- c("A->C or T->G"=1/6, "A->G or T->C"=1/6, "A->T or T->A"=1/6, "G->A or C->T"=1/6, "G->C or C->

TvTi(brcaMAF, expec)
```



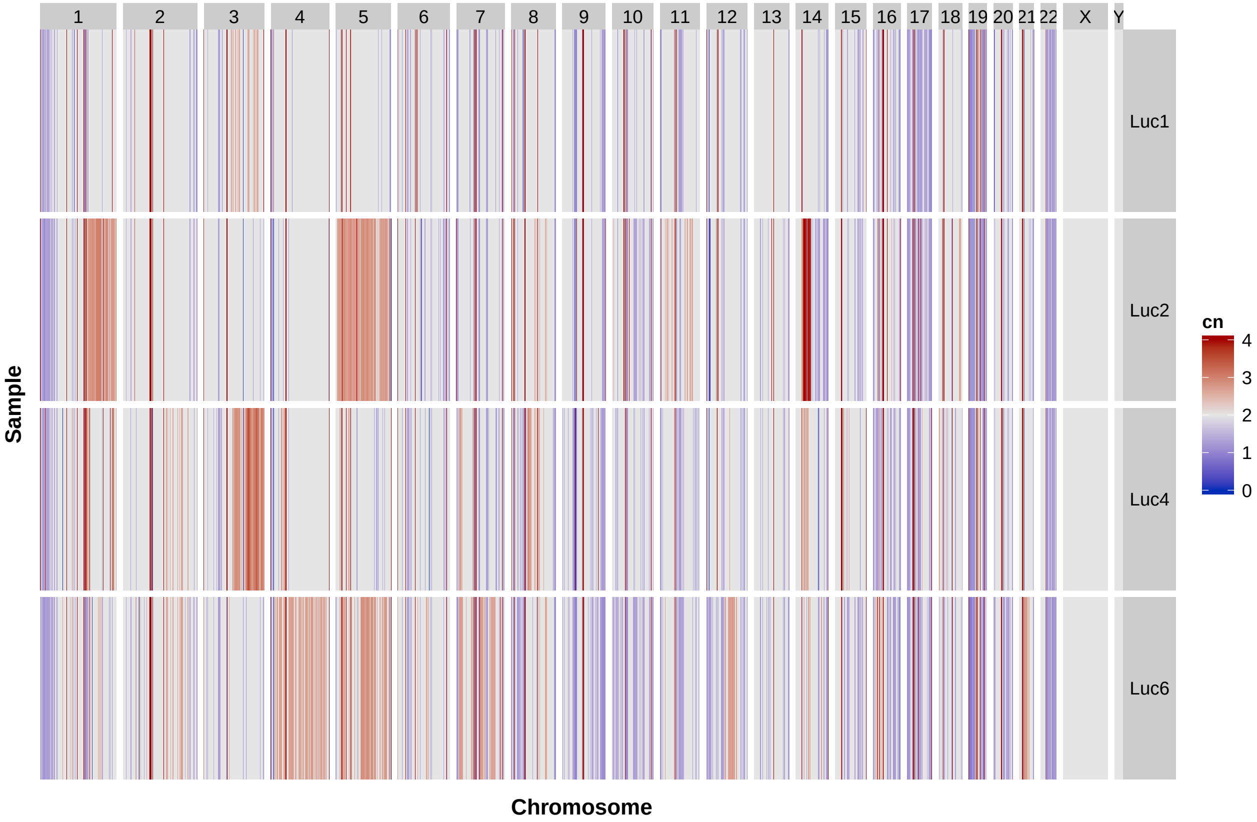
cnSpec

cnSpec produces a plot displaying copy number segments at a cohort level. Basic input consists of a data frame with column names 'chromosome', 'start', 'end' 'segmean' and 'sample' with rows denoting segments with copy number alterations. A UCSC genome is also required, defaults to 'hg19', to determine chromosomal boundaries.

```
# Example input to x
head(LucCN)
```

chromosome start end probes segmean sample 1 1 232500 267500 15 3.31 Luc1 2 1 837500 2582500 699 1.06 Luc1 3 1 2587500 2630000 18 5.33 Luc1 4 1 2690000 2957500 108 1.29 Luc1 5 1 2980000 4072500 379 1.22 Luc1 6 1 6020000 6807500 316 1.24 Luc1

```
cnSpec(LucCN, genome="hg19")
```

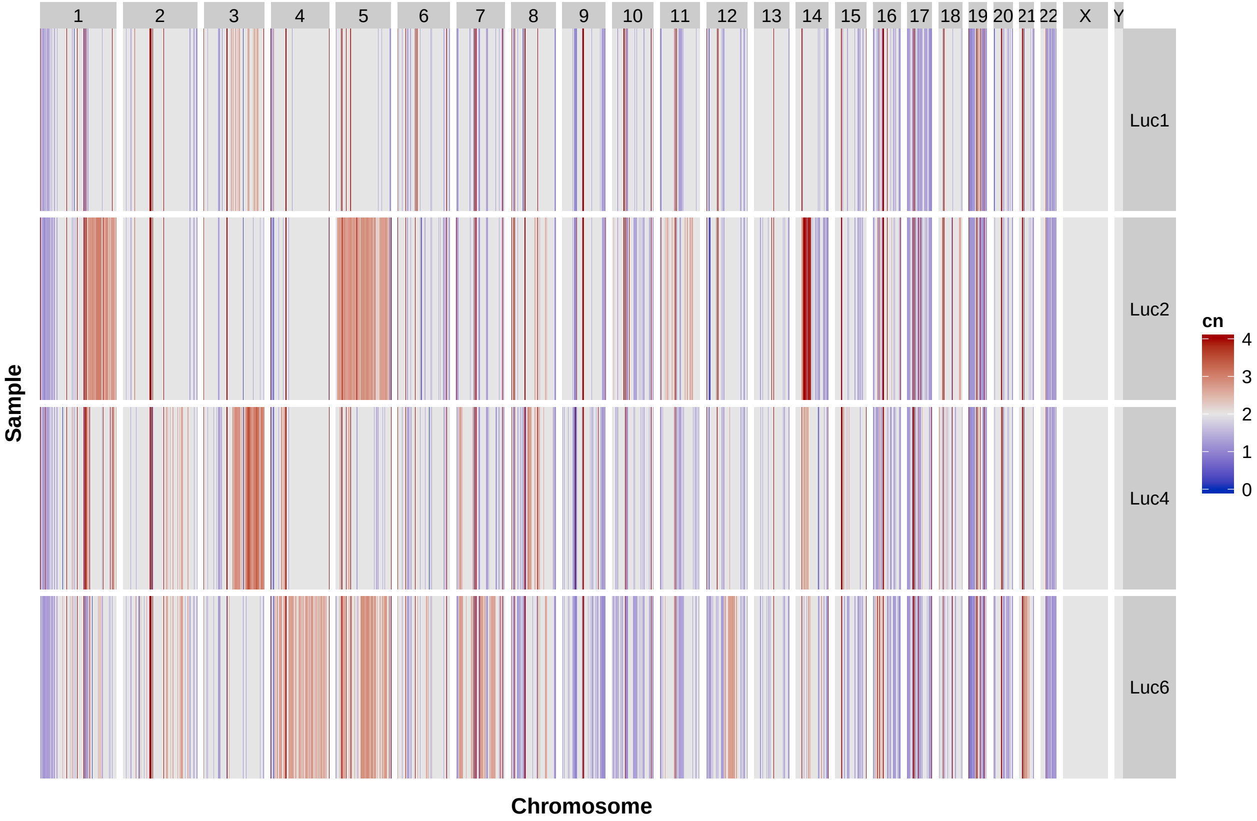


cnSpec will query the UCSC sql database to obtain chromosomal boundary information, this has been built in as a convenience. If internet connectivity is an issue, or if copy number segment calls are derived from an assemlby not supported by UCSC the user can specify chromosomal boundaries via the argument `y`. This should take the form of a data frame with column names “chromosome”, “start”, “end”.

```
# Example input to y
head(hg19chr)
```

chromosome start end 1 1 0 243700000 2 10 0 130600000 3 11 0 130800000 4 12 0 129300000 5 13 0 110300000 6 14 0 104000000

```
cnSpec(LucCN, y=hg19chr)
```



CN_plot

gene_plot

plot_coverage