

Introduction

T-ALL Example dataset

- 1) Obtain clinical, lesion and gene expression data
- 2) Retrieve Genomic Annotations for Genes and Regulatory Features
- 3) Retrieve Chromosome Size Data
- 4) Run Genomic Random Interval (GRIN) Analysis
- 5) Now, let's Take a Look on the GRIN Output Results:
- 6) Write GRIN Results
- 7) Genome-wide Lesion Plot
- 8) Stacked Barplot for a List of Genes of Interest
- 9) Prepare an OncoPrint Lesion Matrix
- 10) Pass the Lesion Matrix to OncoPrint Function
- 11) Prepare OncoPrint Lesion Matrix for Genes in a List of Selected Pathways
- 12) Pass the Lesion Matrix of Selected Pathways to the OncoPrint Function
- 13) Gene-lesion Plots with GRIN Statistics and Transcripts Track
- 14) Locus-lesion Plots with Transcripts Track
- 15) Locus-lesion Plots WITHOUT Transcripts Track
- 16) Chromosome Lesion Plots
- 17) Regulatory Features Lesion Plots
- 18) Gene-Lesion Matrix for later computations

Associate Lesions with EXpression (ALEX)

- 19) Prepare Expression and Lesion Data for ALEX-KW Test and ALEX-plots
- 20) Run Kruskal-Wallis Test for Association between Lesion and Expression Data
- 21) Now, let's Take a Look on the ALEX Kruskal-wallis Results Table:
- 22) Boxplots Showing Expression Level by Lesion Groups for Top Significant Genes
- 23) Prepare ALEX Data for Waterfall Plots
- 24) Return Waterfall Plots for Top Significant Genes
- 25) Run Association Analysis between Lesion and Expression Data on the Pathway Level (JAK/STAT Pathway)
- 26) Lesion Binary Matrix for Association Analysis with Clinical Outcomes
- 27) Run Association Analysis for Lesions with Clinical Outcomes

28) Now, let's Take a Look on the Results Table of the Association between Lesions and Treatment Outcomes:

29) Evaluate CNVs (Gain and Deletions) as Lesion Boundaries

30) Run GRIN analysis Using Lesion Boundaries Instead of the Gene Annotation File

31) Genome-wide Significance Plot for Loss Lesion Boundaries

32) Genome-wide Significance Plot for annotated Genes Affected by Deletions

Introduction to the GRIN2 Package

Abdelrahman Elsayed, PhD and Stanley Pounds, PhD

2024-10-29

Introduction

The **GRIN2** package is an improved version of GRIN software that streamlines its use in practice to analyze genomic lesion data, accelerate its computing, and expand its analysis capabilities to answer additional scientific questions including a rigorous evaluation of the association of genomic lesions with RNA expression.

T-ALL Example dataset

- Genomic Landscape of T-ALL (<https://pubmed.ncbi.nlm.nih.gov/28671688/>)
- RNA-seq and WES data for 265 patients identified 6,887 genomic lesions
- Clinical outcome data

1) Obtain clinical, lesion and gene expression data

```
data(clin.data)
data(lesion.data)
data(expr.data)

head(lesion.data)
#>      ID chrom  Loc.start   Loc.end Lsn.type
#> 1 PARFIH    16  67650782  67650782 mutation
#> 2 PARFIH     3  49759671  49759671 mutation
#> 3 PARFIH     1   6313912   6313912 mutation
#> 4 PARFIH     1 235929422 235929422 mutation
#> 5 PARFIH     4 142053653 142053653 mutation
#> 6 PARFIH     9  27206737  27206737 mutation

# Specify a directory on your local machine to store the analysis results:
# knitr::opts_knit$set(root.dir = normalizePath("folder path/test/"))
```

2) Retrieve Genomic Annotations for Genes and Regulatory Features

```
hg19.ann=get.ensembl.annotation("Human_GRCh37")
# "Human_GRCh38" can be used instead of "Human_GRCh37" to retrieve data for hg38

# 1) Gene annotation data that include around 20,000 coding genes and 25,000 Non-coding processed transcripts such as lncRNAs, miRNAs, snRNA and snoRNAs:
gene.annotation=hg19.ann$gene.annotation

# 2) Annotation data for regulatory features retrieved from ensembl regulatory build that include around 500,000 features (promoters, enhancer, TF and CTCF binding sites, etc...). Ensembl imports publicly available data from different large epigenomic consortia that includes ENCODE, Roadmap Epigenomics and Blueprint (118 epigenome):
hg19.reg.annotation=hg19.ann$reg.annotation.predicted

# 3) Annotation data for experimentally validated regulatory features retrieved from FANTOM 5 project:
hg19.reg.FANTOM=hg19.ann$reg.annotation.validated

# Instead of retrieving annotation data from Ensembl BioMart, users can use their own gene annotation data files. File should have four required columns that include "gene" which is the ensembl ID of annotated genes to which the lesion data will be overlapped, "chrom" which is the chromosome on which the gene is located, "loc.start" which is the gene start position, and "loc.end" the gene end position. hg19.gene annotation will be used as an example gene annotation data file:
data(hg19.gene.annotation)

head(hg19.gene.annotation)
#>           gene chrom loc.start  loc.end
#> 1 ENSG00000177076     9  19408925  19452018
#> 2 ENSG00000122729     9  32384618  32454767
#> 3 ENSG00000167107    17  48503519  48552206
#> 4 ENSG00000130402    19  39138289  39222223
#> 5 ENSG00000173137     8 145596790 145618457
#> 6 ENSG0000035687     1  244571796  244615436
#>                                     description
#> 1          alkaline ceramidase 2 [Source:HGNC Symbol;Acc:23675]
#> 2          aconitase 1, soluble [Source:HGNC Symbol;Acc:117]
#> 3 acyl-CoA synthetase family member 2 [Source:HGNC Symbol;Acc:26101]
#> 4          actinin, alpha 4 [Source:HGNC Symbol;Acc:166]
#> 5    aarF domain containing kinase 5 [Source:HGNC Symbol;Acc:21738]
#> 6    adenylosuccinate synthase [Source:HGNC Symbol;Acc:292]
#> gene.name      biotype chrom.strand chrom.band
#> 1    ACER2 protein_coding           1      p22.1
#> 2    ACO1 protein_coding           1      p21.1
#> 3    ACSF2 protein_coding           1      q21.33
#> 4    ACTN4 protein_coding           1      q13.2
#> 5    ADCK5 protein_coding           1      q24.3
#> 6    ADSS protein_coding          -1      q44
```

3) Retrieve Chromosome Size Data

```
# To retrieve chromosome size data for GRCh37 (hg19) genome build from chr.info txt file available on UCSC genome browser
hg19.chrom.size=get.chrom.length("Human_GRCh37")
# "Human_GRCh38" can be used to retrieve chrom size data for hg38

# Instead of retrieving chromosome size data from UCSC genome browser, users can use their own files that should has two required columns that include "chrom" with the chromosome number and "size" for the size of the chromosome in base pairs:
# data(hg19.chrom.size)

head(hg19.chrom.size)
#>   chrom      size
#> 1     1 2.49e+08
#> 2     2 2.43e+08
#> 3     3 1.98e+08
#> 4     4 1.91e+08
#> 5     5 1.81e+08
#> 6     6 1.71e+08
```

4) Run Genomic Random Interval (GRIN) Analysis

```
# Users can run GRIN analysis by just specifying the genome.version in grin.stats function.
# A) Gene annotation data will be directly retrieved from Ensembl BioMart for the specified genome assembly using get.ensembl.annotation function and chromosome size data will be also retrieved from UCSC genome browser:
grin.results=grin.stats(lesion.data,
                        genome.version="Human_GRCh37")
# "Human_GRCh38" can be used instead of "Human_GRCh37" for hg38 genome assembly

# Users can also use their own annotation and chromosome size data files to run GRIN analysis:
grin.results=grin.stats(lesion.data,
                        hg19.gene.annotation,
                        hg19.chrom.size)
# it takes around 2 minutes to map 6,887 lesions to around 57,000 annotated genes and return the GRIN results.

# B) To run GRIN for computationally predicted regulatory features from Ensembl regulatory build:
# First get a group of 25,000 regulatory features for an example run:
hg19.reg.example=hg19.reg.annotation[c(30000:35000, 75000:80000,
                                       140000:145000, 250000:255000,
                                       395000:400000),]
# whole file with around 500,000 feature takes around 25 minutes to return the results:
# Run GRIN analysis:
grin.results.reg=grin.stats(lesion.data,
                            hg19.reg.example,
                            hg19.chrom.size)

# C) To run GRIN analysis for experimentally verified regulatory features from FANTOM5 project:
# First get a group of 25,000 FANTOM5 regulatory features for an example run:
hg19.FANTOM.example=hg19.reg.FANTOM[c(30000:35000, 75000:80000,
                                       140000:145000, 230000:235000,
                                       395000:400000),]

grin.results.fantom=grin.stats(lesion.data,
                              hg19.FANTOM.example,
                              hg19.chrom.size)
```

5) Now, let's Take a Look on the GRIN Output Results:

```
# Extract GRIN results table:
grin.table=grin.results$gene.hits
sorted.results <- grin.table[order(as.numeric(as.character(grin.table$p2.nsubj))),]
```

First section of GRIN results table will include gene annotation in addition to the number of subjects affected by each type of lesions:

```
head(sorted.results[,c(7,11:14)])
#>      gene.name nsubj.fusion nsubj.gain nsubj.Loss
#> 273      PTEN           0           8          23
#> 219      MYB            4          26           2
#> 64      CDKN1B           0           1          26
#> 105      ETV6            2           2          21
#> 395      WT1            0           1           9
#> 86      DDX3X            2           1           3
#>      nsubj.mutation
#> 273              37
#> 219              13
#> 64               4
#> 105              7
#> 395              24
#> 86               4
```

Results will also include the probability (p) and FDR adjusted q-value for each gene to be affected by each type of lesion:

```
head(sorted.results[,c(7,19:22)])
#>      gene.name q.nsubj.fusion q.nsubj.gain q.nsubj.Loss
#> 273      PTEN      1.00e+00      1.0e+00      4.80e-35
#> 219      MYB      2.52e-09      3.2e-50      8.63e-01
#> 64      CDKN1B      1.00e+00      1.0e+00      1.10e-25
#> 105      ETV6      5.30e-03      1.0e+00      2.36e-16
#> 395      WT1      1.00e+00      1.0e+00      1.44e-06
#> 86      DDX3X      4.80e-05      1.0e+00      8.63e-01
#>      q.nsubj.mutation
#> 273      8.23e-77
#> 219      7.32e-27
#> 64      8.49e-09
#> 105      1.40e-06
#> 395      6.68e-50
#> 86      1.54e-06
```

Another important part of the output is the constellation results testing if the gene is affected by one type of lesions (p1.nsubj) or a constellation of two types of lesions (p2.nsubj), three types of lesions (p3.nsubj), etc.. with FDR adjusted q-values added to the table as well:

```
head(sorted.results[,c(7,27:30)])
#>      gene.name q1.nsubj q2.nsubj q3.nsubj q4.nsubj
#> 273      PTEN 6.48e-77 1.10e-69 1.00e+00      1
#> 219      MYB 4.73e-51 5.50e-53 4.51e-29      1
#> 64      CDKN1B 7.50e-26 2.50e-16 1.00e+00      1
#> 105      ETV6 3.03e-16 6.74e-12 1.23e-09      1
#> 395      WT1 6.38e-50 9.44e-11 1.00e+00      1
#> 86      DDX3X 6.23e-07 2.69e-10 1.00e+00      1
```

The second part of the results table report the same set of results but for the number of hits affecting each gene for each lesion type instead of the number of unique affected subjects. For example, if NOTCH1 gene is affected by 4 mutations in the same subject, this event will be counted as 4 hits in the n.hits stats but 1 subject in the n.subj stats:

```
head(sorted.results[,c(7,31:34)])
#>      gene.name nhit.fusion nhit.gain nhit.loss
#> 273      PTEN          0          8         38
#> 219      MYB           4         27          2
#> 64      CDKN1B          0          1         26
#> 105      ETV6           2          2         22
#> 395      WT1            0          1          9
#> 86      DDX3X           2          1          3
#>      nhit.mutation
#> 273             45
#> 219             14
#> 64              4
#> 105             7
#> 395            27
#> 86              5
```

6) Write GRIN Results

```
# write.grin.xlsx function return an excel file with multiple sheets that include GRIN res
ults table, interpretation of each column in the results, and methods paragraph
write.grin.xlsx(grin.results, "T-ALL_GRIN_result_annotated_genes.xlsx")

# To return the results table without other information (will be helpful in case of large
lesion data files where the gene.lsn.data sheet will be > 1 million rows that halt the wri
te.grin.xlsx function).
grin.res.table=grin.results$gene.hits
# write.xlsx(grin.res.table, "grin.res.table.xlsx")
```

7) Genome-wide Lesion Plot

```
genomewide.plot=genomewide.lsn.plot(grin.results,
                                     max.log10q=150)
```

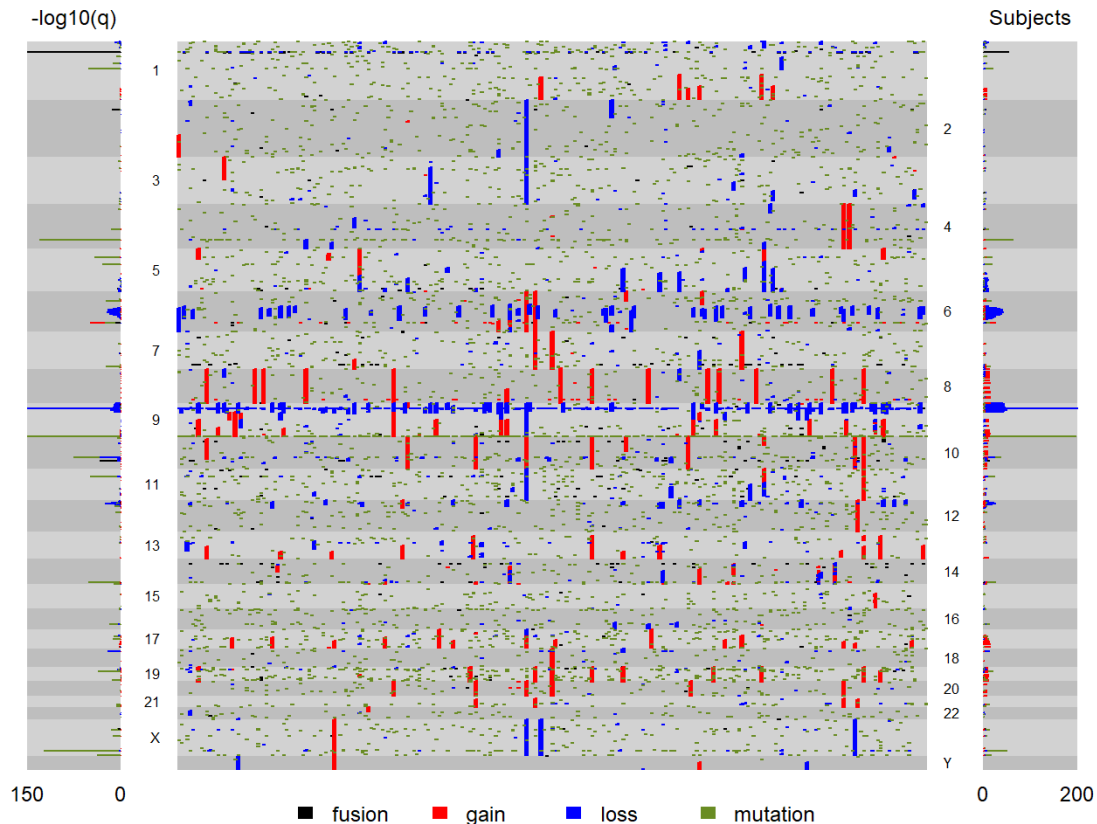


Figure 1. Genome-wide lesion plot

```
# This function use the list of grin.results
```

8) Stacked Barplot for a List of Genes of Interest

```
# This barplot shows the number of patients affected by different types of Lesions in a list of genes of interest:
```

```
count.genes=as.vector(c("CDKN2A", "NOTCH1", "CDKN2B", "TAL1", "FBXW7", "PTEN", "IRF8",
                        "NRAS", "BCL11B", "MYB", "LEF1", "RB1", "MLLT3", "EZH2", "ETV6",
                        "CTCF", "JAK1", "KRAS", "RUNX1", "IKZF1", "KMT2A", "RPL11", "TCF
7",
                        "WT1", "JAK2", "JAK3", "FLT3"))
```

```
# return the stacked barplot
grin.barplt(grin.results,
            count.genes)
```

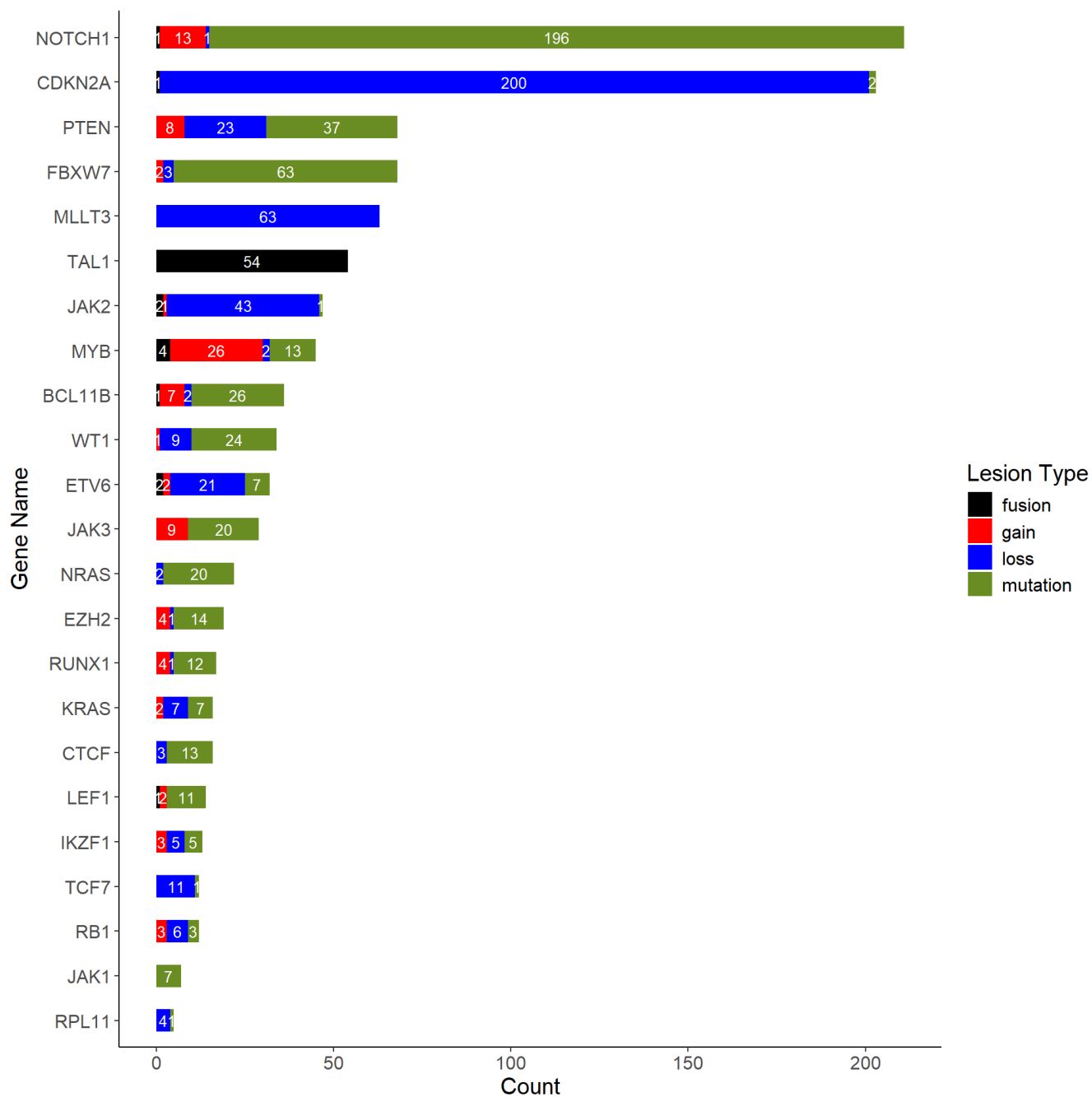



Figure 2. stacked barplot with number of patients affected by different types of lesions in a list of genes of interest

9) Prepare an OncoPrint Lesion Matrix

```
# First identify the list of genes to be included in the oncoprint:
oncoprint.genes=as.vector(c("ENSG00000101307", "ENSG00000171862", "ENSG00000138795",
                           "ENSG00000139083", "ENSG00000162434", "ENSG00000134371",
                           "ENSG00000118058", "ENSG00000171843", "ENSG00000139687",
                           "ENSG00000184674", "ENSG00000118513", "ENSG00000197888",
                           "ENSG00000111276", "ENSG00000258223", "ENSG00000187266",
                           "ENSG00000174473", "ENSG00000133433", "ENSG00000159216",
                           "ENSG00000107104", "ENSG00000099984", "ENSG00000078403",
                           "ENSG00000183150", "ENSG00000081059", "ENSG00000175354",
                           "ENSG00000164438"))

# Prepare a lesion matrix for the selected list of genes with each row as a gene and each
# column is a patient (this matrix is compatible with oncoPrint function in ComplexHeatmap package):
oncoprint.mtx=grin.oncoprint.mtx(grin.results,
                                oncoprint.genes)

head(oncoprint.mtx[,1:6])
#>      PASGFH  PASKSY  PASTDU PASYWF PATDRC PATFWF
#> TCF7      loss;    loss; mutation; loss; loss; loss;
#> KANK1                gain;
#> CDKN1B  loss;    loss;                loss; loss;
#> MYB                mutation;
#> LEF1
#> ETV6      loss;    loss;                loss; loss;
```

10) Pass the Lesion Matrix to OncoPrint Function

```
# Use onco.print.props function to specify a hgt for each lesion category to show all lesions
# that might affect a certain patient. For example, if the same patient is affected by gain
# and mutation, only 25% of the oncoprint rectangle will be filled with the mutations green
# color and the rest will appear as the gain red color.
onco.props<-onco.print.props(lesion.data,
                             hgt = c("gain"=5, "loss"=4, "mutation"=2, "fusion"=1))
column_title = "" # optional

# use oncoprint function from ComplexHeatmap library to plot the oncoprint:
oncoPrint(oncoprint.mtx,
          alter_fun = onco.props$alter_func,
          col = onco.props$col,
          column_title = column_title,
          heatmap_legend_param = onco.props$heatmap_legend_param)
```

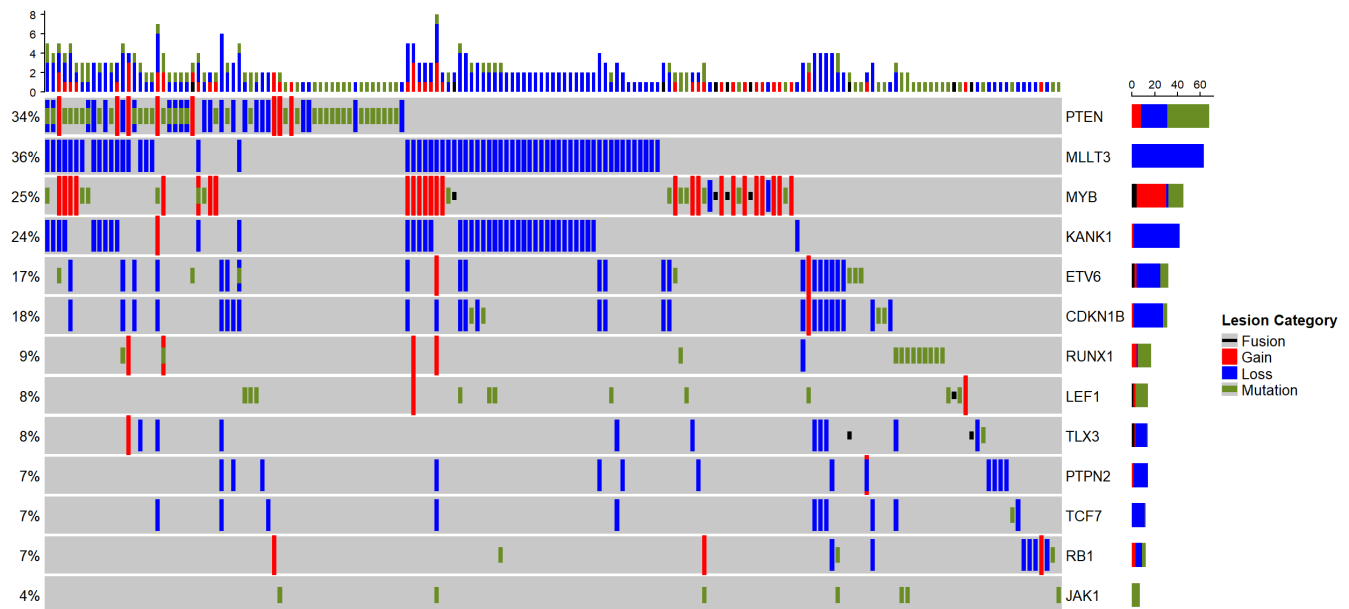


Figure 3. OncoPrint for a selected group of genes significant in the constelaation test for the gene to be affected by at least three types of lesions ($q3.nsubj < 0.05$)

11) Prepare OncoPrint Lesion Matrix for Genes in a List of Selected Pathways

```
# First we should call the pathways data file:
data(pathways)
head(pathways)
#> # A tibble: 6 × 3
#>   gene.name ensembl.id      pathway
#>   <chr>      <chr>      <chr>
#> 1 EBF1      ENSG00000164330 Bcell_Pathway
#> 2 IKZF1      ENSG00000185811 Bcell_Pathway
#> 3 RAG1      ENSG00000166349 Bcell_Pathway
#> 4 RAG2      ENSG00000175097 Bcell_Pathway
#> 5 CCND3      ENSG00000112576 CellCycle_Pathway
#> 6 CDKN1B      ENSG00000111276 CellCycle_Pathway

# define a list of pathways of interest:
PI3K_Pathway=pathways[pathways$pathway=="PI3K_Pathway",]
PI3K_ensembl=as.vector(PI3K_Pathway$ensembl.id)
Bcell_Pathway=pathways[pathways$pathway=="Bcell_Pathway",]
Bcell_ensembl=as.vector(Bcell_Pathway$ensembl.id)
Jak_Pathway=pathways[pathways$pathway=="Jak_Pathway",]
Jak_ensembl=as.vector(Jak_Pathway$ensembl.id)
Ras_Pathway=pathways[pathways$pathway=="Ras_Pathway",]
Ras_ensembl=as.vector(Ras_Pathway$ensembl.id)

oncoprint.genes=c(PI3K_ensembl, Bcell_ensembl, Jak_ensembl, Ras_ensembl)

# prepare the oncoprint matrix:
oncoprint.mtx.path=grin.oncoprint.mtx(grin.results,
                                     oncoprint.genes)
Gene=as.data.frame(rownames(oncoprint.mtx.path))
colnames(Gene)="gene.name"
Gene$index=1:nrow(Gene)
merged.df=merge(Gene,pathways, by="gene.name", all.x=TRUE)
merged.df=merged.df[order(merged.df$index), ]

sel.pathways=factor(merged.df$pathway,
                    levels=c("PI3K_Pathway", "Jak_Pathway", "Ras_Pathway", "Bcell_Pathway"))
```

12) Pass the Lesion Matrix of Selected Pathways to the OncoPrint Function

```
# Use onco.print.props function to specify a hgt for each lesion category to show all lesions that might affect a certain patient:
onco.props.path<-onco.print.props(lesion.data,
                                hgt = c("gain"=5, "loss"=4, "mutation"=2, "fusion"=1))

column_title = "" # optional

# use oncoprint function from complexheatmap library to plot the oncoprint
oncoPrint(oncoprint.mtx.path,
          alter_fun = onco.props.path$alter_func,
          col = onco.props.path$col,
          column_title = column_title,
          heatmap_legend_param = onco.props.path$heatmap_legend_param,
          row_split=spl.pathways)
```

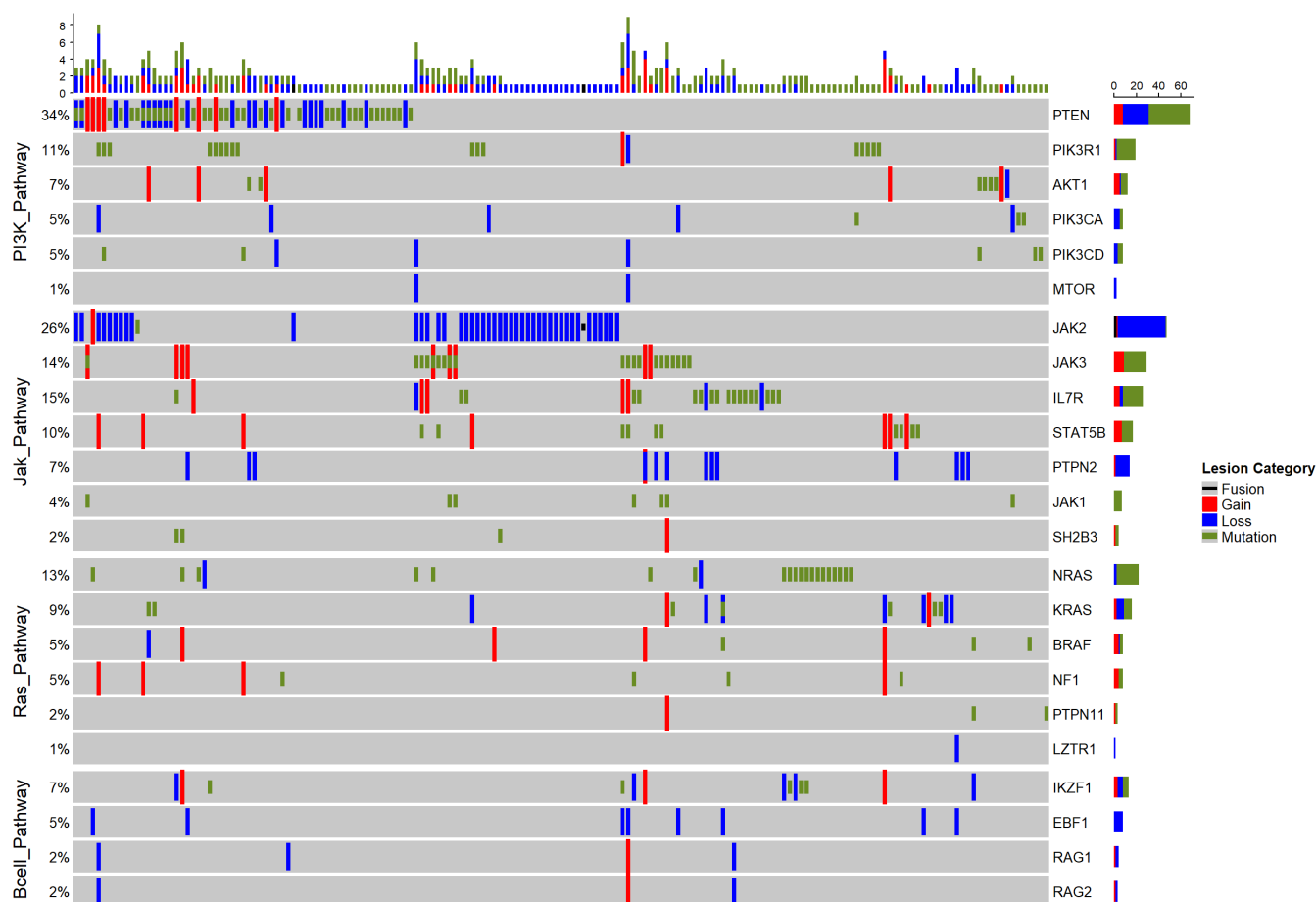


Figure 4. OncoPrint for genes annotated to a selected group of pathways

13) Gene-lesion Plots with GRIN Statistics and Transcripts Track

First we need to call "hg19_cytoband" and "hg38_cytoband" before calling the plot function:

```
data(hg19_cytoband)
data(hg38_cytoband)
```

lsn.transcripts.plot function can be used to generate a plot that shows all different types of lesions that affect a gene of interest with a transcripts track directly retrieved from Ensembl genome browser:

```
lsn.transcripts.plot(grin.results,
                    genome="hg19",
                    gene="WT1",
                    hg19.cytoband=hg19_cytoband)
```

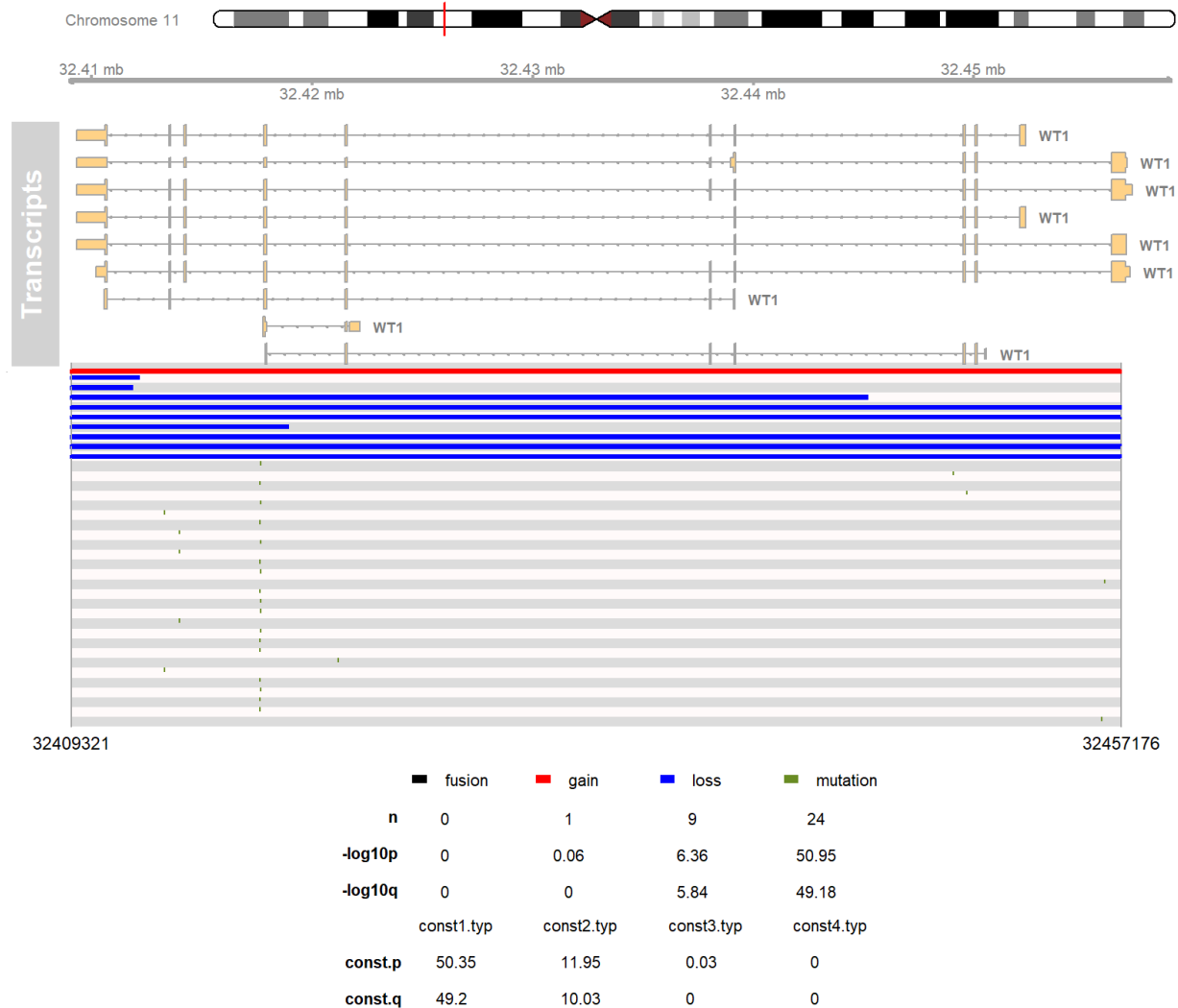


Figure 5. lesion plot showing all different types of lesions affecting WT1 gene with transcripts track directly retrieved from Ensembl database

```
# for hg38 genome assembly:
# ah <- AnnotationHub()
# retrieve gene transcripts for human GRCh38 genome assembly from Ensembl (version 110):
# gtf.V110 <- ah[["AH113665"]]

#lsn.transcripts.plot(grin.results,
#                      genome="hg38",
#                      gene="WT1",
#                      hg38.transcripts=gtf.V110,
#                      hg38.cytoband=hg38_cytoband)
```

14) Locus-lesion Plots with Transcripts Track

```
# lsn.transcripts.plot function can be also used to generate a plot for lesions of a specific lesion group that span a certain locus of interest with transcripts track added:
lsn.transcripts.plot(grin.results,
                     genome="hg19",
                     hg19.cytoband=hg19_cytoband,
                     chrom=9,
                     plot.start=21800000,
                     plot.end=22200000,
                     lesion.grp = "loss",
                     spec.lsn.clr = "blue")
```

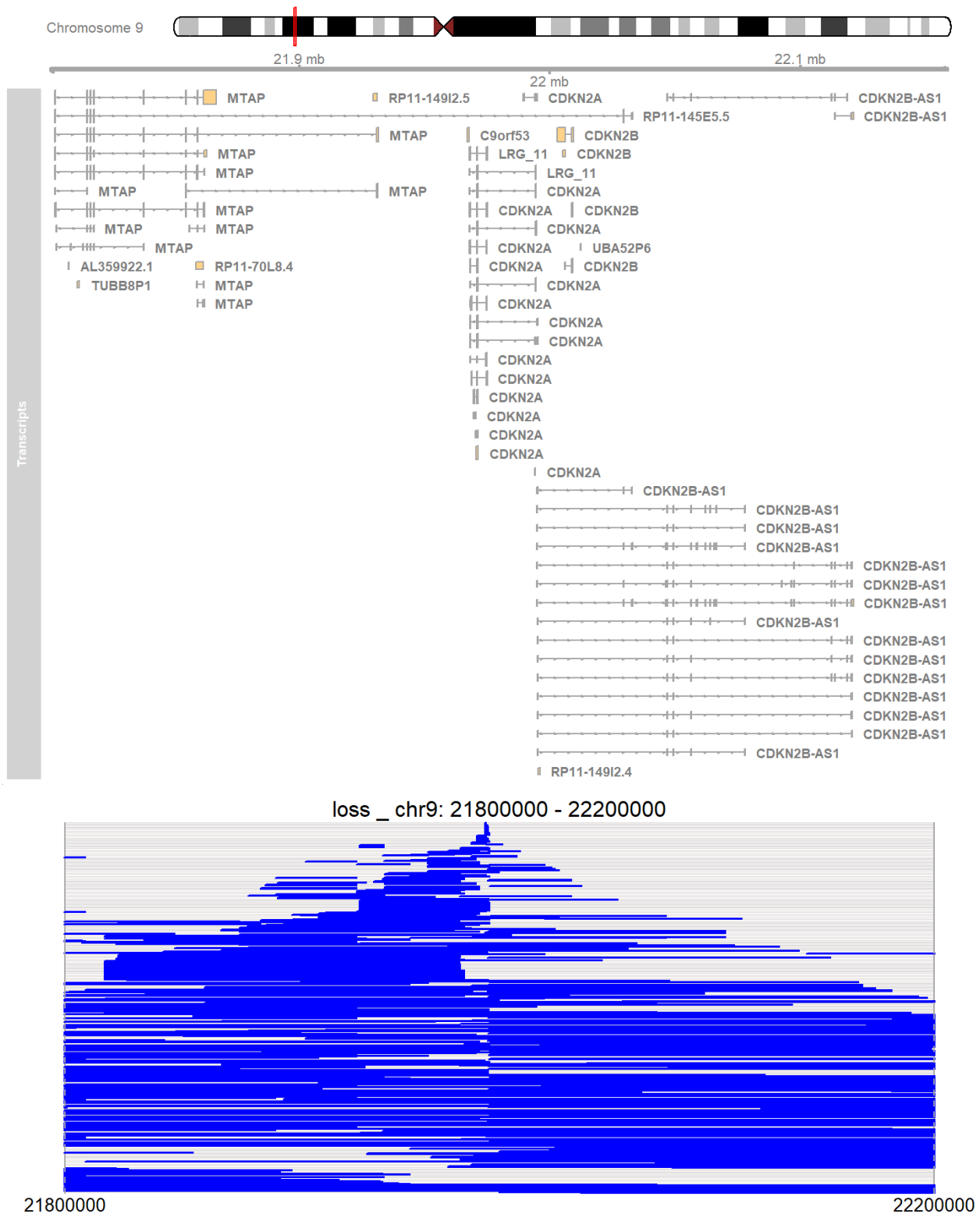


Figure 6. Regional lesion plot showing a specific type of lesion that affect a region of interest


```
# for hg38 genome assembly:
# ah <- AnnotationHub()
# retrieve gene transcripts for human GRCh38 genome assembly from Ensembl (version 110):
# gtf.V110 <- ah[["AH113665"]]

#lsn.transcripts.plot(grin.results,
#                      genome="hg38",
#                      hg38.transcripts="gtf.v110",
#                      hg38.cytoband=hg38_cytoband,
#                      chrom=9,
#                      plot.start=21800000,
#                      plot.end=22200000,
#                      lesion.grp = "loss",
#                      spec.lsn.clr = "blue")
```

15) Locus-lesion Plots WITHOUT Transcripts Track

chrom.lsn.plot function can be used to generate a plot for all lesions of a specific lesion type that affect a locus or region of interest without adding transcripts track. This will allow plotting a larger locus of the chromosome such as a chromosome band.

```
chrom.lsn.plot(grin.results,
               genome="hg19",
               hg19.cytoband=hg19_cytoband,
               chrom=9,
               plot.start=19900000,
               plot.end=25600000,
               lesion.grp = "loss",
               spec.lsn.clr = "blue")
```

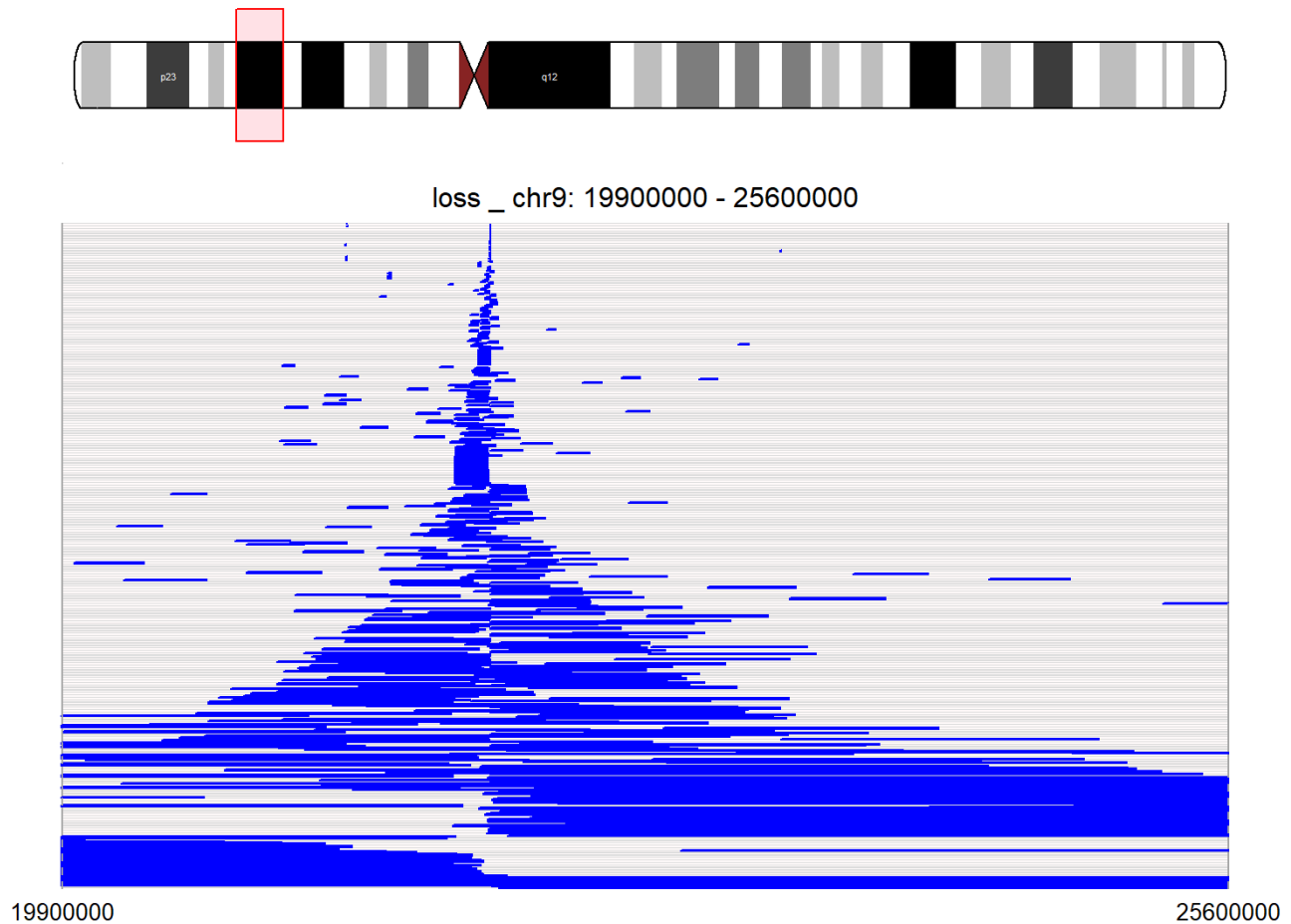


Figure 7. Regional lesion plot showing a specific type of lesion that affect a region of interest

```
# for hg38 genome assembly:
# ah <- AnnotationHub()
# retrieve gene transcripts for human GRCh38 genome assembly from Ensembl (version 110):
# gtf.V110 <- ah[["AH113665"]]

#chrom.Lsn.plot(grin.results,
#               genome="hg38",
#               hg38.transcripts="gtf.v110",
#               hg38.cytoband=hg38_cytoband,
#               chrom=9,
#               plot.start=19900000,
#               plot.end=25600000,
#               lesion.grp = "Loss",
#               spec.Lsn.clr = "blue")
```

16) Chromosome Lesion Plots

chrom.lsn.plot function can be also used to generate a plot for all types of lesions that affect a chromosome of interest with *plot.start=1* and *plot.end* is the chr size:

```
chrom.lsn.plot(grin.results,  
              genome="hg19",  
              hg19.cytoband=hg19_cytoband,  
              chrom=9,  
              plot.start=1,  
              plot.end=141000000)
```

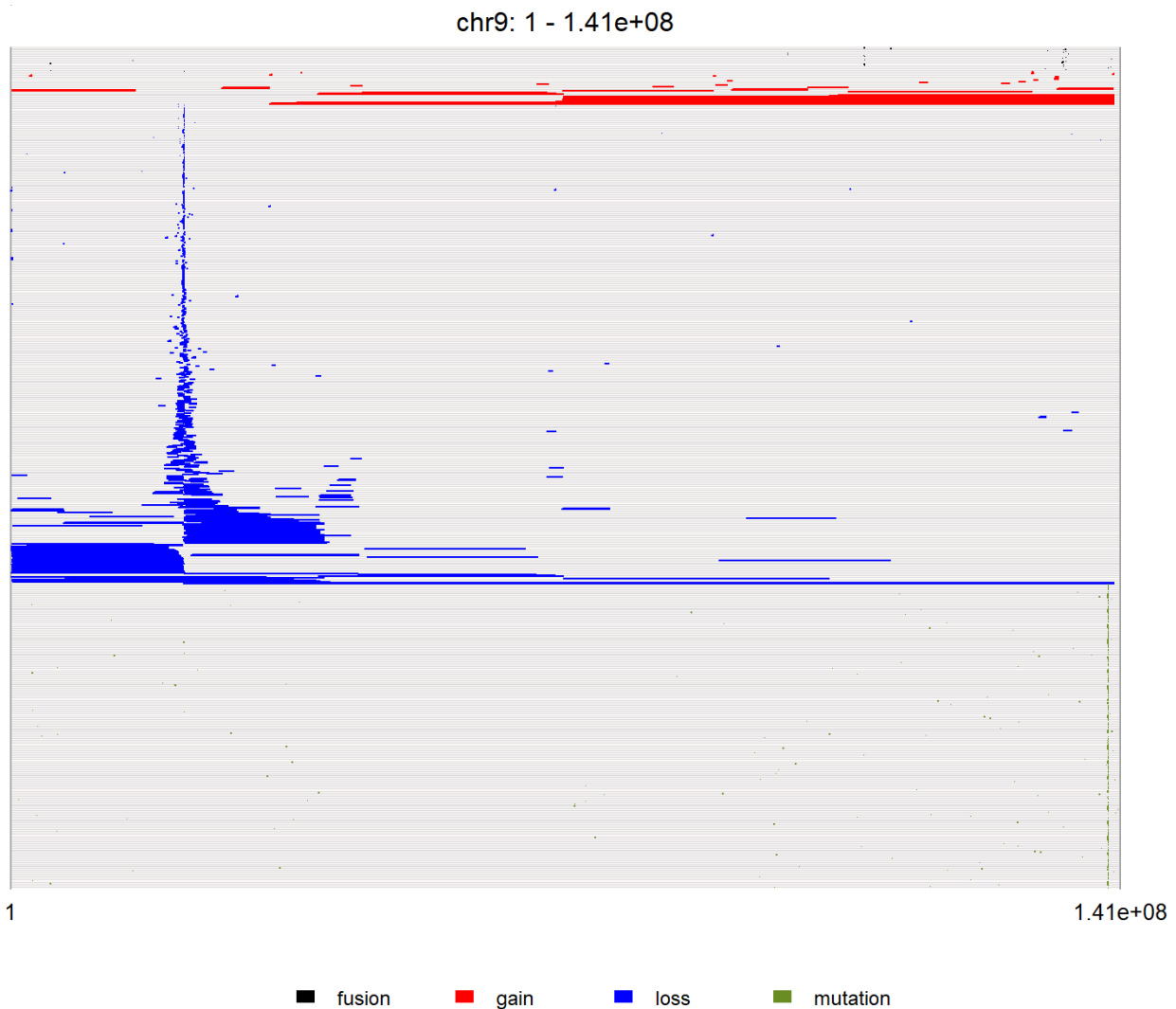


Figure 8. Lesion plot showing different types of lesions that affect a chromosome of interest

```
# for hg38 genome assembly:
#ah <- AnnotationHub()
#gtf.V110 <- ah[["AH113665"]]

#chrom.lsn.plot(grin.results,
#               genome="hg38",
#               hg38.transcripts="gtf.v110",
#               hg38.cytoband=hg38_cytoband,
#               chrom=9,
#               plot.start=1,
#               plot.end=141000000)
```

17) Regulatory Features Lesion Plots

```
# grin.stats.lsn.plot function can be used to generate plots that show all different types
# of lesions that affect a regulatory feature of interest in addition to the GRIN statistic
# s. Plot does not include transcripts track that's typically not available for those featur
# es.
# grin.stats.lsn.plot(grin.results.reg,
#                     feature="ENSR00000105619")

# Same plot can be also prepared for regulatory features from the FANTOM5 project (for exa
# mple: the NRAS promoter site affected by 18 mutations)
grin.stats.lsn.plot(grin.results.fantom,
                    feature="p6@NRAS,0.2452")
```

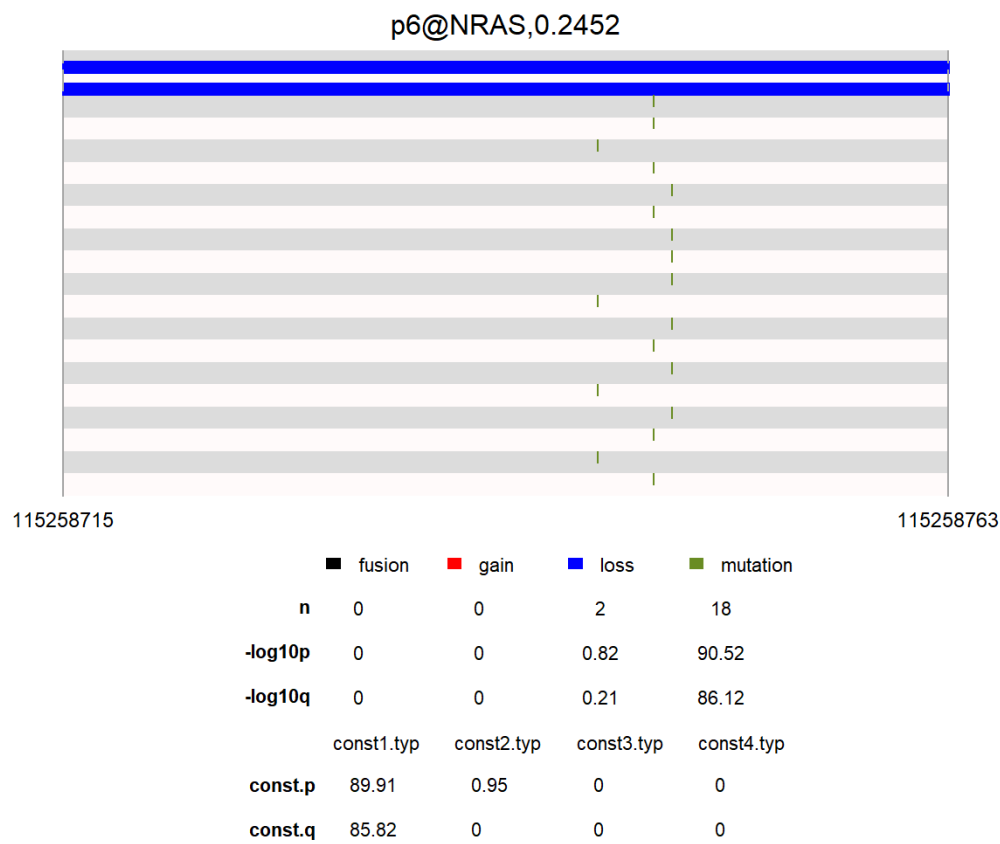


Figure 9. A plot that shows all different types of lesions that affect a regulatory feature of interests in addition the feature GRIN statistics

18) Gene-Lesion Matrix for later computations

```
# Prepare gene and lesion data for later computations
# This lesion matrix has all lesion types that affect a single gene in one row. It can be
used to run association analysis with expression data (part of alex.lsn.expr function)

# First step is to prepare gene and lesion data for later computations
gene.lsn=prep.gene.lsn.data(lesion.data,
                             hg19.gene.annotation)
# Then determine lesions that overlap each gene (Locus)
gene.lsn.overlap= find.gene.lsn.overlaps(gene.lsn)
# Finally, build the lesion matrix using prep.lsn.type.matrix function:
gene.lsn.type.mtx=prep.lsn.type.matrix(gene.lsn.overlap,
                                       min.ngrp=5)

# prep.lsn.type.matrix function return each gene in a row, if the gene is affected by multiple
types of lesions (for example gain AND mutations), entry will be denoted as "multiple"
for this specific patient.
# min.ngrp can be used to specify the minimum number of patients with a lesion to be included
in the final lesion matrix.

head(gene.lsn.type.mtx[,1:5])
#>                PARASZ  PARAYM  PARCVM  PAREGZ  PARFDL
#> ENSG00000005700 "Loss" "none" "Loss" "none" "none"
#> ENSG00000010810 "Loss" "none" "Loss" "none" "none"
#> ENSG00000014123 "Loss" "none" "Loss" "none" "none"
#> ENSG00000056972 "Loss" "none" "Loss" "none" "none"
#> ENSG00000057663 "Loss" "none" "Loss" "none" "none"
#> ENSG00000065615 "Loss" "none" "Loss" "none" "none"
```

Associate Lesions with EXpression (ALEX)

19) Prepare Expression and Lesion Data for ALEX-KW Test and ALEX-plots

alex.prep.lsn.expr function prepare expression, lesion data and return the set of genes with both types of data available ordered by gene IDs in rows and patient IDs in columns:

```
alex.data=alex.prep.lsn.expr(expr.data,
                             lesion.data,
                             hg19.gene.annotation,
                             min.expr=1,
                             min.pts.lsn=5)

# ALEX ordered Lesion data:
alex.lsn=alex.data$alex.lsn
head(alex.lsn[,1:5])
#>
#> ENSG00000005339  none  none  none  none  none
#> ENSG00000005700  Loss  none  Loss  none  none
#> ENSG00000006283  none  none  none  none  none
#> ENSG00000007047  none  none  none  none  none
#> ENSG00000010438  none  Loss  none  none  none
#> ENSG00000010810  Loss  none  Loss  none  none

# ALEX ordered expression data:
alex.expr=alex.data$alex.expr
head(alex.expr[,1:5])
#>
#> ENSG00000005339  4.012  3.72  3.253  2.294  3.568
#> ENSG00000005700  2.503  3.74  3.011  3.524  3.437
#> ENSG00000006283  0.035  0.00  0.016  0.005  0.043
#> ENSG00000007047  2.479  2.50  2.447  2.381  2.255
#> ENSG00000010438  0.155  0.00  0.454  0.000  0.153
#> ENSG00000010810  3.992  4.40  3.985  3.934  4.443
```

20) Run Kruskal-Wallis Test for Association between Lesion and Expression Data

KW.hit.express function runs Kruskal-Wallis test for association between lesion groups and expression level of the same corresponding gene:

```
alex.kw.results=KW.hit.express(alex.data,
                                hg19.gene.annotation,
                                min.grp.size=5)

# write KW test results:
write.csv(alex.kw.results, "alex.kw.results.csv")
```

21) Now, let's Take a Look on the ALEX Kruskal-wallis Results Table:

```
# order the genes by the ones with most significant KW q-value:
sorted.kw <- alex.kw.results[order(as.numeric(as.character(alex.kw.results$q.KW))),]
```

First section of the results table will include gene annotation in addition to the kruskal-wallis test p and q values evaluating if there's a statistically significant differences in the gene expression level between different lesion groups:

```
head(sorted.kw[,c(6,7,11,12)])
#>           gene.name      biotype      p.KW
#> ENSG00000198642    KLHL9 protein_coding 7.14e-21
#> ENSG00000120159    CAAP1 protein_coding 9.50e-20
#> ENSG00000162367     TAL1 protein_coding 1.15e-18
#> ENSG00000137073    UBAP2 protein_coding 4.54e-18
#> ENSG00000165282     PIGO protein_coding 3.88e-17
#> ENSG00000107185     RGP1 protein_coding 1.87e-16
#>           q.KW
#> ENSG00000198642 2.60e-18
#> ENSG00000120159 1.73e-17
#> ENSG00000162367 1.40e-16
#> ENSG00000137073 4.14e-16
#> ENSG00000165282 2.82e-15
#> ENSG00000107185 8.53e-15
```

For each gene, results table will include the number of patients affected by each type of lesion in addition to number of patients affected by multiple types of lesions in the same gene and patients without any lesion:


```
head(sorted.kw[,c(13:18)])
#>                fusion_n.subjects gain_n.subjects
#> ENSG00000198642                0                0
#> ENSG00000120159                0                1
#> ENSG00000162367               54                0
#> ENSG00000137073                0                1
#> ENSG00000165282                0                1
#> ENSG00000107185                0                1
#>                loss_n.subjects multiple_n.subjects
#> ENSG00000198642               99                0
#> ENSG00000120159              50                0
#> ENSG00000162367                0                0
#> ENSG00000137073              42                0
#> ENSG00000165282              42                0
#> ENSG00000107185              42                0
#>                mutation_n.subjects none_n.subjects
#> ENSG00000198642                0             164
#> ENSG00000120159                0             212
#> ENSG00000162367                0             209
#> ENSG00000137073                0             220
#> ENSG00000165282                0             220
#> ENSG00000107185                0             220
```

Results table will also include the mean expression of the gene by different lesion groups in addition to the median expression and standard deviation.

```
head(sorted.kw[,c(19:24)])
#>                fusion_mean gain_mean loss_mean
#> ENSG00000198642          NA          NA      1.89
#> ENSG00000120159          NA      3.78      2.65
#> ENSG00000162367      3.49          NA          NA
#> ENSG00000137073          NA      3.69      2.74
#> ENSG00000165282          NA      2.83      1.45
#> ENSG00000107185          NA      3.07      1.79
#>                multiple_mean mutation_mean none_mean
#> ENSG00000198642          NA          NA      3.06
#> ENSG00000120159          NA          NA      3.32
#> ENSG00000162367          NA          NA      1.58
#> ENSG00000137073          NA          NA      3.45
#> ENSG00000165282          NA          NA      2.01
#> ENSG00000107185          NA          NA      2.54
```

22) Boxplots Showing Expression Level by Lesion Groups for Top Significant Genes

```
# First open a PDF file to add boxplots, one gene per page
pdf("KW-q1e-15-boxplots.pdf", width = 8,height = 5, onefile = TRUE)
# add boxplots based on a specified KW.q test results:
alex.boxplots(alex.data,
              alex.kw.results,
              1e-15,
              hg19.gene.annotation)

dev.off()
#> png
#> 2
```

23) Prepare ALEX Data for Waterfall Plots

```
# waterfall plots allow a side-by-side representation of expression and lesion data of the
gene of interest.
# First prepare expression and lesion data for waterfall plots:
WT1.waterfall.prep=alex.waterfall.prep(alex.data,
                                       alex.kw.results,
                                       "WT1",
                                       lesion.data)

# alex.waterfall.plot can be used to return the plot
WT1.waterfall.plot=alex.waterfall.plot(WT1.waterfall.prep,
                                       lesion.data)
```

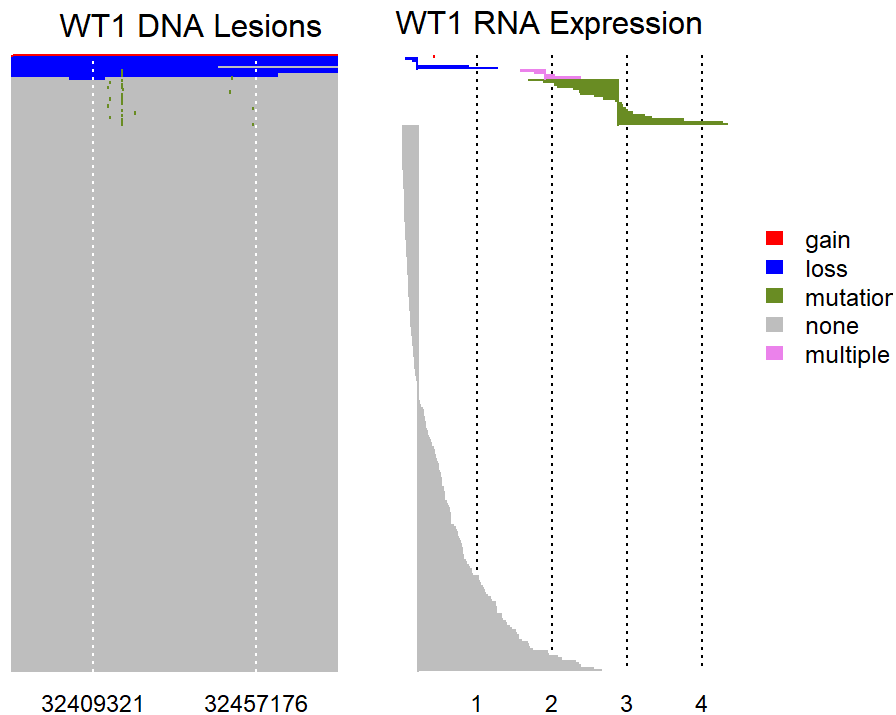


Figure 10. JAK2 Water-fall plot which offers a side-by-side graphical representation of lesion and expression data for each patient

24) Return Waterfall Plots for Top Significant Genes

To prepare Waterfall plots for top significant genes in the KW Results Table, users can use `top.alex.waterfall.plots` function by specifying a directory to store the plots, and minimum KW.q, for example:

```
# top.alex.waterfall.plots("folder path/test/waterfall_plots/",
#                          alex.data,
#                          alex.kw.results,
#                          1e-15,
#                          lesion.data)
```

25) Run Association Analysis between Lesion and Expression Data on the Pathway Level (JAK/STAT Pathway)

```
# alex.pathway function will run association analysis between lesion and expression data f
or all genes in a specified pathway (example: JAK/STAT pathway).
# Function will return two panels figure of lesion and expression data of ordered subjects
based on the computed lesions distance in all genes assigned to the pathway of interest:
alex.path=alex.pathway(alex.data,
                        lesion.data,
                        pathways,
                        "Jak_Pathway")
```

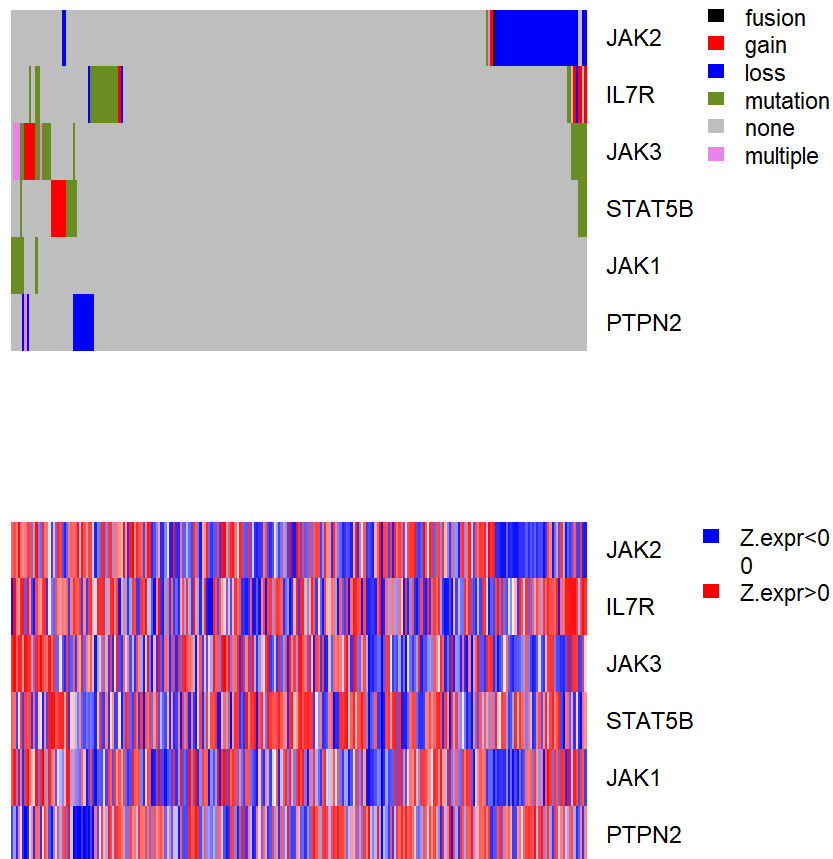


Figure 11. Ordered Lesion and Expression Data based on the Clustering Analysis on the pathway level (JAK/STAT pathway)

```
# To return ordered lesion and expression data of the genes assigned to the pathway of interest (same patients order in the plot):
```

```
write.csv(alex.path, "ordered-lesion-expression-data-JAK-pathway-genes.csv")
```

```
alex.path[1:10,1:5]
```

```
#>           PARSET  PATZWA  PATITB  PATYJK
#> JAK2 _lsn      none     none     none     none
#> JAK3 _lsn      none multiple multiple multiple
#> JAK1 _lsn  mutation mutation mutation mutation
#> IL7R _lsn      none     none     none     none
#> STAT5B _lsn    none     none     none     none
#> PTPN2 _lsn     none     none     none     none
#> JAK2 _expr    2.772    2.949    2.698    4.288
#> JAK3 _expr    4.566    5.828    4.186    4.96
#> JAK1 _expr    5.106    5.817    4.57     6.296
#> IL7R _expr    1.241    7.267    4.613    4.368
#>           PASHDV
#> JAK2 _lsn      none
#> JAK3 _lsn  mutation
#> JAK1 _lsn  mutation
#> IL7R _lsn      none
#> STAT5B _lsn mutation
#> PTPN2 _lsn     none
#> JAK2 _expr    2.715
#> JAK3 _expr    4.722
#> JAK1 _expr    4.633
#> IL7R _expr    5.775
```

26) Lesion Binary Matrix for Association Analysis with Clinical Outcomes

This type of lesion matrices with each gene affected by a certain type of lesion in a separate row is very helpful to run multiple levels of association analysis that include association between lesions and treatment outcomes.

Users should first Prepare gene and Lesion data and determine lesions that overlap each gene (Locus):

```
gene.lsn=prep.gene.lsn.data(lesion.data,
                             hg19.gene.annotation)
gene.lsn.overlap= find.gene.lsn.overlaps(gene.lsn)
```

use prep.binary.lsn.mtx function to prepare the Lesion binary matrix:

```
lsn.binary.mtx.atleast5=prep.binary.lsn.mtx(gene.lsn.overlap,
                                              min.ngrp=5)
```

Each row is a Lesion type that affect a certain gene for example NOTCH1_mutation (entry will be labelled as 1 if the patient is affected by this type of lesion and 0 otherwise).

min.ngrp can be used to specify the minimum number of patients with a Lesion to be included in the final Lesion matrix.

```
head(lsn.binary.mtx.atleast5[,1:5])
#>                PATXKW PASHNK PARMUC PATKWU
#> ENSG00000005339_mutation      0      1      1      1
#> ENSG00000005700_Loss          0      0      0      0
#> ENSG00000006283_gain          0      0      0      0
#> ENSG00000007047_gain          0      0      0      0
#> ENSG00000010438_Loss          0      0      0      0
#> ENSG00000010810_Loss          0      0      0      0
#>                PARXMV
#> ENSG00000005339_mutation      1
#> ENSG00000005700_Loss          0
#> ENSG00000006283_gain          0
#> ENSG00000007047_gain          0
#> ENSG00000010438_Loss          0
#> ENSG00000010810_Loss          0
```

27) Run Association Analysis for Lesions with Clinical Outcomes

```
# Prepare Event-free Survival (EFS) and Overall Survival (OS) as survival objects:
clin.data$EFS <- Surv(clin.data$efs.time, clin.data$efs.censor)
clin.data$OS <- Surv(clin.data$os.time, clin.data$os.censor)

# List all clinical variables of interest to be included in the association analysis:
clinvars=c("MRD.binary", "EFS", "OS")

# Run association analysis between lesions and clinical variables:
assc.outcomes=grin.assoc.lsn.outcome(lsn.binary.mtx.atleast5,
                                     clin.data,
                                     hg19.gene.annotation,
                                     clinvars)

# Write the analysis results:
write.csv(assc.outcomes, "associate.lsn.outcomes.csv")

# Run models adjusted for one or a group of covariates:
assc.outcomes.adj=grin.assoc.lsn.outcome(lsn.binary.mtx.atleast5,
                                         clin.data,
                                         hg19.gene.annotation,
                                         clinvars,
                                         covariate="Sex")

write.csv(assc.outcomes.adj, "associate.lsn.outcomes.adj.csv")
```

28) Now, let's Take a Look on the Results Table of the Association between Lesions and Treatment Outcomes:

```
# order the genes by the ones with most significant KW q-value:
sorted.outcomes <- assc.outcomes[order(as.numeric(as.character(assc.outcomes$`MRD.binary.p
-value`))),]
```

First section of the results table will include gene annotation in addition to the odds ratio, lower95, upper95 confidence intervals in addition to p and FDR adjusted q-values for the logistic regression models testing for the association between lesions and binary outcome variables such as Minimal Residual Disease (MRD). COX proportional hazard models will be used in case of survival objects such as Event-free survival (EFS) and Overall Survival (OS) with hazard ratios reported instead of odds ratio:

```

head(sorted.outcomes[1:7,c(6,11,14,15)])
#>                                     gene.name
#> ENSG00000147889_Loss                CDKN2A
#> ENSG00000184937_mutation            WT1
#> ENSG00000099810_Loss                MTAP
#> ENSG00000198642_Loss                KLHL9
#> ENSG00000171843_Loss                MLLT3
#> ENSG00000188352_Loss                FOCAD
#>                                     MRD.binary.odds.ratio
#> ENSG00000147889_Loss                0.213
#> ENSG00000184937_mutation            6.889
#> ENSG00000099810_Loss                0.311
#> ENSG00000198642_Loss                0.321
#> ENSG00000171843_Loss                0.225
#> ENSG00000188352_Loss                0.302
#>                                     MRD.binary.p-value
#> ENSG00000147889_Loss                5.89e-07
#> ENSG00000184937_mutation            2.70e-05
#> ENSG00000099810_Loss                7.07e-05
#> ENSG00000198642_Loss                6.11e-04
#> ENSG00000171843_Loss                1.06e-03
#> ENSG00000188352_Loss                1.33e-03
#>                                     MRD.binary.q-value
#> ENSG00000147889_Loss                0.00012
#> ENSG00000184937_mutation            0.00276
#> ENSG00000099810_Loss                0.00482
#> ENSG00000198642_Loss                0.02498
#> ENSG00000171843_Loss                0.02498
#> ENSG00000188352_Loss                0.02498

```

Results table will also include the number of patients with/without lesion who experienced or did not experience the event:


```

head(sorted.outcomes[1:7,c(6, 16:19)])
#>                                     gene.name
#> ENSG00000147889_loss                CDKN2A
#> ENSG00000184937_mutation            WT1
#> ENSG00000099810_loss                MTAP
#> ENSG00000198642_loss                KLHL9
#> ENSG00000171843_loss                MLLT3
#> ENSG00000188352_loss                FOCAD
#>                                     MRD.binary.event.with.Lsn
#> ENSG00000147889_loss                37
#> ENSG00000184937_mutation            16
#> ENSG00000099810_loss                36
#> ENSG00000198642_loss                14
#> ENSG00000171843_loss                6
#> ENSG00000188352_loss                10
#>                                     MRD.binary.event.without.Lsn
#> ENSG00000147889_loss                33
#> ENSG00000184937_mutation            54
#> ENSG00000099810_loss                34
#> ENSG00000198642_loss                56
#> ENSG00000171843_loss                64
#> ENSG00000188352_loss                60
#>                                     MRD.binary.no.event.with.Lsn
#> ENSG00000147889_loss                163
#> ENSG00000184937_mutation            8
#> ENSG00000099810_loss                150
#> ENSG00000198642_loss                85
#> ENSG00000171843_loss                57
#> ENSG00000188352_loss                69
#>                                     MRD.binary.no.event.without.Lsn
#> ENSG00000147889_loss                31
#> ENSG00000184937_mutation            186
#> ENSG00000099810_loss                44
#> ENSG00000198642_loss                109
#> ENSG00000171843_loss                137
#> ENSG00000188352_loss                125

```

29) Evaluate CNVs (Gain and Deletions) as Lesion Boundaries

This analysis is lesion type specific and covers the entire genome. It's meant to cover and assess the regions without any annotated genes or regulatory features. The first boundary for each chromosome will start from the first nucleotide base on the chromosome till the start position of the first lesion that affect the chromosome. Similarly, the last boundary will start from the end position of the last lesion that affect the chromosome till the last base on the chromosome.

First extract data for gains and deletions from the lesion data file:

```
gain=lesion.data[lesion.data$lsn.type=="gain",]
loss=lesion.data[lesion.data$lsn.type=="loss",]
```

Then use grin.lsn.boundaries function to return the lesion boundaries:

```
lsn.bound.gain=grin.lsn.boundaries(gain, hg19.chrom.size)
lsn.bound.loss=grin.lsn.boundaries(loss, hg19.chrom.size)
```

It return a table of ordered boundaries based on the unique start and end positions of different lesions in a specific category on each chromosome.

```
head(lsn.bound.loss[,1:5])
#>           gene chrom loc.start loc.end   diff
#> 1      chr1_1_51585      1         1  51585  51584
#> 2  chr1_51586_593440      1    51586 593440 541854
#> 3  chr1_593441_711152      1   593441 711152 117711
#> 4  chr1_711153_713167      1   711153 713167   2014
#> 5  chr1_713168_751594      1   713168 751594   38426
#> 6 chr1_751595_2247723      1   751595 2247723 1496128
```

30) Run GRIN analysis Using Lesion Boundaries Instead of the Gene Annotation File

```
grin.results.gain.bound=grin.stats(gain,
                                   lsn.bound.gain,
                                   hg19.chrom.size)
write.grin.xlsx(grin.results.gain.bound, "GRIN.results.gain.boundaries.final.xlsx")

grin.results.loss.bound=grin.stats(loss,
                                   lsn.bound.loss,
                                   hg19.chrom.size)
write.grin.xlsx(grin.results.loss.bound, "GRIN.results.loss.boundaries.final.xlsx")
```

31) Genome-wide Significance Plot for Loss Lesion Boundaries

genomewide.log10q.plot function will return a genome-wide plot based on $-\log_{10}$ q-value testing if each of the evaluated lesion boundaries is significantly affected by a deletions in our example:

```
genomewide.log10q.plot(grin.results.loss.bound,
                        lsn.grps=c("loss"),
                        lsn.colors=c("loss" = "blue"),
                        max.log10q = 50)
```

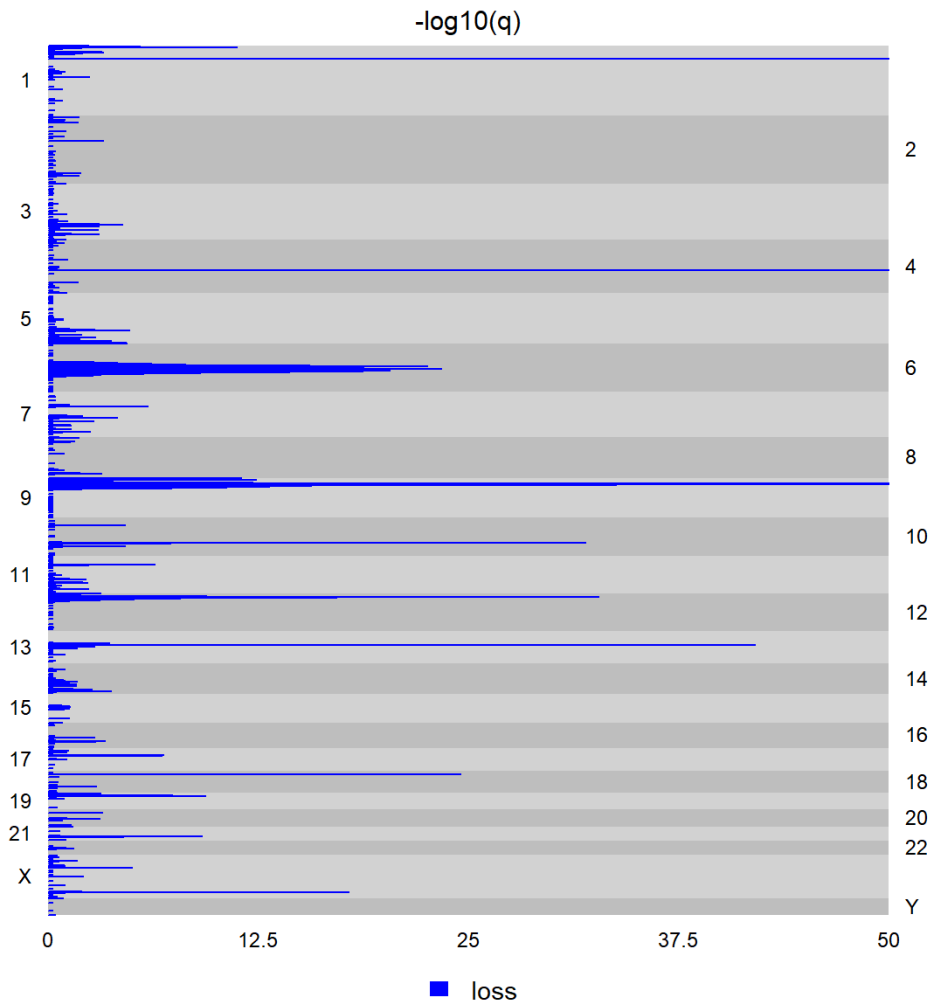


Figure 12. Genome-wide $-\log_{10}q$ plot of loss lesion boundaries

32) Genome-wide Significance Plot for annotated Genes Affected by Deletions

```
# genomewide.log10q.plot function can be also used to return genome-wide significance plot
# for annotated genes to be affected by a certain type of lesions.
# Here we should use GRIN results for annotated genes affected by loss instead of lesion boundaries.
# Users can notice that some regions mostly without annotated markers were only captured
# in the lesion boundaries analysis that cover the entire genome:
genomewide.log10q.plot(grin.results,
                      lsn.grps=c("loss"),
                      lsn.colors=c("loss" = "blue"),
                      max.log10q = 50)
```

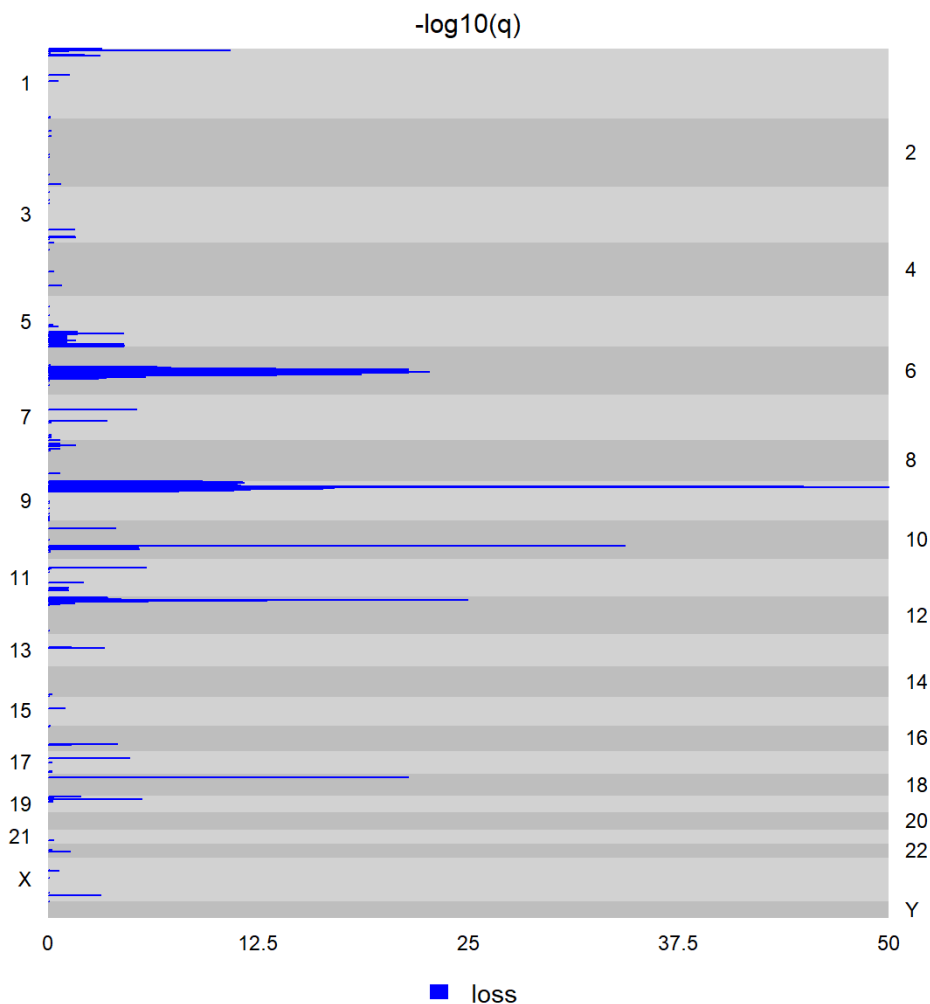


Figure 13. Genome-wide -log10q plot for annotated genes affected by deletions

```
library(GRIN2)
```