

cnvGSA: Gene-Set Analysis of (Rare) Copy Number Variants

Daniele Merico, Robert Ziman and Joseph Lugo

The Centre for Applied Genomics

daniele.merico@gmail.com, robert.ziman@gmail.com, joseph.r.lugo@gmail.com

March 20, 2015

Contents

| | | |
|----------|---|-----------|
| 1 | Overview | 2 |
| 2 | Workflow outline | 3 |
| 3 | Loading input data | 4 |
| 3.1 | CNV data | 5 |
| 3.2 | Phenotype/Covariate Data | 6 |
| 3.3 | Gene sets | 7 |
| 3.3.1 | Marking Known Loci | 8 |
| 4 | Loading test parameters from a file | 9 |
| 4.1 | Configuring test parameters | 11 |
| 5 | Full workflow example: case-control analysis of rare CNVs from the Pinto et al. 2014 ASD study | 14 |
| 5.1 | Loading the data and running the association test | 14 |
| 5.2 | Reviewing the results | 15 |
| 5.2.1 | Res.ls results | 16 |
| 5.2.2 | Detailed analysis of gene-set associations | 18 |
| 5.2.3 | Visualizations | 19 |
| 5.2.4 | Enrichment Map Pre-Processing | 22 |
| 6 | References | 24 |

1 Overview

cnvGSA is an R package for testing the gene-set rare variant burden in case-control studies of copy number variation (CNV).

Gene-sets need to be pre-compiled based on user-curated data or publicly available gene annotations (Gene Ontology, pathways, etc...)

”Competitive” gene-set over-representation tests are commonly used to analyze differentially gene expressed genes (e.g. Fisher’s Exact Test, GSEA), but they are not suitable for rare CNV [1][2]; the most appropriate choice for rare CNV is a ”self-contained” burden test with global burden correction such as implemented by **cnvGSA** or other tools [2].

Global burden correction is important because, for many disorders implicating rare CNV (autism, schizophrenia, etc...), case subjects are enriched in large, recurrent CNV that are not observed (or observed at very low frequency) in controls, and in which only a minority of genes contribute to disease risk [3][4][5][6]. In the absence of global burden correction, many gene-sets would present a biologically unspecific burden, uniquely driven by those larger and recurrent CNV [2]. Global burden correction thus helps identifying specific pathways and functional categories implicated in disease risk by rare CNV.

In **cnvGSA**, subjects are treated as statistical sampling units. Subject-level covariates that may act as confounders can be provided by the user (sex, ethnicity, CNV genotyping platform, CNV genotyping site, array quality metrics, etc...). The gene-set burden is tested using a logistic regression approach. Two logistic regression models are fit: model A includes the subject-level covariates (described above) and a variable quantifying global CNV burden for each subject (total CNV length, or total number of CNV-overlapped genes per subject, etc...); model B includes all variables present in model A, plus the number of CNV-overlapped genes that are members of the gene-set being tested. Presence of significantly higher burden in cases compared to controls is tested by comparing the two models using a deviance test, as implemented by `anova.glm`.

Note: only rare CNV (e.g. at frequency <1%) should be present in the input data. When analyzing the contribution of common variants to disorders with a genetic basis, a burden test is not the best approach. Earlier versions of **cnvGSA**, or highly similar burden tests, have been used in several high profile studies of rare CNV in neuropsychiatric disorders [2][4][5][6]. **cnvGSA** provides functions to visualize gene-set burden results and also to export them to the Cytoscape plugin Enrichment Map [7].

2 Workflow outline

The general procedure for performing a CNV gene-set analysis involves loading CNV, phenotype/covariate and gene-set data, setting filters and parameters, running the analysis, and reviewing the results. To facilitate these operations, the package provides `"CnvGSAInput"`, an S4 class acting as a simple container data structure with slots for each of these required inputs:

```
> library("cnvGSA")
> slotNames("CnvGSAInput")
[1] "cnvData.ls" "phData.ls" "gsData.ls" "config.ls" "params.ls"
```

The input slots should hold the following:

- `cnvData.ls` - CNV data
- `phData.ls` - Phenotype/Covariate data
- `gsData.ls` - Gene-set data
- `config.ls` - paths and file names
- `params.ls` - Test parameters

To ease the discussion here, a pre-built input object has been saved for convenience in the companion data package for this vignette:

```
> library("cnvGSAdata")
> data("cnvGSA_input_example")
> ls()

[1] "cnvGSA.in"

> class(cnvGSA.in)

[1] "CnvGSAInput"
attr(,"package")
[1] "cnvGSA"

> slotNames(cnvGSA.in)

[1] "cnvData.ls" "phData.ls" "gsData.ls" "config.ls" "params.ls"
```

Each of the slots can be accessed using an accessor function of the same name (e.g. `cnvData.ls(input)` gets or sets `cnvData.ls`, `gsData.ls(cnvGSA.in)` gets or sets `gsData.ls`, etc.).

The input object is used with `cnvGSA`'s main function, `cnvGSALogRegTest()`:

- `cnvGSALogRegTest(cnvGSA.in)` - Performs a gene-set association test of case vs. control subjects using logistic regression models.

This function produces as its output an object of class "`CnvGSAOutput`" – likewise a simple S4 class that has slots for each of the output elements.

```
> data("cnvGSA_output_example")
> ls()

[1] "cnvGSA.in" "cnvSGA.out"

> class(cnvSGA.out)

[1] "CnvGSAOutput"
attr(,"package")
[1] "cnvGSA"

> slotNames(cnvGSA.out)
[1] "res.ls"      "gsTables.ls" "gsData.ls"   "phData.ls"
```

The output slots contain the following:

- `res.ls` - Original and filtered CNV data
- `gsTables.ls` - Burden analysis results for gene-sets

As with the slots in the input object, each of these can likewise be accessed using an accessor function of the same name (`gsTables.ls()` gets `gsTables.ls`, etc.).

Throughout the following sections, the pre-built `cnvGSA.in` example object from above will be shown to illustrate its typical elements. The function `cnvGSAIn` will take care of most of the processing and formatting. The most important part for the user is to make sure that the input data has all the necessary columns with the correct column names. If this is satisfied the script will take care of the rest.

3 Loading input data

The elements of a `CnvGSAInput` object – i.e. `cnvData.ls`, `phData.ls`, `gsData.ls`, `config.ls`, and `params.ls` – are easily filled when using the function `cnvGSAIn` as described above. It is more important to ensure that the input data is formatted correctly so the script will run properly. Creating the input data should look like the following pseudocode:

```
# Create input object
cnvGSA.in <- cnvGSAIn(configFile)
```

Note that each of these elements can be observed after creating the S4 object using the accessors described below.

3.1 CNV data

The input CNV data for `cnvGSAIn` should follow the following format:

- `$cnv.df` - A data frame containing the CNVs. Each row contains data for one CNV:
 - `SampleID` - ID assigned to the subject's DNA sample in which the CNV was found. The values here should match the corresponding values in the `$s2class` data frame (see below). It is assumed that the correspondence for each is always 1-to-1 with a subject.
 - `Chr` - Chromosome on which the CNV is located.
 - `BP1` - Start position of the CNV on the chromosome.
 - `BP2` - End position of the CNV on the chromosome.
 - `TYPE` - CNV type (typically "LOSS" or "GAIN", but can be any other label indicating deletions and gains).
 - `geneID` - Genes affected by the CNV, stored in a delimited format inside a character string; e.g. "54777;255352;84435" for semicolon-delimited EntrezGene identifiers. (We recommend using this ID system – and in any case the example data in this vignette follows it.) CNVs that are not genic should have an empty string (i.e. "") in this column.

After running `cnvGSAIn` the following will be included in the `$cnvData.ls` slot

```
> str(cnvData.ls(cnvGSA.in),strict.width="cut")
List of 2
 $ cnv.df      :'data.frame': 10666 obs. of  17 variables:
  ..$ SID      : chr [1:10666] "1020_4" "1020_4" "1020_4" "1030_3" ...
  ..$ CHR      : chr [1:10666] "4" "6" "3" "10" ...
  ..$ BP1      : int [1:10666] 34802932 35606076 4110452 56265896 64316996 24115481
  ..$ BP2      : int [1:10666] 35676439 35673400 4145874 56361311 64593616 24202712
  ..$ TYPE     : int [1:10666] 3 3 1 1 1 3 1 3 3 3 ...
  ..$ geneID    : chr [1:10666] NA "2289" NA NA ...
  ..$ exon     : chr [1:10666] "" "FKBP5" "" "" ...
  ..$ SEX      : chr [1:10666] "Male" "Male" "Male" "Male" ...
  ..$ Condition: int [1:10666] 1 1 1 1 1 1 1 1 1 1 ...
  ..$ geneID_DH : chr [1:10666] NA "2289" NA NA ...
  ..$ CnvKey    : chr [1:10666] "4@34802932@35676439@3" "6@35606076@35673400@3" "3@4
  ..$ OlpKL_CNV : num [1:10666] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ OlpKL_SID : num [1:10666] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ SubjCnvKey: chr [1:10666] "1020_4@@4@34802932@35676439@3" "1020_4@@6@35606076@
  ..$ CnvLength_ALL : num [1:10666] 873508 67325 35423 95416 276621 ...
  ..$ CnvLength_TYPE: num [1:10666] 0 0 35423 95416 276621 ...
  ..$ CnvCount_TYPE: num [1:10666] 0 0 1 1 1 0 1 0 0 0 ...
 $ cnv2gene.df:'data.frame': 22322 obs. of  9 variables:
```

```

..$ SubjCnvKey : Factor w/ 10666 levels "1020_4@@3@4110452@4145874@1",...: 1 2 3 4 5 5
..$ geneID      : chr [1:22322] NA NA "2289" NA ...
..$ SEX         : chr [1:22322] "Male" "Male" "Male" "Male" ...
..$ CHR         : chr [1:22322] "3" "4" "6" "10" ...
..$ BP1         : int [1:22322] 4110452 34802932 35606076 56265896 64316996 64316996 39
..$ BP2         : int [1:22322] 4145874 35676439 35673400 56361311 64593616 64593616 40
..$ SID         : chr [1:22322] "1020_4" "1020_4" "1020_4" "1030_3" ...
..$ TYPE        : int [1:22322] 1 3 3 1 1 1 3 3 3 3 ...
..$ geneID_TYPE: chr [1:22322] NA NA NA NA ...

```

Its elements are as follows:

- `$cnv.df` - A data frame containing the CNVs. Each row contains data for one CNV. It has added columns to the previous description and they are:
 - `CnvKey` - Made from combining CHR, BP1, BP2, TYPE with @
 - `OlpKL_CNV` - Chromosome on which the CNV is located.
 - `OlpK_SID` - Start position of the CNV on the chromosome.
 - `SubjCnvKey` - Made from combining SID and Cnvkey using @@
 - `CnvLength_ALL` - CNV type (typically "LOSS" or "GAIN", but can be any other label indicating deletions and gains).
 - `CnvLength_TYPE` - Genes affected by the CNV, stored in a delimited format inside a character string; e.g. "54777;255352;84435" for semicolon-delimited Entrez-Gene identifiers. (We recommend using this ID system – and in any case the example data in this vignette follows it.) CNVs that are not genic should have an empty string (i.e. "") in this column.
 - `CnvCount_TYPE` - Genes affected by the CNV, stored in a delimited format inside a character string; e.g. "54777;255352;84435" for semicolon-delimited Entrez-Gene identifiers. (We recommend using this ID system – and in any case the example data in this vignette follows it.) CNVs that are not genic should have an empty string (i.e. "") in this column.
- `$cnv2gene.df` - A data frame that is used to map the CNVs to the corresponding genes.

3.2 Phenotype/Covariate Data

After the CNV data has been loaded the next data structure that needs to be loaded is `phData`, which contains the phenotype/covariate data. The `phData()` function below is just an accessor for this slot:

Its elements are as follows:

- `$ph.df` - A data frame containing the CNVs. Each row contains data for one CNV:

- **SampleID** - ID assigned to the subject's DNA sample in which the CNV was found.
 - **Covariates** - The covariates that will be included in the analysis. A set of columns, which act as a character, numeric, integer or factor.
 - **Condition** - Identifies the sample as a case(1) or control(0).
 - **Overlap** - Finding those that have overlap in the CNV data frame. Maps it to the corresponding \$SID.
- **\$ph_TYPE.df** - A data frame that is a subset of \$ph.df. This data frame only contains the events of the specified type that the user chooses in the config file (ex. "LOSS"). It will have all the same columns as \$ph.df with the addition of every gene-set as a column. In each of these columns there will be a count of the number of desired events (ex. "LOSS") that happened for each row.

3.3 Gene sets

Before we can properly load the data, we must make sure that the data is formatted correctly. The gene-set data should be one list containing all the gene sets that the user wants to include in the analysis. This list should have names which will act as the \$GsID for the later tables and will be the ID's for a certain gene set. The file containing this gene-sets should be called `gs_all.ls`. There should also be a separate file that will contain the gene set names. This file should be named `gsid2name.chv`.

After the data is put in the correct format, the next data structure that needs to be loaded is `gsData`, which contains the gene-set data. Again, the `gsData()` function below is just an accessor function for this slot:

```
> str(cnvGSA.in@gsData.ls,strict.width="cut")
```

```
List of 7
```

```
$ gs_info.df           :'data.frame': 52 obs. of  4 variables:
```

```
..$ GsKey : chr [1:52] "GS1" "GS2" "GS3" "GS4" ...
```

```
..$ GsID  : chr [1:52] "BspanVHM_PreNat" "BspanVHM_EqlNat" "BspanVHM_PstNat" "BspanVH_
```

```
..$ GsName: chr [1:52] "BspanVHM_PreNat" "BspanVHM_EqlNat" "BspanVHM_PstNat" "BspanVH_
```

```
..$ GsSize: int [1:52] 3037 3037 3131 4600 4605 4596 4601 4131 4185 4190 ...
```

```
$ gs_sel_U.df         :'data.frame': 120486 obs. of  3 variables:
```

```
..$ GsKey : Factor w/ 52 levels "GS1","GS10","GS11",...: 1 1 1 1 1 1 1 1 1 1 ...
```

```
..$ gID   : chr [1:120486] "7105" "4800" "81887" "6405" ...
```

```
..$ GsName: chr [1:120486] "BspanVHM_PreNat" "BspanVHM_PreNat" "BspanVHM_PreNat" "Bspa
```

```
$ gs_colnames_TYPE.chv: chr [1:51] "GS1" "GS10" "GS11" "GS12" ...
```

```
$ gs.ls              :List of 52
```

```
..$ GS1 : int [1:3037] 7105 4800 81887 6405 1595 3927 90293 57679 7035 10181 ...
```

```
..$ GS2 : int [1:3037] 8813 54467 9957 23072 8379 56603 79007 8837 9108 381 ...
```

```
..$ GS3 : int [1:3131] 2519 2729 90529 57185 22875 572 51056 29916 55365 4074 ...
```

```
..$ GS4 : int [1:4600] 7105 4800 57185 1595 572 9957 3927 90293 56603 79007 ...
```

```

..$ GS5 : int [1:4605] 8813 2519 2729 90529 81887 22875 6405 54467 51056 23072 ...
$ geneCount.df          : 'data.frame': 6153 obs. of  4 variables:
..$ geneID   : chr [1:6153] "1" "10" "10003" "100033413" ...
..$ Name     : chr [1:6153] "alpha-1-B glycoprotein" "N-acetyltransferase 2 (arylamine
..$ Controls: int [1:6153] 1 1 0 0 0 0 0 0 0 0 ...
..$ Cases    : int [1:6153] 1 0 1 6 6 6 6 6 6 6 ...
$ geneCount.tab         : 'table' int [1:6203, 1:2] 1 1 0 0 0 0 0 0 0 0 ...
..- attr(*, "dimnames")=List of 2
.. ..$ geneID   : chr [1:6203] "1" "10" "10003" "100033413" ...
.. ..$ Condition: chr [1:2] "0" "1"
$ gs_all.ls            :List of 51
..$ BspanVHM_PreNat    : int [1:3038] 7105 4800 81887 6405 1595 3927 90293 57
..$ BspanVHM_EqlNat    : int [1:3038] 8813 54467 9957 23072 8379 56603 79007
..$ BspanVHM_PstNat    : int [1:3131] 2519 2729 90529 57185 22875 572 51056 2
..$ BspanVH_lg2rpkm4.74 : int [1:4600] 7105 4800 57185 1595 572 9957 3927 9029
..$ BspanHM_lg2rpkm3.21 : int [1:4605] 8813 2519 2729 90529 81887 22875 6405 5

```

The list elements are:

- `$gs_info.df` - A data frame with the columns `$GsKey`, `$GsID`, `$GsName`, `$GsSize` that is used to map the CNVs to the appropriate column
- `$gs_sel.U.df` - A data frame with the columns `$GsKey`, `$geneID`, `$GsName` that is used to map each `$GsKey` with the corresponding `$geneID`. It splits all the genes apart so that there is a one to one relationship.
- `$gs_colnames.TYPE.chv` - A single character vector that contains all the `$GsKey` elements.
- `$gs.ls` - A list containing all the gene sets and may be filtered depending on what the user specifies. The names for this list are the `$GsKey`.
- `$geneCount.tab` - A table that is used to obtain the `geneCount.df` data frame. It uses the `table` function in R to compare the `$geneID` with the `$Condition`.
- `$geneCount.df` - A data frame with the columns `$geneID`, `$Name`, `$Controls`, `$Cases`. This shows how many cases and controls are present for each gene.
- `$gs_all.ls` - A list containing all the gene-sets and their corresponding genes. The names for this list are the `$GsID`.

3.3.1 Marking Known Loci

The option to include known loci into the analysis is available and should resemble the correct format discussed below. These files are defined in the config file. There are two files that are needed to mark the known loci. If the known loci files are properly loaded, the script will mark the known loci in `f.readData()`. The files needed are:

- **klLoci** - Data frame that contains all the known locus for the. It should include the following columns:
 - **CHR** - Chromosome on which the CNV is located.
 - **BP1** - Start position of the CNV on the chromosome.
 - **BP2** - End position of the CNV on the chromosome.
 - **TYPE** - CNV type (typically "LOSS" or "GAIN", but can be any other label indicating deletions and gains).
 - **geneID** - Genes affected by the CNV, stored in a delimited format inside a character string; e.g. "54777;255352;84435" for semicolon-delimited EntrezGene identifiers. (We recommend using this ID system – and in any case the example data in this vignette follows it.) CNVs that are not genic should have an empty string (i.e. "") in this column.
- **klGene** - Data frame that contains the genes of interest for the analysis.
 - **geneID** - Genes affected by the CNV, stored in a delimited format inside a character string; e.g. "54777;255352;84435" for semicolon-delimited EntrezGene identifiers. (We recommend using this ID system – and in any case the example data in this vignette follows it.) CNVs that are not genic should have an empty string (i.e. "") in this column..
 - **Symbol** - Gene Symbol.
 - **TYPE** - CNV type (typically "LOSS" or "GAIN", but can be any other label indicating deletions and gains).

4 Loading test parameters from a file

To make it easier to integrate the association test into a larger bioinformatics pipeline, it is convenient to read in the parameters from an external source such as a text file. One such implementation is to record each parameter on its own line using tsv syntax:

```
param value
#####
# LogReg Config
#####
## CONFIG.LS ##
cnvFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSADData/cnv_AGP_demo.txt"
phFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSADData/ph_AGP_demo.txt"
geneIDFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSADData/geneID_demo.txt"
klGeneFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSADData/kl_gene_AGP_demo.txt"
klLociFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSADData/kl_loci_AGP_demo.txt"
gsFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSADData/gs_data_demo.RData"
outputPath "/Users/josephlugo/Documents/R/PGC2_test/Restructure_Results/8.5/testResults/"
geneListFile ""
```

```

## PARAMS.LS ##
Kl "ALL"
projectName "NS"
gsUSet ""
cnvType 1
covariates "SEX"
kl0lp 0.5
corrections "no_corr,uni_gc,tot_l,cnv_ml"
geneSep ";"
keySep "@"
geneSetSizeMax 1500
geneSetSizeMin 25
filtGs "NO"
covInterest "SEX"
thresholdSzCt 1
fLevels 10
#####
# Viz Config
#####
FDRThreshold 0.1
plotHeight 13
gsList "/Users/josephlugo/Documents/R/PGC2_test/cnvGSADData/gsList.txt"
outputPathViz "/Users/josephlugo/Documents/R/PGC2_test/Restructure_Results/8.5/Viz/"
labelSize 0.7
#####
# ENR Config
#####
pVal "Pvalue_U_dev"
FDR "FDR_BH_U"
coeff "Coeff_U"
keepCoeff "YES"
filtGsEnr "NO"
outputPathEnr "/Users/josephlugo/Documents/R/PGC2_test/Restructure_Results/8.5/enr/"
minCaseCount 0
minControlCount 0
minRatio 0

```

To update the parameters in the S4 object you simply need to change the config file and then run the function below. Once this function is run, all the changed parameters will be updated in the CnvGSAInput S4 object.

```
> cnvGSA.in <- f.readConfig(configFile,cnvGSA.in)
```

4.1 Configuring test parameters

The test requires that the user provides the necessary information to read the correct files needed for the tests to run properly. the `config.ls()` accessor function can be used to interact with this list. The main association test procedure accepts several file names as a simple list (note that the `config.ls()` function below is just an accessor function for this slot):

```
> str( config.ls(cnvGSA.in) )
List of 9
 $ cnvFile      : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSADData/cnv_AGP_demo.txt"
 $ phFile       : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSADData/ph_AGP_demo.txt"
 $ geneIDFile   : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSADData/geneID_demo.txt"
 $ klGeneFile   : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSADData/kl_gene_AGP_demo.txt"
 $ klLociFile   : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSADData/kl_loci_AGP_demo.txt"
 $ gsFile       : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSADData/gs_data_demo.RD"
 $ outputPath   : chr "/Users/josephlugo/Documents/R/PGC2_test/Restructure_Results/8.5/test"
 $ geneListFile : chr ""
 $ config.df     : 'data.frame': 37 obs. of  2 variables:
 ..$ param: chr [1:37] "cnvFile" "phFile" "geneIDFile" "klGeneFile" ...
 ..$ value: chr [1:37] "/Users/josephlugo/Documents/R/PGC2_test/cnvGSADData/cnv_AGP_demo.txt"
```

The parameters for `config.ls` are as follows:

- `cnvFile` - File name for the cnv file user would like to read in.
- `phfile` - File name for the phenotype file user would like to read in.
- `geneIDFile` - File name for the file describing the ID system used for this data.
- `klGeneFile` - File name for the file with the genes of interest.
- `klLocifile` - File name for the file with containing the loci of the genes.
- `gsFile` - File name for the gene set data the user would like to read in.
- `gsNameFile` - File name for the file that contains all the `$GsNames` for the gene sets.
- `outputPath` - Path for the folder where you would like to put your output files.
- `geneListFile` - Path for the folder containing the list of genes that the user would like to include in the regression test. This is optional and will only look at the genes in the list. Use this to look at specific genes of interest.

The main association test procedure accepts several parameters as a simple list (note that the `params.ls()` function below is just an accessor function for this slot):

```

str( params.ls(cnvGSA.in) )
List of 16
 $ K1          : chr "ALL"
 $ projectName : chr "NS"
 $ gsUSet      : chr ""
 $ cnvType     : chr "1"
 $ covariates  : chr "SEX"
 $ kl0lp       : num 0.5
 $ corrections : chr [1:4] "no_corr" "uni_gc" "tot_1" "cnvn_ml"
 $ geneSep     : chr ";"
 $ keySep      : chr "@"
 $ geneSetSizeMin: num 25
 $ geneSetSizeMax: num 1500
 $ filtGs      : chr "NO"
 $ covInterest : chr "SEX"
 $ thresholdSzCt : num 1
 $ fLevels     : num 10
 $ check_type  : chr "1"

```

The parameters for `params.ls` are as follows:

- **K1** - Whether or not you would like to include known loci in the analyses. If you choose "ALL", the tests will be run twice for the known loci and when there are no known loci. Possible values are:
 - "YES" - Include known loci in analysis
 - "NO" - Do not include known loci in analysis
 - "ALL" - Produce results from both "YES" and "NO". Will create two separate data frames.
- **projectName** - This is where the user defines the type of gene-set they are using (ex. for Neurosets the user can put "NS").
- **gsUset** - This is where the user defines the gene set they would like to use as the universal gene set. *This is optional.*
- **cnvType** - The type of CNV that the user would like to use for testing. Possible values are:
 - Any value in the \$TYPE column which may include ("1", "0", "LOSS", "GAIN", etc...). The user may have multiple levels for this column but they should only choose one for the analysis.
 - "ALL" - Include both the LOSSES and the GAINS in the analysis.

- **covariates** - The covariates they would like to include in the regression model. This will be a list of all the desired covariates without any spaces in between (ex. "SEX,CNV_platform").
- **kl0lp** - The threshold of the percentage of overlap having to do with the known loci.
- **corrections** - The correction models that they would like to include in the test. Each correction model will provide different results and comparisons can be made to see the difference between all the different corrections. More than one correction can be included in the analysis. Enter as a list of corrections without any space in between. Possible values are:
 - "no_corr" - Do not include any correction in the analysis.
 - "uni_gc" - Add the universe count correction model.
 - "tot_l" - Add the total count correction model.
 - "cnvn_ml" - Add the mean length and number correction model.
- **geneSep** - The separator that will be used to separate the gene list in the `$geneID` column of the `cnv.df` object. Make sure that you choose the right value here. The results will be wrong is the wrong value is input.
- **keyValue** - The separator that will be used to separate the values in the `$SubjCnvKey` column of the `cnv.df` object.
- **geneSetSizeMax** - The max number of genes that the gene sets are allowed to have.
- **geneSetSizeMin** - The minimum number of genes that the gene sets are allowed to have.
- **filt** - Once the `$geneSetSizeMax` and `$geneSetSizeMax` parameter's have been set you can choose whether or not to include this filter in the analysis. Possible values are:
 - "YES" - Include the filter in the analysis.
 - "NO" - Do not include the filter in the analysis.
- **covInterest** - The covariate of interest that will be included in the output. There can only be one covariate chosen for this parameter.
- **thresholdSzCt** - The threshold for the number of events that you choose to be accepted.
- **flevels** - The max amount of levels the factor `$covInterest` is allowed to have.

This package includes the ability to create visualizations from the output of the main association test. This function is described in the workflow example and it takes in these parameters:

- `FDRThreshold` - Will only include those that are under this number for the plots.
- `plotHeight` - Vertical length of the plots.
- `gsList` - text file containing list of genes that you are interested in looking at plotting. Should be comma separated with no space in between.
- `outputPathViz` - Path for the output of the visualization script.
- `labelSize` - How long you would like the labels to be on the plots.

There is also a function that will create the files necessary to create an enrichment map. This function takes in these parameters:

- `pVal` - the p-value you would like to use for the enrichment map.
- `FDR` - the FDR value you would like to use for the enrichment map.
- `coeff` - the coefficient you would like to use for the enrichment map.
- `keepCoeff` - whether or not you would like to keep the negative coefficients.
- `outputPathEnr` - Path for the output of the enr script.
- `minCaseCount` - the min number of cases each gene is allowed to have.
- `minControlCount` - the min number of controls each gene is allowed to have.
- `minRatio` - the min case/control ratio each gene is allowed to have.

Note: larger gene-sets have more statistical power; this can be taken into account by separately testing gene-sets with different sizes.

5 Full workflow example: case-control analysis of rare CNVs from the Pinto et al. 2014 ASD study

5.1 Loading the data and running the association test

The following code performs an analysis of approx. 5500 CNVs from 2000 subjects against approx. 3700 gene-sets using an `fdr_iter` of 1000. The CNV data-set consists of rare CNVs as described in Pinto et al., Nature 2010 [4], and the gene-sets are a collection imported from the Gene Ontology, KEGG, Biocarta, Reactome, and PFAM (see section 3.3).

```

library( "cnvGSA" )
library( "cnvGSAdata" )

##
## Update the config file
##

## This will give you all the paths needed to update the config file
cnvFile      <- system.file( "extdata", "cnv_AGP_demo.txt", package="cnvGSAdata" )
phFile       <- system.file( "extdata", "ph_AGP_demo.txt", package="cnvGSAdata" )
geneIDFile   <- system.file( "extdata", "geneID_demo.txt", package="cnvGSAdata" )
klGeneFile   <- system.file( "extdata", "kl_gene_AGP_demo.txt", package="cnvGSAdata" )
klLociFile   <- system.file( "extdata", "kl_loci_AGP_demo.txt", package="cnvGSAdata" )
gsFile       <- system.file( "extdata", "gs_data_demo.txt", package="cnvGSAdata" )
## make sure everything is tab separated in the config file
## look at the params above to see how to fill everything in

##
## Create the input object
##

configFile <- (INPUT PATH TO CONFIG FILE)
cnvGSA.in  <- CnvGSAInput() # need S4 object
cnvGSA.in  <- cnvGSAIn(configFile)

##
## Run association test and save the output
##

cnvGSA.out <- CnvGSAOutput() # need S4 object
cnvGSA.out <- cnvGSALogRegTest(cnvGSA.in, cnvGSA.out)

```

5.2 Reviewing the results

As stated in the workflow outline, the output object is a simple S4 class containing a slot for each output data structure:

```

> slotNames(cnvGSA.out)
[1] "res.ls"      "gsTables.ls" "gsData.ls"   "phData.ls"

```

Similar to the input, each of these is a list structure containing further data structures: `res.ls` contains the results of the logistic regression tests with or without the known loci included, `gsTables.ls` contains the gene-set tables for all the gene-sets. `gsData.ls`

contains all the gene-set data, and `phData.ls` contains all the pheontype/covariate data. Just as with the input object, each of these can be accessed using an accessor function of the same name.

5.2.1 Res.ls results

Taking a look first at `res.ls`:

List of 2

```
$ covAll_chipAll_TYPE_KLy.df : 'data.frame': 50 obs. of 39 variables:
$ covAll_chipAll_TYPE_KLn.df : 'data.frame': 50 obs. of 39 variables:
```

The data frames `$covAll_chipAll_TYPE_KLy.df` and `$covAll_chipAll_TYPE_KLn.df` contain the actual logistic regression results. `$covAll_chipAll_TYPE_KLy.df` has the results for the tests where the known loci were included, whereas `$covAll_chipAll_TYPE_KLn.df` has the columns but the known loci are not included in these tests. The TYPE is specified by the user and will vary.

The structure of both data frames is shown in the listing below:

```
> str( cnvGSA.out@res.ls$covAll_chipAll_TYPE_KLy.df )
'data.frame': 50 obs. of 38 variables:
 $ Coeff          : num  1.461 1.03 0.41 0.821 0.392 ...
 $ Pvalue_glm     : num  9.19e-09 2.15e-08 5.92e-07 2.09e-07 8.70e-07 ...
 $ Pvalue_dev     : num  8.67e-13 2.08e-12 4.45e-10 9.15e-09 1.96e-09 ...
 $ Pvalue_dev_s   : num  12.06 11.68 9.35 8.04 8.71 ...
 $ Coeff_U        : num  1.328 0.935 0.366 0.647 0.337 ...
 $ Pvalue_U_glm   : num  9.33e-07 2.81e-06 2.93e-04 2.88e-04 6.15e-04 ...
 $ Pvalue_U_dev   : num  4.19e-08 1.73e-07 7.35e-05 1.71e-04 2.82e-04 ...
 $ Pvalue_U_dev_s : num  7.38 6.76 4.13 3.77 3.55 ...
 $ Coeff_TL       : num  1.349 0.944 0.358 0.71 0.344 ...
 $ Pvalue_TL_glm  : num  2.33e-07 7.15e-07 4.46e-05 2.49e-05 3.67e-05 ...
 $ Pvalue_TL_dev  : num  4.63e-10 2.00e-09 5.31e-07 6.53e-06 1.86e-06 ...
 $ Pvalue_TL_dev_s : num  9.33 8.7 6.27 5.19 5.73 ...
 $ Coeff_CNML     : num  1.375 0.967 0.366 0.753 0.353 ...
 $ Pvalue_CNML_glm : num  9.72e-08 2.55e-07 1.66e-05 5.06e-06 1.20e-05 ...
 $ Pvalue_CNML_dev : num  8.52e-11 2.51e-10 1.02e-07 8.20e-07 3.03e-07 ...
 $ Pvalue_CNML_dev_s : num  10.07 9.6 6.99 6.09 6.52 ...
 $ SZ_g1n         : num  3.86 5.34 9.25 4.92 7.24 ...
 $ CT_g1n         : num  0.899 1.627 4.88 2.269 4.409 ...
 $ SZ_g2n         : num  0.74 1.16 2.7 1.22 2.7 ...
 $ CT_g2n         : num  0.0428 0.1284 0.8134 0.0856 0.8134 ...
 $ SZ_g3n         : num  0.529 0.687 1.639 0.687 1.745 ...
 $ CT_g3n         : num  0 0 0.342 0 0.171 ...
```



```

$ SZ_g4n      : num  0.37 0.582 1.163 0.106 1.322 ...
$ CT_g4n      : num  0 0 0.1712 0 0.0856 ...
$ SZ_g5n      : num  0.106 0.423 0.793 0.106 1.005 ...
$ CT_g5n      : num  0 0 0.1284 0 0.0856 ...
$ SZ_gTT      : num  1891 1891 1891 1891 1891 ...
$ CT_gTT      : num  2336 2336 2336 2336 2336 ...
$ SZ_Female   : num  6.3 6.3 10.37 5.19 7.04 ...
$ SZ_Male     : num  3.45 5.18 9.07 4.87 7.28 ...
$ CT_Female   : num  0.962 1.603 4.567 2.484 5.048 ...
$ CT_Male     : num  0.827 1.654 5.239 2.022 3.676 ...
$ FDR_BH_U    : num  2.09e-06 4.33e-06 9.19e-04 1.71e-03 1.91e-03 ...
$ FDR_BH_TL   : num  2.31e-08 4.99e-08 8.85e-06 5.44e-05 2.32e-05 ...
$ FDR_BH_CNML : num  4.26e-09 6.27e-09 1.70e-06 6.83e-06 3.79e-06 ...
$ GsID        : chr   "FMR1_Targets_Darnell" "PSD_BayesGrant_fullset" ...
$ GsName      : chr   "FMR1_Targets_Darnell" "PSD_BayesGrant_fullset" ...
$ GsSize      : int   840 1407 4600 1230 3131 4605 927 1424 3037 116 ...

```

Each row contains results for a single gene-set. The columns are as follows:

- **GsID**, **GsName**, and **GsSize** show the gene-set's key made by the script, identifier, name, and number of member genes respectively.
- **Coeff**, **Coeff_U**, **Coeff_TL**, and **Coeff_CNML** show the gene-set's coefficients for each of the different corection models that were used. This represents the log odds of being a case in the exposed vs. unexposed, adjusted for the covariates and whatever correction was being used.
- **Pvalue_glm**, **Pvalue_U_glm**, **Pvalue_TL_glm**, and **Pvalue_CNML_glm** show the gene-set's p-values for each of the different correction models that were used. These p-values were obtained using the z-test.
- **Pvalue_dev**, **Pvalue_U_dev**, **Pvalue_TL_dev**, and **Pvalue_CNML_dev** show the gene-set's p-values for each of the different corection models that were used. These p-values were obtained using the chi-square test.
- **Pvalue_dev_s**, **Pvalue_U_dev_s**, **Pvalue_TL_dev_s**, and **Pvalue_CNML_dev_s** show the gene-set's p-values for each of the different corection models that were used. These p-values were obtained by using the formula $-\log(10)(\text{p-value from chi-square test})(\text{sign of coeff})$
- **SZ_g1n**, **SZ_g2n**, **SZ_g3n**, **SZ_g4n**, and **SZ_g5n** show the percentage of samples from the cases that each gene-set has at least 1,2,3,4 and 5 events in.
- **CT_g1n**, **CT_g2n**, **CT_g3n**, **CT_g4n**, and **CT_g5n** show the percentage of samples from the controls that each gene-set has at least 1,2,3,4 and 5 events in.
- **SZ_gTT** and **CT_gTT** show the total number of cases and controls there are.
- **SZ_Female** and **SZ_male** show the percentage of samples from the cases that must have at least a user specified number of events and also is part of the SEX covariate. This will change depending on which covariate the user would like to see in the output.

- `CT_Female` and `CT_male` show the percentage of samples from the controls that must have at least a user specified number of events and also is part of the `SEX` covariate. This will change depending on which covariate the user would like to see in the output.
- `FDR_BH_U`, `FDR_BH_TL`, and `FDR_BH_CNML` show FDR values using three methods. `FDR_BH_U` is the Benjamini-Hochberg FDR for the p-values calculated using the `uni_gc` correction for each gene-set calculated in relation to all the gene-sets in the input. `FDR_BH_TL` is the same as the previous, except with the `tot_l` correction. `FDR_BH_CNML` is the same as the previous, except with the `cnvn_ml` correction.

5.2.2 Detailed analysis of gene-set associations

As a further aid in understanding the CNVs and genes contributing to the association results, there is `gsTables.ls` – a list of data frames, one for each of the gene-sets, containing information about the CNVs and corresponding genes affecting the gene-set:

```
> str( gsTables.ls, max.level=1, list.len=5 )
List of 52
 $ BspanHM_lg2rpkm3.21      : 'data.frame': 707 obs. of  10 variables:
 .. [list output truncated]
 $ BspanLA_lg2rpkm.MIN      : 'data.frame': 1883 obs. of  10 variables:
 .. [list output truncated]
 $ BspanML_lg2rpkm0.93      : 'data.frame': 1026 obs. of  10 variables:
 .. [list output truncated]
 $ BspanVH_lg2rpkm4.74      : 'data.frame': 546 obs. of  10 variables:
 .. [list output truncated]
 $ BspanVHM_EqlNat          : 'data.frame': 418 obs. of  10 variables:
 .. [list output truncated]
 [list output truncated]
```

To get `gsTables.ls` you need to run the following:

```
> cnvGSAGsTables(cnvGSA.in,cnvGSA.out)
```

The structure of the data frame for a particular gene-set is similar to that of `cnvData$cnv` in the input:

```
> gsTables.ls[[2]][10:19,]
      geneID_TYPE      SubjCnvKey      SEX CHR      BP1
227  100310812 HABC_900437_900437@@7@101900904@102126404@1 Female  7 101900904
229  100310812 HABC_900216_900216@@7@101900914@102126404@1 Female  7 101900914
231  100310812 HABC_900320_900320@@7@101901092@102121342@1 Female  7 101901092
233  100310812 HABC_900736_900736@@7@101900904@102126404@1 Female  7 101900904
235  100310812 HABC_902946_902946@@7@101900904@102139865@1  Male  7 101900904
237  100310812 HABC_901157_901157@@7@101900846@102146951@1 Female  7 101900846
```

| | | | | | | |
|-----|-----------|--------------------|--------------------------|---------------------|---------------------|-----------|
| 239 | 100310812 | HABC_900766_900766 | @7@101900904@102126404@1 | Female | 7 | 101900904 |
| 241 | 100310812 | HABC_902681_902681 | @7@101900846@102126404@1 | Female | 7 | 101900846 |
| 243 | 100310812 | 20011_1048001 | @7@101900846@102139865@1 | Male | 7 | 101900846 |
| 245 | 100310812 | 16086_1571084001 | @7@101900914@102126404@1 | Male | 7 | 101900914 |
| | BP2 | SID | TYPE | GsID | | GsName |
| 227 | 102126404 | HABC_900437_900437 | 1 | BspanLA_lg2rpkm.MIN | BspanLA_lg2rpkm.MIN | |
| 229 | 102126404 | HABC_900216_900216 | 1 | BspanLA_lg2rpkm.MIN | BspanLA_lg2rpkm.MIN | |
| 231 | 102121342 | HABC_900320_900320 | 1 | BspanLA_lg2rpkm.MIN | BspanLA_lg2rpkm.MIN | |
| 233 | 102126404 | HABC_900736_900736 | 1 | BspanLA_lg2rpkm.MIN | BspanLA_lg2rpkm.MIN | |
| 235 | 102139865 | HABC_902946_902946 | 1 | BspanLA_lg2rpkm.MIN | BspanLA_lg2rpkm.MIN | |
| 237 | 102146951 | HABC_901157_901157 | 1 | BspanLA_lg2rpkm.MIN | BspanLA_lg2rpkm.MIN | |
| 239 | 102126404 | HABC_900766_900766 | 1 | BspanLA_lg2rpkm.MIN | BspanLA_lg2rpkm.MIN | |
| 241 | 102126404 | HABC_902681_902681 | 1 | BspanLA_lg2rpkm.MIN | BspanLA_lg2rpkm.MIN | |
| 243 | 102139865 | 20011_1048001 | 1 | BspanLA_lg2rpkm.MIN | BspanLA_lg2rpkm.MIN | |
| 245 | 102126404 | 16086_1571084001 | 1 | BspanLA_lg2rpkm.MIN | BspanLA_lg2rpkm.MIN | |

(Note that the selection above shows only 10 rows from a single data frame in the list.).

Examining the data frame for a particular gene-set may reveal that its association due to certain genes may actually be better explained by *other* genes (those that have a clearer functional impact or that have previously been associated with the cases under consideration).

5.2.3 Visualizations

Once the `CnvGSAOutput` and `CnvGSAInput` S4 objects are made you can run the `f.makeViz` to create a set of different visualizations. An example of how to run it is shown below:

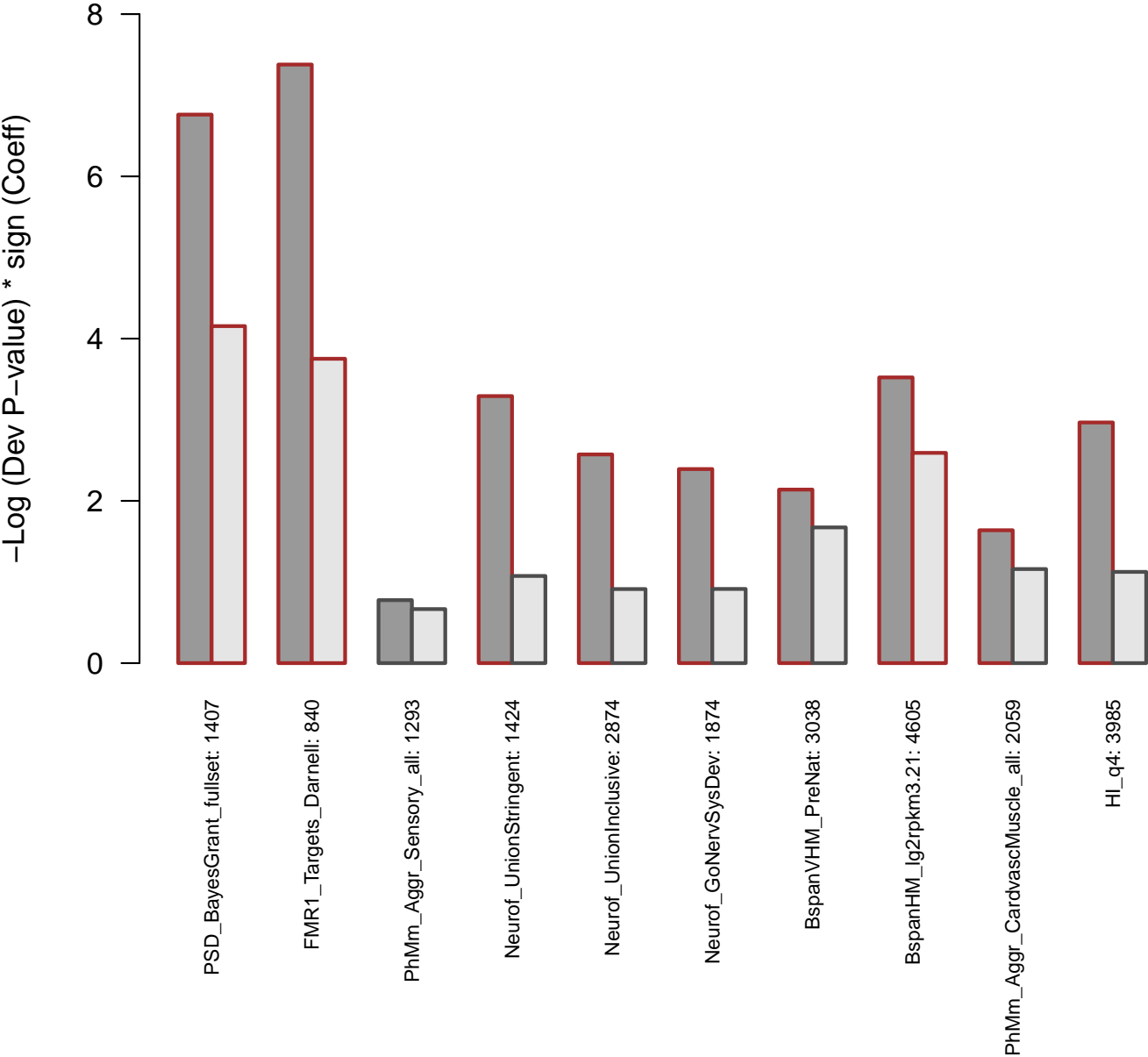
```
> library(cnvGSAdata)
> data("cnvGSA_input_example")
> data("cnvGSA_output_example")
> f.makeViz(cnvGSA.in,cnvGSA.out)
```

There are three different plots that will be made by this function. They show the significance, effect and support. The significance plots will be broken down by the different corrections, so there will be a total of 5 plots made when the script is run. The gene-sets that will be displayed must be input by the user and should only include ones that are particularly interesting. The user must input a comma separated file with no spaces containing all the gene-sets that they would like to look at. The file should resemble this:

```
PSD_BayesGrant_fullset,FMR1_Targets_Darnell,PhMm_Aggr_Sensory_all,
Neurof_UnionStringent,Neurof_UnionInclusive,Neurof_GoNervSysDev,BspanVHM_PreNat,
BspanHM_lg2rpkm3.21,PhMm_Aggr_CardvascMuscle_all,HI_q4
```

For the plots, anything with an FDR below the user specified cutoff, will have a brown border. This way the user can easily see how reliable the results are. An example of one of the plots that will be made is included below:

1: Significance: U



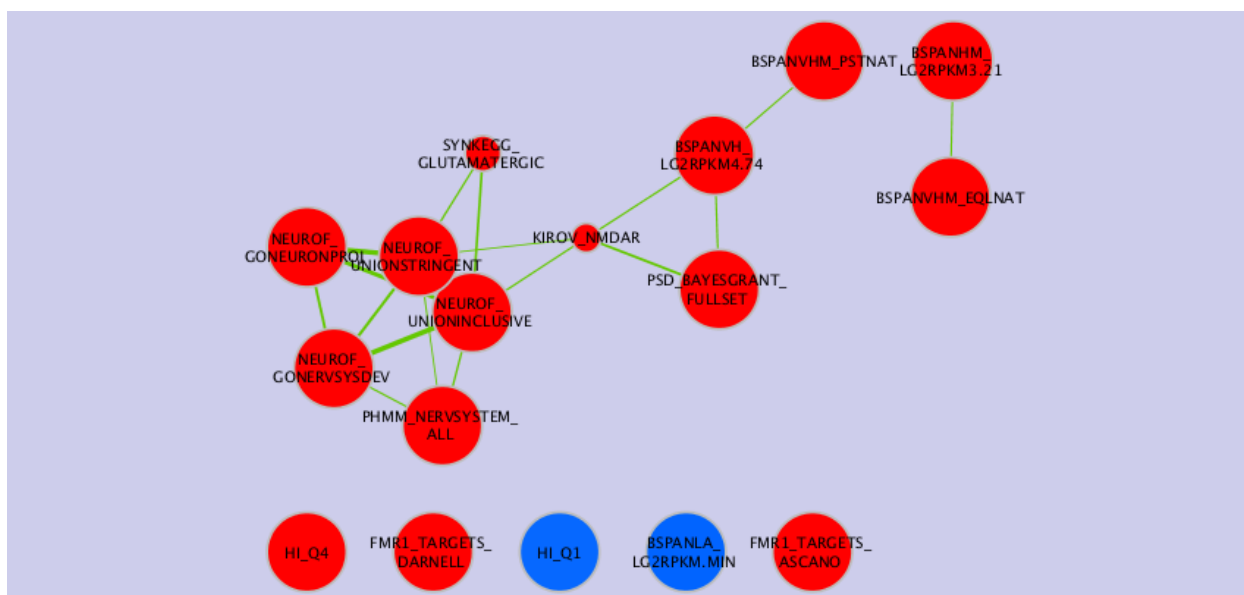
5.2.4 Enrichment Map Pre-Processing

To conduct further analysis of the results, this package has the function `f.enrFiles` that will do all the pre-processing for the files necessary to make an enrichment map. The enrichment map is created using Cytoscape and displays a network of gene-sets and how they interact with one another. It will produce clusters that will convey the data in a meaningful way.

The `f.enrFiles` function will create two files (GMT and generic) needed to make the enrichment map. For more information on these types of files, see the **Enrichment Map User Guide**. Once the function is run, these two files will be output and will be available to use with the **Cytoscape** software. It is recommended to use the Jaccard+Overlap combined option when creating the map. You can see a basic example below.

```
## To create the two files, simply run this function
> f.enrFiles(cnvGSA.in,cnvGSA.out)
```

The output will allow the user to use Cytoscape to create a network map that will look similar to this:



If you would like to use your own files to make the enrichment map they should follow the examples below. This can be found on the **Enrichment Map User Guide** as described above.

The GMT file should follow something like this:

- Each row of the geneset file represents one geneset and consists of:
 - geneset name (-tab-) description (-tab-) a list of tab-delimited genes that are part of that geneset.

The generic file should follow this format:

- The generic results file is a tab delimited file with enriched gene-sets and their corresponding p-values (and optionally, FDR corrections)
- The Generic Enrichment Results file needs:
 - gene-set ID (must match the gene-set ID in the GMT file),
 - gene-set name or description,
 - p-value,
 - FDR correction value
 - Phenotype: +1 or -1, to identify enrichment in up- and down-regulation, or, more in general, in either of the two phenotypes being compared in the two-class analysis (+1 maps to red and -1 maps to blue).

6 References

- [1] Goeman JJ, Bhlmann P. Analyzing gene expression data in terms of gene sets: methodological issues. *Bioinformatics*. 2007 Apr 15; **23**(8): 980–7.
- [2] Raychaudhuri S et al. Accurately assessing the risk of schizophrenia conferred by rare copy-number variation affecting genes with brain function. *PLoS Genet*. 2010 Sep 9; **6**(9): e1001097.
- [3] C. R. Marshall et al. Structural Variation of Chromosomes in Autism Spectrum Disorder. *Am J Hum Genet*. 2008 Feb 8; **82**(2): 477–488.
- [4] Pinto, D et al. Functional impact of global rare copy number variation in autism spectrum disorders. *Nature*. 2010 Jul 15; **466**(7304): 368–72.
- [5] Kirov G et al. De novo CNV analysis implicates specific abnormalities of postsynaptic signalling complexes in the pathogenesis of schizophrenia. *Mol Psychiatry*. 2012 Feb; **17**(2): 142–53.
- [6] Pinto, D et al. Convergence of Genes and Cellular Pathways Dysregulated in Autism Spectrum Disorders. *Am J Hum Genet*. 2014 May 1; **94**(5): 677–694.
- [7] Merico D et al. Enrichment map: a network-based method for gene-set enrichment visualization and interpretation. *PLoS One*. 2010 Nov 15; **5**(11): e13984.