MIGSA: Massive and Integrative Gene Set Analysis

Juan C Rodriguez
CONICET
Universidad Católica de Córdoba
Universidad Nacional de Córdoba

Cristóbal Fresno
Instituto Nacional de Medicina Genómica

Andrea S Llera
CONICET
Fundación Instituto Leloir

Elmer A Fernández
CONICET
Universidad Católica de Córdoba
Universidad Nacional de Córdoba

Abstract

The **MIGSA** package allows to perform a massive and integrative gene set analysis over several experiments and gene sets simultaneously. It provides a common gene expression analytic framework that grants a comprehensive and coherent analysis. Only a minimal user parameter setting is required to perform both singular and gene set enrichment analyses in an integrative manner by means of enhanced versions of the best available methods, i.e. **dEnricher** and **mGSZ** respectively.

One of the greatest strengths of this big omics data tool is the availability of several functions to explore, analyze and visualize its results in order to facilitate the data mining task over huge information sources.

The MIGSA package also allows to easily load the most updated gene sets collections from several repositories.

Keywords: singular enrichment analysis, over representation analysis, gene set enrichment analysis, functional class scoring, big omics data, r package, bioconductor.

1. Introduction

The functional analysis methodology allows researchers to gain biological insight from a list of deregulated gene sets between experimental conditions of interest. As suggested by (Rodriguez *et al.* 2016) both singular enrichment analysis (SEA) and gene set enrichment analysis (GSEA) must be performed over the same dataset in order to gain as much biological insight as possible. This strategy is known as Integrative Functional Analysis (IFA) and integrates into the same analysis with enhanced versions of the dEnricher (Fang and Gough 2014) and mGSZ (Mishra *et al.* 2014) methods.

At present, there are several freely available datasets which provide data over the same disease, characteristic of interest (e.g. survival), or subjects studied over several different platforms. The Cancer Genome Atlas (TCGA) among other projects makes possible the study and comparison in a massive way of these datasets, not only among them but, also against our

own population of interest. This unprecedented opportunity allows researchers to search for common functional patterns between these studies, or, more interestingly, particular patterns of our experiment in question. However, this type of approach has not been implemented in any existing tool yet, leaving aside valuable biological information that might assist research hypotheses.

Here, we present a Massive and Integrative Gene Set Analysis tool called **MIGSA**. It allows to evaluate and compare, massively and transparently, a large collection of datasets coming from diverse sources, maintaining the gene set enrichment ideas of IFA and minimizing parameter settings. In addition, it includes a gene ranking score alternative for RNAseq data by integrating the *Voom+Limma* methodological approach. It provides an enhanced version of mGSZ (`MIGSAmGSZ`) faster than the default implementation, in order to speed up even more its execution, **MIGSA** can be run using multicore architectures. In this sense it can be applied over a large collection of datasets on many gene sets in a fast way. Finally, **MIGSA** provides several user-friendly methods to easily explore and visualize results at gene set, dataset and individual gene level to aid researchers in their biological hypothesis understanding.

2. Preliminaries

2.1. Citing MIGSA

MIGSA implements a body of methodological research by the authors and co-workers. Citations are the main means by which the authors receive professional credit for their work. The **MIGSA** package can be cited as:

Rodriguez JC, Merino GA, Llera AS, Fernández EA (2019).
"Massive integrative gene set analysis enables functional characterization of breast cancer subtypes." *Journal of Biomedical Informatics*, **93**, 103157.

Rodriguez JC, González GA, Fresno C, Llera AS, Fernández EA (2016).
"Improving information retrieval in functional analysis." *Computers in Biology and Medicine*, **79**, 10–20.

2.2. Installation

MIGSA is a package for the R computing environment and it is assumed that you have already installed R. See the R project at http://www.r-project.org. To install the latest version of **MIGSA**, you will need to be using the latest version of R.
MIGSA is part of the Bioconductor project at http://www.bioconductor.org. (Prior to R 3.4).
To get **MIGSA** package you can type in an R session:

```
> ## try http:// if https:// URLs are not supported
> if (!requireNamespace("BiocManager", quietly = TRUE)) {
+   install.packages("BiocManager")
+ }
> BiocManager::install("MIGSA")
```

3. Gene sets

MIGSA allows to perform the functional analysis of any type of gene sets provided by the user. Such gene sets should be present as GeneSetCollection objects from the **GSEABase** R library, in this section we will give a brief introduction on how to construct such an object from our own gene sets. In addition, the tools provided by **MIGSA** to automatically load various collections of known gene sets will be presented.

3.1. Sample GeneSetCollection creation

Here we present a simple way to create a GeneSetCollection object from own gene sets, for more detailed information please refer to the **GSEABase** documentation.
For this example we are going to manually create the GeneSetCollection object for the gene sets hsa00232, hsa00130 and hsa00785 from KEGG.
First, we will have to create each gene set separately, and then the GeneSetCollection object.

```
> library(GSEABase)
> gs1 <- GeneSet(c("10", "1544", "1548", "1549", "1553", "7498", "9"),
+   setName = "hsa00232",
+   setIdentifier = "Caffeine metabolism"
+ )
> gs1

setName: hsa00232
geneIds: 10, 1544, ..., 9 (total: 7)
geneIdType: Null
collectionType: Null
details: use 'details(object)'

> gs2 <- GeneSet(c("10229", "27235", "3242", "51004", "51805", "6898", "84274"),
+   setName = "hsa00130",
+   setIdentifier = "Ubiquinone and other terpenoid-quinone biosynthesis"
+ )
> gs3 <- GeneSet(c("11019", "387787", "51601"),
+   setName = "hsa00785",
+   setIdentifier = "Lipoic acid metabolism"
+ )
> ## And now construct the GeneSetCollection object.
> gsetsColl <- GeneSetCollection(list(gs1, gs2, gs3))
> gsetsColl

GeneSetCollection
  names: hsa00232, hsa00130, hsa00785 (3 total)
  unique identifiers: 10, 1544, ..., 51601 (17 total)
  types in collection:
    geneIdType: NullIdentifier (1 total)
    collectionType: NullCollection (1 total)
```

3.2. MIGSA gene sets loading

As mentioned above, **MIGSA** provides functions for automatically loading known collections of gene sets. These functions are `loadGo` and `downloadEnrichrGeneSets`, the first constructs the GeneSetCollection object using the **org.Hs.eg.db** R package. Meanwhile, `downloadEnrichrGeneSets` constructs the object by downloading the gene sets from the Enrichr database (http://amp.pharm.mssm.edu/Enrichr/#stats). Enrichr gene set names can be listed with the `enrichrGeneSets` function.

```
> ## Not run:
>
> ## Load cellular component gene sets (another possibility would be "MF" or "BP")
> ccGsets <- loadGo("CC")
> # It is a GeneSetCollection object
>
> ## Load KEGG and Reactome gene sets
> keggReact <- downloadEnrichrGeneSets(c("KEGG_2015", "Reactome_2015"))
> ## It is a list object containing two GeneSetCollection objects
>
> ## End(Not run)
```

4. MIGSAmGSZ

4.1. mGSZ speedup

As stated below, **MIGSA** provides the `MIGSAmGSZ` function, which implements *mGSZ* but running much faster. In order to test `MIGSAmGSZ`'s correctness and speed up over *mGSZ*, it was evaluated using the TCGA's microarray breast cancer dataset. Basal vs. Luminal A contrast was tested (16,207 genes x 237 subjects) over the Gene Ontology and KEGG gene sets (20,425 gene sets).
This analysis was carried out using an Intel(R) Xeon(R) E5-2620 v3 @ 2.40GHz (24 cores), 128 GB RAM. Different number of cores were used to analyze the speed up.
Let's test it!
Note that we are using MulticoreParam as I am testing under Linux.

```
> library(BiocParallel)
> library(mGSZ)
> library(MIGSA)
> library(MIGSAdata)
> data(tcgaMAdata)
> subtypes <- tcgaMAdata$subtypes
> geneExpr <- tcgaMAdata$geneExpr
> ## MA data: filter genes with less than 30% of genes read per condition
> dim(geneExpr)

[1] 16207   237
```

```
> geneExpr <- geneExpr[
+   rowSums(is.na(geneExpr[, subtypes == "Basal" ])) <
+     .3 * sum(subtypes == "Basal") &
+     rowSums(is.na(geneExpr[, subtypes == "LumA" ])) <
+       .3 * sum(subtypes == "LumA"),
+ ]
> dim(geneExpr)

[1] 16207   237

> ## Not run:
>
> ## Download GO and KEGG gene sets using MIGSA
> gSets <- list(
+   KEGG = downloadEnrichrGeneSets("KEGG_2015")[[1]],
+   BP = loadGo("BP"),
+   CC = loadGo("CC"),
+   MF = loadGo("MF")
+ )
> gSetsList <- do.call(c, lapply(gSets, MIGSA:::asList))
> rm(gSets)
> nCores <- c(1, 2, 4, 8, 10, 12, 14)
> allRes <- lapply(nCores, function(actCores) {
+   # setting in how many cores to run
+   register(MulticoreParam(
+     workers = actCores, threshold = "DEBUG",
+     progressbar = TRUE
+   ))
+
+   set.seed(8818)
+   newtimeSpent <- Sys.time()
+   MIGSAmGSZres <- MIGSAmGSZ(geneExpr, gSetsList, subtypes)
+   newtimeSpent <- Sys.time() - newtimeSpent
+
+   res <- list(timeSpent = newtimeSpent, res = MIGSAmGSZres)
+
+   return(res)
+ })
> set.seed(8818)
> timeSpent <- Sys.time()
> mGSZres <- mGSZ(geneExpr, gSetsList, subtypes)
> timeSpent <- Sys.time() - timeSpent
> mGSZres <- mGSZres$mGSZ
> ## this tests that the returned values are equal, must give all TRUE
> lapply(allRes, function(actRes) {
+   actRes <- actRes$res
+   actRes <- actRes[, 1:4]
```

```
+   mergedRes <- merge(mGSZres, actRes,
+     by = "gene.sets",
+     suffixes = c("mGSZ", "MIGSAmGSZ")
+   )
+
+   all(unlist(lapply(2:4, function(x) {
+     all.equal(mergedRes[, x], mergedRes[, x + 3])
+   })))
+ })
> ## End(Not run)

> ## As last chunk of code was not executed, we load that data:
> library(MIGSAdata)
> data(mGSZspeedup)
> nCores <- mGSZspeedup$nCores
> allRes <- mGSZspeedup$allRes
> timeSpent <- mGSZspeedup$timeSpent
> ## End(Loading data)
>
> newtimeSpent <- lapply(allRes, function(actRes) {
+   actRes$timeSpent
+ })
> names(newtimeSpent) <- nCores
> speeduptable <- c(timeSpent, unlist(newtimeSpent))
> names(speeduptable) <- c(1, nCores)
> ## Let's put all times in the same unit in order to measure speedup
> newtimeSpent <- lapply(newtimeSpent, function(acttime) {
+   units(acttime) <- "secs"
+   return(acttime)
+ })
> units(timeSpent) <- "secs"
> speedup <- do.call(c, lapply(newtimeSpent, function(acttime) {
+   as.numeric(timeSpent) / as.numeric(acttime)
+ }))
> speeduptable <- rbind(speeduptable, c(1, speedup))
> ## calculate efficiency
> speeduptable <- rbind(
+   speeduptable,
+   speeduptable[2, ] / as.numeric(colnames(speeduptable))
+ )
> rownames(speeduptable) <- c("Runtime", "Speedup", "Efficiency")
> round(speeduptable, 2)

              1    1     2     4     8    10    12    14
Runtime    2.46 1.55 46.50 24.98 15.63 13.67 14.79 28.43
Speedup    1.00 1.58  3.18  5.91  9.45 10.81  9.98  5.19
Efficiency 1.00 1.58  1.59  1.48  1.18  1.08  0.83  0.37
```

As it can be seen in Table 1, no matter the number of cores in which `MIGSAmGSZ` was tested, it outperformed *mGSZ*. Running in one core, it has shown a speedup of 1.6X, reaching for a top of 10.8X speedup with ten cores, giving the same results in 14 minutes in contrast to *mGSZ*'s 2.46 hours execution.

Table 1: MIGSAmGSZ speedup

	mGSZ	MIGSAmGSZ						
#cores	1	1	2	4	8	10	12	14
Runtime	2.46h	1.55h	46.5m	24.98m	15.63m	13.67m	14.79m	28.43m
Speedup	1	1.58	3.18	5.91	9.45	10.81	9.98	5.19
Efficiency	1	1.58	1.59	1.48	1.18	1.08	0.83	0.37

4.2. MIGSAmGSZ simple example

Following, we show how to simply execute one `MIGSAmGSZ` analysis.
In this example we will generate an expression matrix with 200 genes (ten differentially expressed) and eight subjects (four of condition "C1" and four of "C2"), and 50 gene sets of ten genes each one.

```
> library(MIGSA)
> ## Let's create our gene expression matrix with 200 genes and 8 subjects
> nSamples <- 8
> # 8 subjects
> nGenes <- 200
> # 200 genes
> geneNames <- paste("g", 1:nGenes, sep = "")
> # with names g1 ... g200
>
> ## Create random gene expression data matrix.
> set.seed(8818)
> exprMatrix <- matrix(rnorm(nGenes * nSamples), ncol = nSamples)
> ## It must have rownames, as they will be treated as the gene names!
> rownames(exprMatrix) <- geneNames
> ## There will be 10 differentially expressed genes.
> nDeGenes <- 10
> ## Let's generate the offsets to sum to the differentially expressed genes.
> deOffsets <- matrix(2 * abs(rnorm(nDeGenes * nSamples / 2)), ncol = nSamples / 2)
> ## Randomly select which are the DE genes.
> deIndexes <- sample(1:nGenes, nDeGenes, replace = FALSE)
> exprMatrix[deIndexes, 1:(nSamples / 2)] <-
+   exprMatrix[deIndexes, 1:(nSamples / 2)] + deOffsets
> ## 4 subjects with condition C1 and 4 with C2.
> conditions <- rep(c("C1", "C2"), c(nSamples / 2, nSamples / 2))
> nGSets <- 50
> # 50 gene sets
> ## Let's create randomly 50 gene sets, of 10 genes each
```

```
> gSets <- lapply(1:nGSets, function(i) sample(geneNames, size = 10))
> names(gSets) <- paste("set", as.character(1:nGSets), sep = "")
> ## with names set1 ... set50
>
> ## And simply execute MIGSAmGSZ
> MIGSAmGSZres <- MIGSAmGSZ(exprMatrix, gSets, conditions)

INFO [2021-05-19 20:45:14] Number of unique permutations: 66
INFO [2021-05-19 20:45:14] Getting ranking at cores: 4

> ## It is just a simple data.frame
> head(MIGSAmGSZres)

      gene.sets     pvalue mGszScore
set3       set3 0.03130511 -1.815278
set40     set40 0.05469617  2.748644
set46     set46 0.05793451 -2.848961
set20     set20 0.07557986  1.632850
set38     set38 0.07695490  1.533487
set7       set7 0.07785534  1.591581
                                                      impGenes
set3    g10, g197, g57, g144, g19, g8, g153, g81, g63, g138
set40  g147, g121, g141, g66, g166, g45, g94, g34, g37, g82
set46                                        g162, g196, g136
set20       g20, g55, g141, g26, g172, g21, g60, g158, g186
set38 g159, g117, g86, g77, g93, g181, g146, g26, g149, g45
set7    g111, g147, g129, g118, g34, g69, g87, g88, g91, g8
```

5. MIGSA simple example

Following, we show how to simply execute one *MIGSA* analysis.
In this example we will generate two expression matrices with 300 genes (30 differentially expressed) and 16 subjects (8 of condition "C1" and 8 of "C2"), and two sets of 30 gene sets of ten genes each one.

```
> library(MIGSA)
> ## Let's simulate two expression matrices of 300 genes and 16 subjects.
> nGenes <- 300
> # 300 genes
> nSamples <- 16
> # 16 subjects
> geneNames <- paste("g", 1:nGenes, sep = "")
> # with names g1 ... g300
>
> ## Create the random gene expression data matrices.
```

```
> set.seed(8818)
> exprData1 <- matrix(rnorm(nGenes * nSamples), ncol = nSamples)
> rownames(exprData1) <- geneNames
> exprData2 <- matrix(rnorm(nGenes * nSamples), ncol = nSamples)
> rownames(exprData2) <- geneNames
> ## There will be 30 differentially expressed genes.
> nDeGenes <- nGenes / 10
> ## Let's generate the offsets to sum to the differentially expressed genes.
> deOffsets <- matrix(2 * abs(rnorm(nDeGenes * nSamples / 2)), ncol = nSamples / 2)
> ## Randomly select which are the DE genes.
> deIndexes1 <- sample(1:nGenes, nDeGenes, replace = FALSE)
> exprData1[deIndexes1, 1:(nSamples / 2)] <-
+   exprData1[deIndexes1, 1:(nSamples / 2)] + deOffsets
> deIndexes2 <- sample(1:nGenes, nDeGenes, replace = FALSE)
> exprData2[deIndexes2, 1:(nSamples / 2)] <-
+   exprData2[deIndexes2, 1:(nSamples / 2)] + deOffsets
> exprData1 <- new("MAList", list(M = exprData1))
> exprData2 <- new("MAList", list(M = exprData2))
> ## 8 subjects with condition C1 and 8 with C2.
> conditions <- rep(c("C1", "C2"), c(nSamples / 2, nSamples / 2))
> fitOpts <- FitOptions(conditions)
> nGSets <- 30
> # 30 gene sets
> ## Let's create randomly 30 gene sets, of 10 genes each
>
> gSets1 <- lapply(1:nGSets, function(i) sample(geneNames, size = 10))
> names(gSets1) <- paste("set", as.character(1:nGSets), sep = "")
> myGSs1 <- as.Genesets(gSets1)
> gSets2 <- lapply(1:nGSets, function(i) sample(geneNames, size = 10))
> names(gSets2) <- paste("set", as.character((nGSets + 1):(2 * nGSets)), sep = "")
> myGSs2 <- as.Genesets(gSets2)
> igsaInput1 <- IGSAinput(
+   name = "igsaInput1", expr_data = exprData1,
+   fit_options = fitOpts
+ )
> igsaInput2 <- IGSAinput(
+   name = "igsaInput2", expr_data = exprData2,
+   fit_options = fitOpts
+ )
> experiments <- list(igsaInput1, igsaInput2)
> ## As we did not set gene sets for each IGSAinput, then we will have to
> ## provide them in MIGSA function
>
> ## another way of generating the same MIGSA input would be setting the
> ## gene sets individually to each IGSAinput:
> igsaInput1 <- IGSAinput(
+   name = "igsaInput1", expr_data = exprData1,
```

```
+   fit_options = fitOpts,
+   gene_sets_list = list(myGeneSets1 = myGSs1, myGeneSets2 = myGSs2)
+ )
> igsaInput2 <- IGSAinput(
+   name = "igsaInput2", expr_data = exprData2,
+   fit_options = fitOpts,
+   gene_sets_list = list(myGeneSets1 = myGSs1, myGeneSets2 = myGSs2)
+ )
> experiments <- list(igsaInput1, igsaInput2)
> ## And then simply run MIGSA
> migsaRes <- MIGSA(experiments)

INFO [2021-05-19 20:45:16] ************************************
INFO [2021-05-19 20:45:16] Starting MIGSA analysis.
INFO [2021-05-19 20:45:16] ************************************
INFO [2021-05-19 20:45:16] igsaInput1 : Starting IGSA analysis.
INFO [2021-05-19 20:45:16] 60 Gene Sets.
INFO [2021-05-19 20:45:16] igsaInput1 : dEnricher starting.
INFO [2021-05-19 20:45:16] DE genes 15 of a total of 300 ( 5 %)
INFO [2021-05-19 20:45:16] Using BRIII: 300 genes.
INFO [2021-05-19 20:45:16] Running SEA at cores: 4
INFO [2021-05-19 20:45:17] igsaInput1 : dEnricher finnished.
INFO [2021-05-19 20:45:17] igsaInput1 : mGSZ starting.
INFO [2021-05-19 20:45:17] Number of unique permutations: 197
INFO [2021-05-19 20:45:17] Getting ranking at cores: 4
INFO [2021-05-19 20:45:19] igsaInput1 : mGSZ finnished.
INFO [2021-05-19 20:45:19] igsaInput1 : IGSA analysis ended.
INFO [2021-05-19 20:45:19] ************************************
INFO [2021-05-19 20:45:19] igsaInput2 : Starting IGSA analysis.
INFO [2021-05-19 20:45:19] 60 Gene Sets.
INFO [2021-05-19 20:45:19] igsaInput2 : dEnricher starting.
INFO [2021-05-19 20:45:19] DE genes 6 of a total of 300 ( 2 %)
INFO [2021-05-19 20:45:19] Using BRIII: 300 genes.
INFO [2021-05-19 20:45:19] Running SEA at cores: 4
INFO [2021-05-19 20:45:20] igsaInput2 : dEnricher finnished.
INFO [2021-05-19 20:45:20] igsaInput2 : mGSZ starting.
INFO [2021-05-19 20:45:20] Number of unique permutations: 199
INFO [2021-05-19 20:45:20] Getting ranking at cores: 4
INFO [2021-05-19 20:45:22] igsaInput2 : mGSZ finnished.
INFO [2021-05-19 20:45:23] igsaInput2 : IGSA analysis ended.

> ## migsaRes contains the p-values obtained in each experiment for each gene set
> head(migsaRes)

     id Name     GS_Name  igsaInput1  igsaInput2
1  set1      myGeneSets1 0.009009155 0.616744852
2 set10      myGeneSets1 0.216721979 0.406003062
```

```
3 set11      myGeneSets1 0.406075459 0.449168928
4 set12      myGeneSets1 0.103740300 0.516416957
5 set13      myGeneSets1 0.406075459 0.803806865
6 set14      myGeneSets1 0.042134906 0.009405552

> ## Other possible analyses:
> ## If we want some gene sets to be evaluated in just one IGSAinput we
> ## can do this:
>
> ## If we want to test myGSs1 in exprData1 and myGSs2 in exprData2:
> igsaInput1 <- IGSAinput(
+   name = "igsaInput1", expr_data = exprData1,
+   fit_options = fitOpts, gene_sets_list = list(myGeneSets1 = myGSs1)
+ )
> igsaInput2 <- IGSAinput(
+   name = "igsaInput2", expr_data = exprData2,
+   fit_options = fitOpts, gene_sets_list = list(myGeneSets2 = myGSs2)
+ )
> experiments <- list(igsaInput1, igsaInput2)
> ## If we want to test myGSs1 in exprData1 and both in exprData2:
> igsaInput1 <- IGSAinput(
+   name = "igsaInput1", expr_data = exprData1,
+   fit_options = fitOpts, gene_sets_list = list(myGeneSets1 = myGSs1)
+ )
> igsaInput2 <- IGSAinput(
+   name = "igsaInput2", expr_data = exprData2,
+   fit_options = fitOpts,
+   gene_sets_list = list(myGeneSets1 = myGSs1, myGeneSets2 = myGSs2)
+ )
> experiments <- list(igsaInput1, igsaInput2)
> ## And this way, all possible combinations.
```

6. MIGSA's utility

In this section we are going to demonstrate **MIGSA**'s utility by analyzing several well known breast cancer datasets. For each dataset, subjects were classified into breast cancer intrinsic subtypes (Basal-Like, Her2-Enriched, Luminal B, Luminal A and Normal-Like) using the PAM50 algorithm (Parker *et al.* 2009) by means of the **pbcmc** R library (Fresno *et al.* 2016) and processed as suggested by Sorlie et al. (Sørlie *et al.* 2010). Only those subjects classified as Basal-Like or Luminal A were included.
Enrichment was tested over 20,245 Gene Ontology gene sets (14,291 biological processes, 1,692 cellular components and 4,263 molecular functions), and 179 from KEGG.

6.1. Used datasets

A total of eight datasets were tested, six of them were loaded by means of the **pbcmc** R

library, i.e., Mainz, Nki, Transbig, Unt, Upp and Vdx); and two were downloaded from the TCGA repository, i.e., microarray and RNAseq data matrices. For each dataset, genes reliably detected in less than 30% of the samples per condition were removed from the analysis. In addition, in RNAseq data, genes with a mean less than 15 counts per condition were also removed. Detailed datasets information can be seen in Table 2.

Table 2: Datasets details

Dataset	Platform	Subjects		Genes
		Basal	Luminal A	
Mainz	Microarray	18	117	13,091
Nki	Microarray	66	100	12,975
TCGA	Microarray	95	142	16,207
TCGA	RNAseq	95	142	16,741
Transbig	Microarray	37	89	13,091
Unt	Microarray	22	42	18,528
Upp	Microarray	19	150	18,528
Vdx	Microarray	80	134	13,091
Total	-	432	916	-

6.2. MIGSA on TCGA data

Let's run MIGSA over the TCGA RNAseq and microarray datasets. We are going to load both datasets using the **MIGSAdata** package, please refer to the gettingTcgaData vignette for details about these matrices.
NOTE: This chunk of code took 29.83m to execute on 10 cores.

```
> library(edgeR)
> library(limma)
> library(MIGSA)
> library(MIGSAdata)
> data(tcgaMAdata)
> data(tcgaRNAseqData)
> geneExpr <- tcgaMAdata$geneExpr
> rnaSeq <- tcgaRNAseqData$rnaSeq
> subtypes <- tcgaMAdata$subtypes
> # or tcgaRNAseqData$subtypes; are the same
> fitOpts <- FitOptions(subtypes)
> ## MA data: filter genes with less than 30% of genes read per condition
> dim(geneExpr)

[1] 16207   237

> geneExpr <- geneExpr[
+   rowSums(is.na(geneExpr[, subtypes == "Basal" ])) <
+     .3 * sum(subtypes == "Basal") &
+     rowSums(is.na(geneExpr[, subtypes == "LumA" ])) <
```

```
+       .3 * sum(subtypes == "LumA"),
+ ]
> dim(geneExpr)

[1] 16207   237

> ## create our IGSAinput object
> geneExpr <- new("MAList", list(M = geneExpr))
> geneExprIgsaInput <- IGSAinput(
+   name = "tcgaMA",
+   expr_data = geneExpr,
+   fit_options = fitOpts,
+   # with this treat we will get around 5% differentially expressed genes
+   sea_params = SEAparams(treat_lfc = 1.05)
+ )
> summary(geneExprIgsaInput)

INFO [2021-05-19 20:45:29] DE genes 802 of a total of 16207 ( 4.95 %)
     exp_name      #samples      contrast           #C1           #C2
     "tcgaMA"         "237" "BasalVSLumA"          "95"         "142"
   #gene_sets        #genes     treat_lfc     de_cutoff adjust_method
          "0"       "16207"        "1.05"        "0.01"         "fdr"
    #de_genes            br   perm_number     %de_genes
        "802"       "briii"         "200"        "4.95"

> ## RNAseq data: filter genes with less than 30% of genes read per
> ## condition and (below)
> dim(rnaSeq)

[1] 19948   237

> rnaSeq <- rnaSeq[
+   rowSums(is.na(rnaSeq[, subtypes == "Basal" ])) <
+     .3 * sum(subtypes == "Basal") &
+     rowSums(is.na(rnaSeq[, subtypes == "LumA" ])) <
+       .3 * sum(subtypes == "LumA"),
+ ]
> dim(rnaSeq)

[1] 19948   237

> ## a mean less than 15 counts per condition.
> rnaSeq <- rnaSeq[
+   rowMeans(rnaSeq[, subtypes == "Basal" ], na.rm = TRUE) >= 15 &
+     rowMeans(rnaSeq[, subtypes == "LumA"  ], na.rm = TRUE) >= 15,
+ ]
> dim(rnaSeq)
```

```
[1] 16741   237

> ## create our IGSAinput object
> rnaSeq <- DGEList(counts = rnaSeq)
> rnaSeqIgsaInput <- IGSAinput(
+   name = "tcgaRNA",
+   expr_data = rnaSeq,
+   fit_options = fitOpts,
+   # with this treat we will get around 5% differentially expressed genes
+   sea_params = SEAparams(treat_lfc = 1.45)
+ )
> summary(rnaSeqIgsaInput)

INFO [2021-05-19 20:45:36] DE genes 826 of a total of 16741 ( 4.93 %)
     exp_name      #samples      contrast           #C1           #C2
    "tcgaRNA"         "237" "BasalVSLumA"          "95"         "142"
   #gene_sets        #genes     treat_lfc     de_cutoff adjust_method
          "0"       "16741"        "1.45"        "0.01"         "fdr"
    #de_genes            br   perm_number     %de_genes
        "826"       "briii"         "200"        "4.93"

> experiments <- list(geneExprIgsaInput, rnaSeqIgsaInput)

> ## Not run:
>
> gSets <- list(
+   KEGG = downloadEnrichrGeneSets("KEGG_2015")[[1]],
+   BP = loadGo("BP"),
+   CC = loadGo("CC"),
+   MF = loadGo("MF")
+ )
> set.seed(8818)
> tcgaMigsaRes <- MIGSA(experiments, geneSets = gSets)
> ## Time difference of 29.83318 mins in 10 cores
> ## End(Not run)
```

6.3. MIGSA on pbcmc datasets

Let's run *MIGSA* over the pbcmc microarray datasets. We are going to load six datasets using the **MIGSAdata** package, please refer to the gettingPbcmcData vignette for details on how we got this matrices.
NOTE: This chunk of code took 1.27 hours to execute on 10 cores.

```
> library(limma)
> library(MIGSA)
> library(MIGSAdata)
```

```
> data(pbcmcData)
> ## with these treat log fold change values we will get around 5% of
> ## differentially expressed genes for each experiment
> treatLfcs <- c(0.7, 0.2, 0.6, 0.25, 0.4, 0.75)
> names(treatLfcs) <- c("mainz", "nki", "transbig", "unt", "upp", "vdx")
> experiments <- lapply(names(treatLfcs), function(actName) {
+   actData <- pbcmcData[[actName]]
+   actExprs <- actData$geneExpr
+   actSubtypes <- actData$subtypes
+
+   # filtrate genes with less than 30% per condition
+   actExprs <- actExprs[
+     rowSums(is.na(actExprs[, actSubtypes == "Basal" ])) <
+       .3 * sum(actSubtypes == "Basal") &
+       rowSums(is.na(actExprs[, actSubtypes == "LumA" ])) <
+         .3 * sum(actSubtypes == "LumA"),
+   ]
+
+   # create our IGSAinput object
+   actExprData <- new("MAList", list(M = actExprs))
+   actFitOpts <- FitOptions(actSubtypes)
+   actIgsaInput <- IGSAinput(
+     name = actName,
+     expr_data = actExprData,
+     fit_options = actFitOpts,
+     sea_params = SEAparams(treat_lfc = treatLfcs[[actName]])
+   )
+   return(actIgsaInput)
+ })

> ## Not run:
>
> gSets <- list(
+   KEGG = downloadEnrichrGeneSets("KEGG_2015")[[1]],
+   BP = loadGo("BP"),
+   CC = loadGo("CC"),
+   MF = loadGo("MF")
+ )
> set.seed(8818)
> pbcmcMigsaRes <- MIGSA(experiments, geneSets = gSets)
> ## Time difference of 1.26684 hours in 10 cores
> ## End(Not run)
```

6.4. MIGSA exploring breast cancer enrichment results

Let's start with the exploratory task. First, merge both MIGSAres objects into one with all the datasets results.

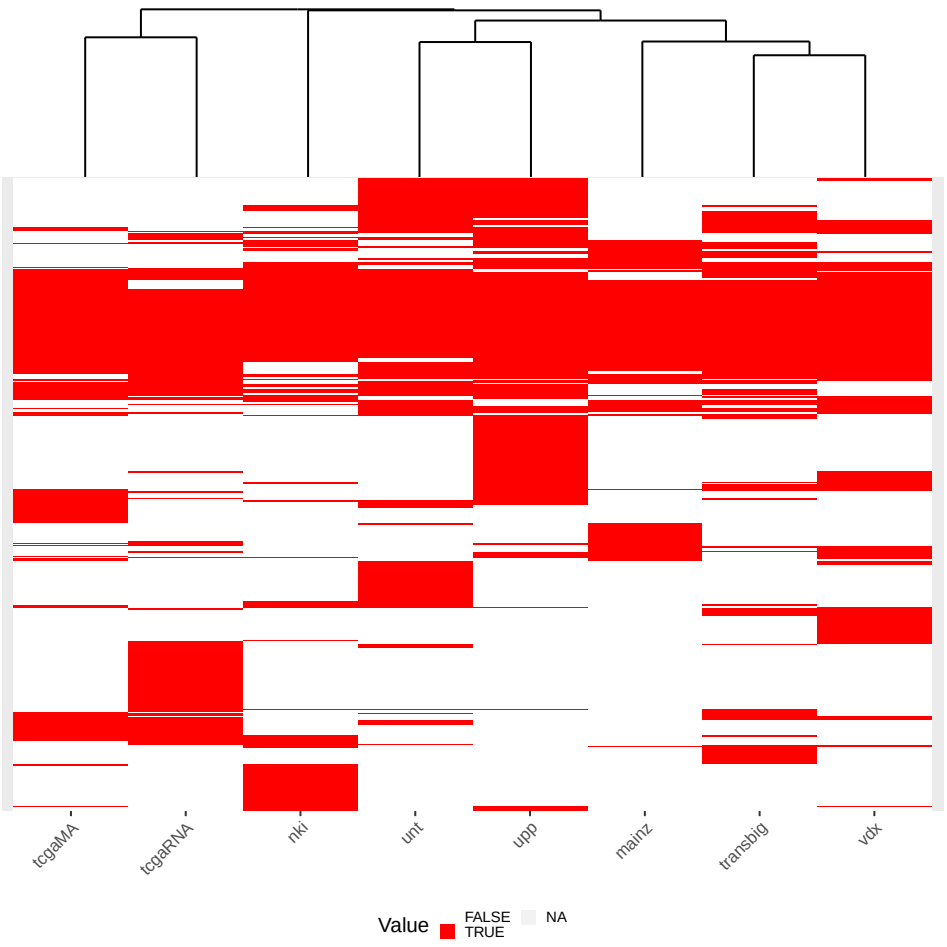
NOTE: In order to follow this code, sections 6.2 and 6.3 must have been executed. If not, jump to the next "End(Not run)" tag.

```
> ## Not run:
>
> dim(pbcmcMigsaRes)
> # [1] 20425     9
> dim(tcgaMigsaRes)
> # [1] 20425     5
>
> ## Let's merge both results in one big MIGSAres object
> bcMigsaRes <- merge(pbcmcMigsaRes, tcgaMigsaRes)
> dim(bcMigsaRes)
> # [1] 20425     11
> ## End(Not run)

> ## As last chunk of code was not executed, we load that data:
> library(MIGSA)
> library(MIGSAdata)
> data(bcMigsaResAsList)
> bcMigsaRes <- MIGSA:::MIGSAres.data.table(
+   bcMigsaResAsList$dframe,
+   bcMigsaResAsList$genesRank
+ )
> rm(bcMigsaResAsList)
> ## End(Loading data)
>
> ## Let's see a summary of enriched gene sets at different cutoff values
> summary(bcMigsaRes)

            mainz  nki tcgaMA tcgaRNA transbig  unt  upp  vdx
enr_at_0_01   655  768    754     889      821  958 1117  829
enr_at_0_05  1866 2217   2098    2224     1873 1992 2325 2148
enr_at_0_1   2948 3492   3185    3462     3137 3221 3612 3322

> ## We will set a cutoff of 0.01 (recommended)
> ## A gene set will be considered enriched if its p-value is < 0.01 on
> ## SEA or GSEA.
> bcMigsaRes <- setEnrCutoff(bcMigsaRes, 0.01)
> ## The bcMigsaRes data object that is included in MIGSA package is the
> ## following:
> # bcMigsaRes <- bcMigsaRes[1:200,];
```

Let's start exploring this MIGSA results object.

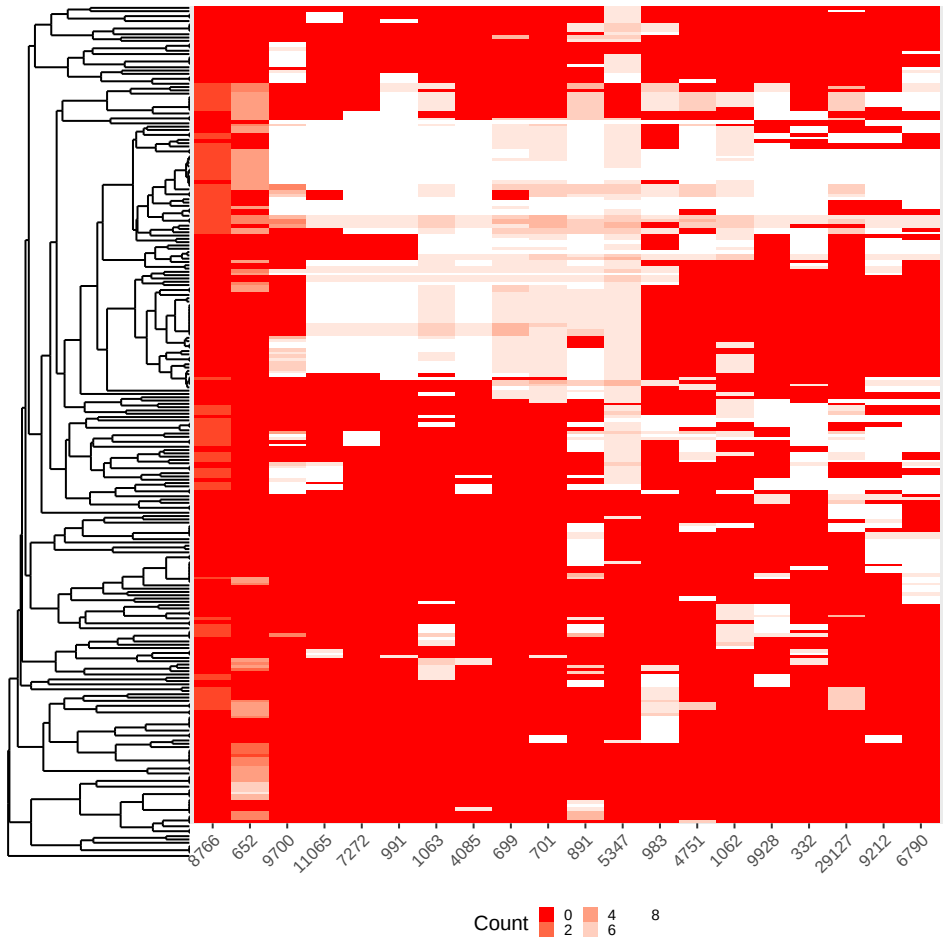
```
> colnames(bcMigsaRes)
```

```
 [1] "id"       "Name"     "GS_Name"  "mainz"    "nki"      "tcgaMA"
 [7] "tcgaRNA"  "transbig" "unt"      "upp"      "vdx"

> dim(bcMigsaRes)

[1] 20425    11

> summary(bcMigsaRes)

INFO [2021-05-19 20:45:48] Gene sets enriched in 0 experiments: 18191
INFO [2021-05-19 20:45:48] Gene sets enriched in 1 experiments: 921
INFO [2021-05-19 20:45:48] Gene sets enriched in 2 experiments: 377
INFO [2021-05-19 20:45:48] Gene sets enriched in 3 experiments: 231
INFO [2021-05-19 20:45:48] Gene sets enriched in 4 experiments: 150
INFO [2021-05-19 20:45:48] Gene sets enriched in 5 experiments: 104
INFO [2021-05-19 20:45:48] Gene sets enriched in 6 experiments: 96
INFO [2021-05-19 20:45:48] Gene sets enriched in 7 experiments: 113
INFO [2021-05-19 20:45:48] Gene sets enriched in 8 experiments: 242
$consensusGeneSets

    0     1     2     3     4     5     6     7     8
18191   921   377   231   150   104    96   113   242

$enrichmentIntersections
         mainz nki tcgaMA tcgaRNA transbig unt upp  vdx
mainz    655   393 397    372     489      421 485  477
nki      393   768 443    419     464      457 508  424
tcgaMA   397   443 754    525     495      503 532  451
tcgaRNA  372   419 525    889     460      457 480  434
transbig 489   464 495    460     821      550 605  573
unt      421   457 503    457     550      958 679  510
upp      485   508 532    480     605      679 1117 596
vdx      477   424 451    434     573      510 596  829

> ## We can see that 18,191 gene sets were not enriched, while 242 were
> ## enriched in every dataset.
> ## Moreover, there is a high consensus between datasets, with a maximum of 679
> ## enriched gene sets in common between upp and unt.
> ##
> ## Let's keep only gene sets enriched in at least one data set
> bcMigsaRes <- bcMigsaRes[ rowSums(bcMigsaRes[, -(1:3)], na.rm = TRUE) > 0, ]
> dim(bcMigsaRes)

[1] 2234   11

> ## Let's see enrichment heat map
> ## i.e. a heat map of binary data (enriched/not enriched)
> aux <- migsaHeatmap(bcMigsaRes)
```




```
> ## In this heat map we can see a high number of gene sets that are being
> ## enriched in consensus by most of the datasets. Let's explore them.
> ## We can obtain them (enriched in at least 80% of datasets) by doing
> consensusGsets <- bcMigsaRes[ rowSums(bcMigsaRes[, -(1:3)], na.rm = TRUE)
+ > 6.4, ]
> dim(consensusGsets)

[1] 355  11

> ## And let's see from which sets are them
> table(consensusGsets$GS_Name)

       BP         CC KEGG_2015         MF
      287         49          1         18

> ## Moreover, let's see which are the genes that are mostly contributing
> ## to gene set enrichment (genes contributing in at least 70 gene sets)
> ## i.e. a heat map showing the number of datasets in which each gene (columns)
> ## contributed to enrich each gene set (rows).
```

```
> aux <- genesHeatmap(bcMigsaRes,
+   enrFilter = 6.4, gsFilter = 70,
+   dendrogram = "col"
+ )
```




```
> ## Well, we could continue exploring them, however, at the first heat map we
> ## can see that TCGA datasets are defining a separate cluster, this is caused
> ## by a big group of gene sets that seem to be enriched mainly by TCGA.
> ## Let's explore them:
> ## (gene sets enriched by both TCGA datasets and in less than 20% of the other)
> tcgaExclusive <- bcMigsaRes[
+   rowSums(bcMigsaRes[, c("tcgaMA", "tcgaRNA")], na.rm = TRUE) == 2 &
+     rowSums(bcMigsaRes[, c("mainz", "nki", "transbig", "unt", "upp", "vdx")],
+       na.rm = TRUE
+     ) < 1.2,
+ ]
> dim(tcgaExclusive)

[1] 83 11
```

```
> table(tcgaExclusive$GS_Name)
```

```
       BP        CC KEGG_2015        MF
       62         3         1        17
```

```
> ## Let's see which is this KEGG enriched gene set
> tcgaExclusive[ tcgaExclusive$GS_Name == "KEGG_2015", "id" ]
```

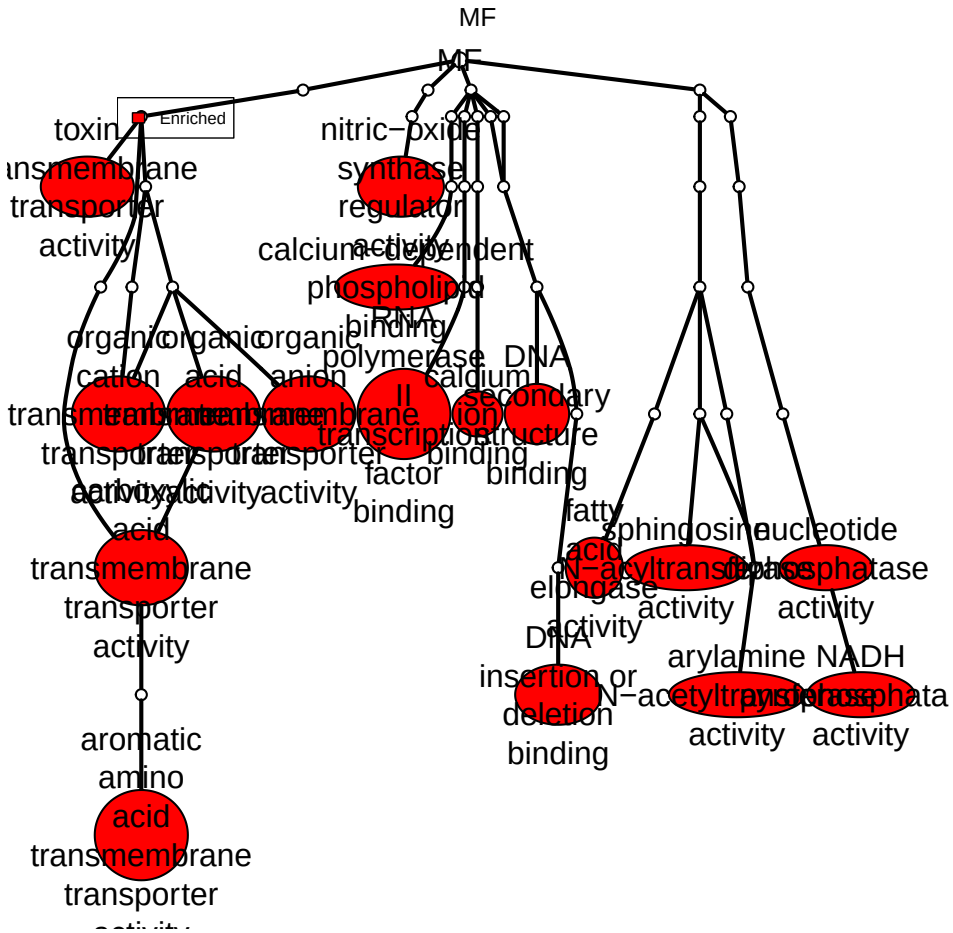
```
                       id
20362 nitrogen metabolism
```

```
> ## Let's see in which depths of the GO tree are these gene sets
> table(getHeights(
+   tcgaExclusive[ tcgaExclusive$GS_Name != "KEGG_2015", "id", drop = TRUE]
+ ))
```

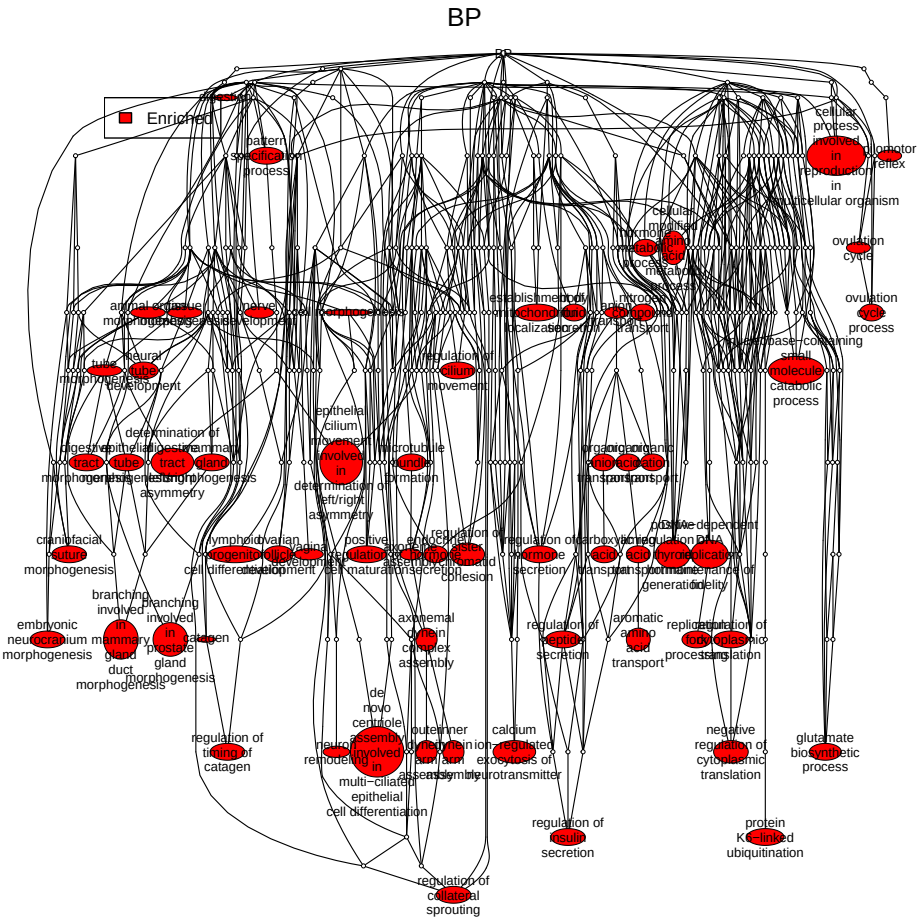
```
 2  3  4  5  6  7  8 10
 6 13 21 21 12  7  1  1
```

```
> ## We can see that the most of the gene sets are between depths three and five
```

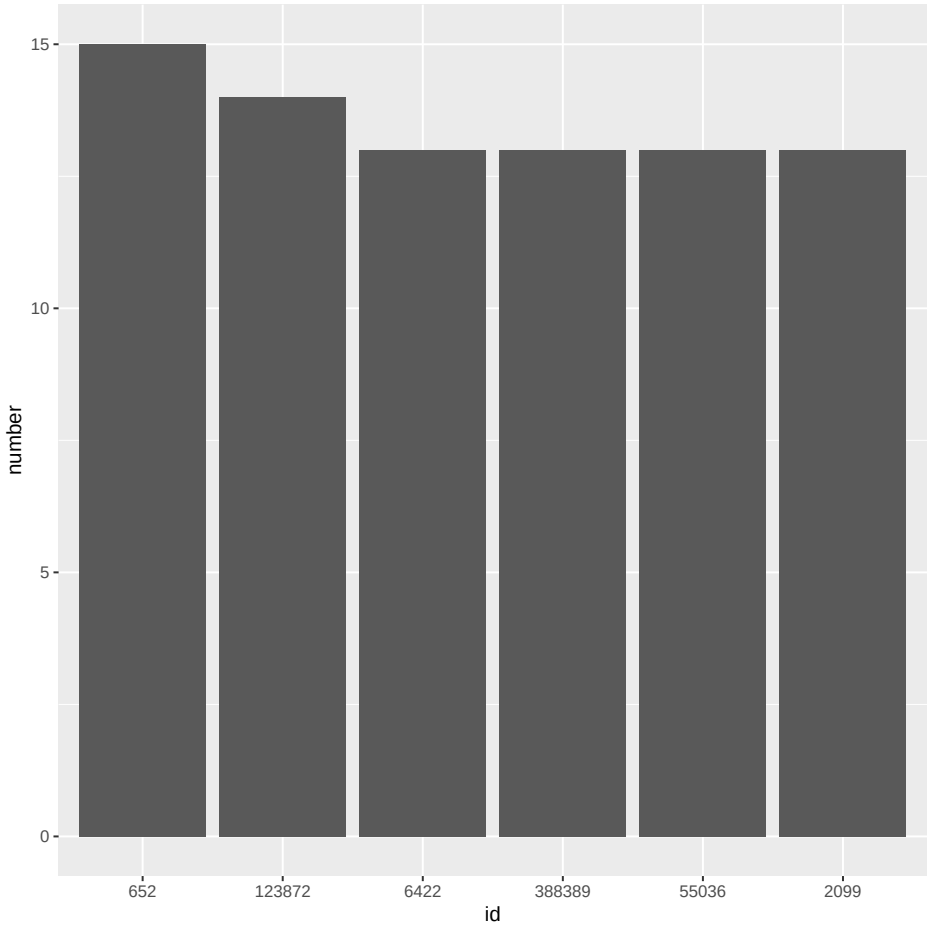
```
> ## And plot the GO tree of the other gene sets (except of CC, as it
> ## has only three gene sets, and it will look bad)
> aux <- migsaGoTree(tcgaExclusive, ont = "MF")
```




```
> aux <- migsaGoTree(tcgaExclusive, ont = "BP")
```




```
> ## Let's explore which are the genes that repeat the most in these
> ## gene sets (that are present in at least 15% of the gene sets)
> ## i.e. a bar plot of the number of gene sets in which each gene contributed to
> ## enrich.
> mostEnrichedGenes <- genesBarplot(tcgaExclusive, gsFilter = 12.45)
```

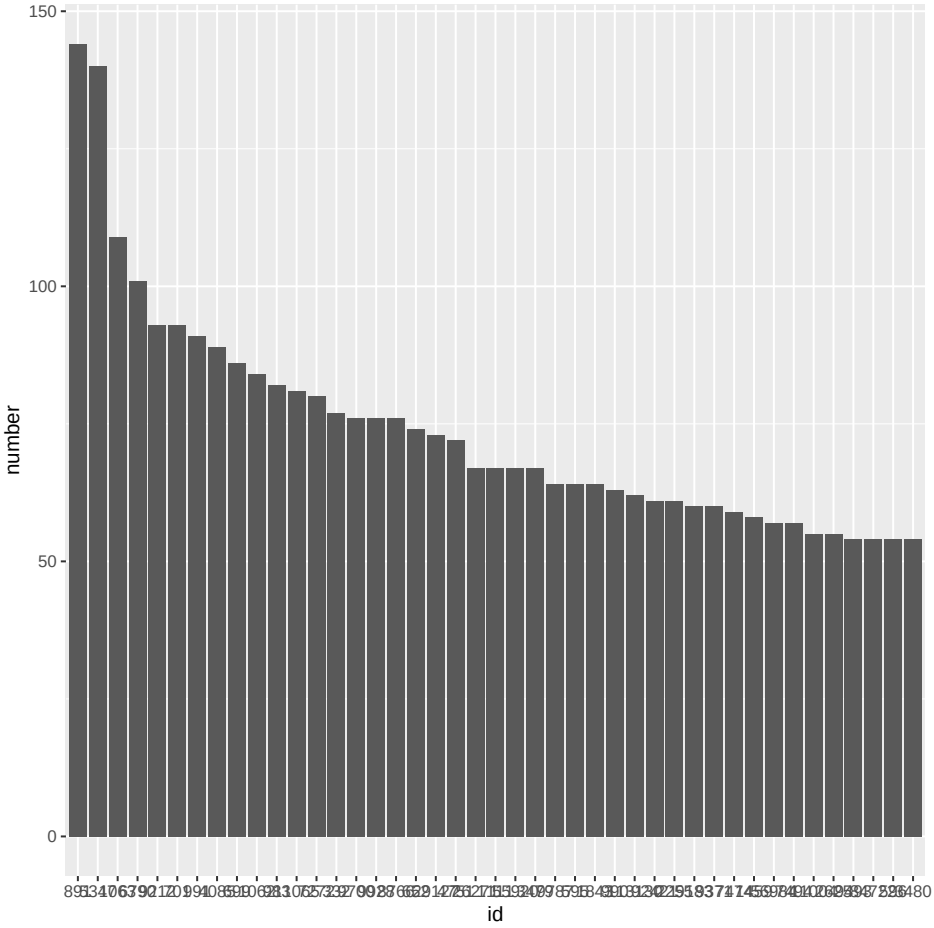



```
> mostEnrichedGenes$data
```

```
           id number
652       652     15
123872 123872     14
6422     6422     13
388389 388389     13
55036   55036     13
2099     2099     13
```

```
> ## Gene 652 is contributing to enrichment in 15 gene sets. And in total
> ## there are 6 genes that are being really active in TCGA enriched
> ## gene sets
> tcgaImportantGenes <- as.character(mostEnrichedGenes$data$id)
```

```
> ## Let's do the same analysis for the rest of the datasets, so we can filtrate
> ## which genes are acting exclusively in TCGA datasets
> consMostEnrichedGenes <- genesBarplot(consensusGsets, gsFilter = 53.25)
```




```
> consImportantGenes <- as.character(consMostEnrichedGenes$data$id)
> ## Let's see which genes they share
> intersect(tcgaImportantGenes, consImportantGenes)
```

```
[1] "652"
```

```
> ## And get the really tcga exclusive genes (5 genes)
> tcgaExclGenes <- setdiff(tcgaImportantGenes, consImportantGenes)
```

Another way of exploring the data is for example, suppose we have a list of genes of interest, we can filter our results having the gene sets that were enriched by our interest genes as follows:

```
> ## Let's sample 4 genes from consImportantGenes (as if they are our interest
> ## genes)
> set.seed(8818)
> myInterestGenes <- sample(consImportantGenes, 4)
> ## So we can get the filtered MIGSAres object by doing:
> intGenesMigsa <- filterByGenes(bcMigsaRes, myInterestGenes)
> dim(intGenesMigsa)
```

```
[1] 341  11
```

```
> head(intGenesMigsa)
```

```
            id                                          Name GS_Name mainz   nki
40  GO:0000070         mitotic sister chromatid segregation      BP  TRUE  TRUE
41  GO:0000075                       cell cycle checkpoint      BP  TRUE  TRUE
47  GO:0000086        G2/M transition of mitotic cell cycle      BP  TRUE  TRUE
69  GO:0000132 establishment of mitotic spindle orientation      BP  TRUE  TRUE
87  GO:0000166                          nucleotide binding      MF FALSE FALSE
116 GO:0000226        microtubule cytoskeleton organization      BP  TRUE  TRUE
    tcgaMA tcgaRNA transbig   unt  upp   vdx
40    TRUE    TRUE     TRUE  TRUE TRUE  TRUE
41    TRUE   FALSE     TRUE  TRUE TRUE  TRUE
47    TRUE    TRUE     TRUE  TRUE TRUE  TRUE
69    TRUE    TRUE     TRUE FALSE TRUE  TRUE
87   FALSE   FALSE    FALSE  TRUE TRUE FALSE
116   TRUE    TRUE     TRUE  TRUE TRUE  TRUE
```

And with this new MIGSAres object reproduce the same analysis done below.

Session Info

```
> sessionInfo()
```

```
R version 4.1.0 RC (2021-05-16 r80304)
Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS Mojave 10.14.6

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats4    parallel  stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] edgeR_3.35.0         MIGSAdata_1.15.0     MIGSA_1.17.0
 [4] mGSZ_1.0             ismev_1.42           mgcv_1.8-35
 [7] nlme_3.1-152         MASS_7.3-54          limma_3.49.0
[10] GSA_1.03.1           BiocParallel_1.27.0  GSEABase_1.55.0
```

```
[13] graph_1.71.0          annotate_1.71.0       XML_3.99-0.6
[16] AnnotationDbi_1.55.0 IRanges_2.27.0        S4Vectors_0.31.0
[19] Biobase_2.53.0       BiocGenerics_0.39.0

loaded via a namespace (and not attached):
 [1] Category_2.59.0         bitops_1.0-7            matrixStats_0.58.0
 [4] bit64_4.0.5             httr_1.4.2              GenomeInfoDb_1.29.0
 [7] Rgraphviz_2.37.0        tools_4.1.0             utf8_1.2.1
[10] R6_2.5.0                vegan_2.5-7             DBI_1.1.1
[13] colorspace_2.0-1        permute_0.9-5           tidyselect_1.1.1
[16] bit_4.0.4               compiler_4.1.0          formatR_1.9
[19] ggdendro_0.1.22         labeling_0.4.2          scales_1.1.1
[22] genefilter_1.75.0       RBGL_1.69.0             digest_0.6.27
[25] stringr_1.4.0           AnnotationForge_1.35.0 XVector_0.33.0
[28] pkgconfig_2.0.3         fastmap_1.1.0           rlang_0.4.11
[31] rstudioapi_0.13         RSQLite_2.2.7           farver_2.1.0
[34] GOstats_2.59.0          generics_0.1.0          jsonlite_1.7.2
[37] dplyr_1.0.6             RCurl_1.98-1.3          magrittr_2.0.1
[40] GO.db_3.13.0            GenomeInfoDbData_1.2.6 futile.logger_1.4.3
[43] Matrix_1.3-3            Rcpp_1.0.6              munsell_0.5.0
[46] fansi_0.4.2             lifecycle_1.0.0         stringi_1.6.2
[49] zlibbioc_1.39.0         org.Hs.eg.db_3.13.0     plyr_1.8.6
[52] grid_4.1.0              blob_1.2.1              crayon_1.4.1
[55] lattice_0.20-44         Biostrings_2.61.0       splines_4.1.0
[58] KEGGREST_1.33.0         locfit_1.5-9.4          pillar_1.6.1
[61] reshape2_1.4.4          futile.options_1.0.1    glue_1.4.2
[64] lambda.r_1.2.4          data.table_1.14.0       png_0.1-7
[67] vctrs_0.3.8             gtable_0.3.0            purrr_0.3.4
[70] assertthat_0.2.1        cachem_1.0.5            ggplot2_3.3.3
[73] xtable_1.8-4            survival_3.2-11         tibble_3.1.2
[76] memoise_2.0.0           cluster_2.1.2           ellipsis_0.3.2
```

References

Fang H, Gough J (2014). "The dnet'approach promotes emerging research on cancer patient survival." *Genome medicine*, **6**(8), 1.

Fresno C, Gonzalez GA, Llera AS, Fernandez EA (2016). *pbcmc: Permutation-Based Confidence for Molecular Classification*. R package version 1.3.2, URL http://www.bdmg.com.ar/.

Mishra P, Törönen P, Leino Y, Holm L (2014). "Gene set analysis: limitations in popular existing methods and proposed improvements." *Bioinformatics*, **30**(19), 2747–2756.

Parker JS, Mullins M, Cheang MC, Leung S, Voduc D, Vickery T, Davies S, Fauron C, He X, Hu Z, *et al.* (2009). "Supervised risk predictor of breast cancer based on intrinsic subtypes." *Journal of clinical oncology*, **27**(8), 1160–1167.

Rodriguez JC, González GA, Fresno C, Llera AS, Fernández EA (2016). "Improving information retrieval in functional analysis." *Computers in Biology and Medicine*, **79**, 10–20.

Sørlie T, Borgan E, Myhre S, Vollan HK, Russnes H, Zhao X, Nilsen G, Lingjærde OC, Børresen-Dale AL, Rødland E (2010). "The importance of gene-centring microarray data." *The lancet oncology*, **11**(8), 719–720.

Affiliation:

Juan C Rodriguez & Elmer A Fernández
Bioscience Data Mining Group
Facultad de Ingeniería
Universidad Católica de Córdoba - CONICET
X5016DHK Córdoba, Argentina
E-mail: jcrodriguez@bdmg.com.ar, efernandez@bdmg.com.ar
URL: http://www.bdmg.com.ar/