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. \texttt{dEnricher} and \texttt{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 \texttt{dEnricher} (Fang and Gough 2014) and \texttt{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 (**MIGSA**mGSZ) 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:


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:

```r
> ## 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.

```r
> 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)'
```

```r
> 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"
+ )
> gsets <- GeneSetCollection(list(gs1, gs2, gs3))
> gsets
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.

```r
> ## 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.

```r
> 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" ])) <
+ 0.3 * sum(subtypes == "Basal") &
+   rowSums(is.na(geneExpr[, subtypes == "LumA" ])) <
+ 0.3 * 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
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)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime</td>
<td>2.46</td>
<td>1.55</td>
<td>46.50</td>
<td>24.98</td>
<td>15.63</td>
<td>13.67</td>
<td>14.79</td>
<td>28.43</td>
</tr>
<tr>
<td>Speedup</td>
<td>1.00</td>
<td>1.58</td>
<td>3.18</td>
<td>5.91</td>
<td>9.45</td>
<td>10.81</td>
<td>9.98</td>
<td>5.19</td>
</tr>
<tr>
<td>Efficiency</td>
<td>1.00</td>
<td>1.58</td>
<td>1.59</td>
<td>1.48</td>
<td>1.18</td>
<td>1.08</td>
<td>0.83</td>
<td>0.37</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>#cores</th>
<th>mGSZ</th>
<th>MIGSAmGSZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime</td>
<td>2.46h</td>
<td>1.55h</td>
</tr>
<tr>
<td></td>
<td>46.5m</td>
<td>24.98m</td>
</tr>
<tr>
<td></td>
<td>15.63m</td>
<td>13.67m</td>
</tr>
<tr>
<td></td>
<td>14.79m</td>
<td>28.43m</td>
</tr>
<tr>
<td>Speedup</td>
<td>1</td>
<td>1.58</td>
</tr>
<tr>
<td></td>
<td>3.18</td>
<td>5.91</td>
</tr>
<tr>
<td></td>
<td>9.45</td>
<td>10.81</td>
</tr>
<tr>
<td></td>
<td>9.98</td>
<td>5.19</td>
</tr>
<tr>
<td>Efficiency</td>
<td>1</td>
<td>1.58</td>
</tr>
<tr>
<td></td>
<td>1.59</td>
<td>1.48</td>
</tr>
<tr>
<td></td>
<td>1.18</td>
<td>1.08</td>
</tr>
<tr>
<td></td>
<td>0.83</td>
<td>0.37</td>
</tr>
</tbody>
</table>

### 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.

```r
> 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
> ## 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 [2020-10-29 21:18:26] Number of unique permutations: 66
INFO [2020-10-29 21:18:26] Getting ranking at cores: 4

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

<table>
<thead>
<tr>
<th>gene.sets</th>
<th>pvalue</th>
<th>mGszScore</th>
</tr>
</thead>
<tbody>
<tr>
<td>set3</td>
<td>0.03130511</td>
<td>-1.815278</td>
</tr>
<tr>
<td>set40</td>
<td>0.05469617</td>
<td>2.748644</td>
</tr>
<tr>
<td>set46</td>
<td>0.05793451</td>
<td>-2.848961</td>
</tr>
<tr>
<td>set20</td>
<td>0.07557986</td>
<td>1.632850</td>
</tr>
<tr>
<td>set38</td>
<td>0.07695490</td>
<td>1.533487</td>
</tr>
<tr>
<td>set7</td>
<td>0.07785534</td>
<td>1.591581</td>
</tr>
</tbody>
</table>

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,
MIGSA: Massive and Integrative Gene Set Analysis

+ 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 [2020-10-29 21:18:28] *****************************************************
INFO [2020-10-29 21:18:28] *****************************************************
INFO [2020-10-29 21:18:28] DE genes 15 of a total of 300 ( 5 %)
INFO [2020-10-29 21:18:28] Running SEA at cores: 4
INFO [2020-10-29 21:18:29] Number of unique permutations: 197
INFO [2020-10-29 21:18:29] Getting ranking at cores: 4
INFO [2020-10-29 21:18:31] *****************************************************
INFO [2020-10-29 21:18:31] DE genes 6 of a total of 300 ( 2 %)
INFO [2020-10-29 21:18:31] Running SEA at cores: 4
INFO [2020-10-29 21:18:32] Number of unique permutations: 199
INFO [2020-10-29 21:18:34] igsaInput2 : mGSZ finnished.

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

<table>
<thead>
<tr>
<th>id</th>
<th>Name</th>
<th>GS_Name</th>
<th>igsaInput1</th>
<th>igsaInput2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>set1</td>
<td>myGeneSets1</td>
<td>0.009009155</td>
<td>0.616744852</td>
</tr>
<tr>
<td>2</td>
<td>set10</td>
<td>myGeneSets1</td>
<td>0.216721979</td>
<td>0.406003062</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Dataset</th>
<th>Platform</th>
<th>Subjects</th>
<th>Genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mainz</td>
<td>Microarray</td>
<td>18</td>
<td>117</td>
</tr>
<tr>
<td>Nki</td>
<td>Microarray</td>
<td>66</td>
<td>100</td>
</tr>
<tr>
<td>TCGA</td>
<td>Microarray</td>
<td>95</td>
<td>142</td>
</tr>
<tr>
<td>TCGA</td>
<td>RNAseq</td>
<td>95</td>
<td>142</td>
</tr>
<tr>
<td>Transbig</td>
<td>Microarray</td>
<td>37</td>
<td>89</td>
</tr>
<tr>
<td>Unt</td>
<td>Microarray</td>
<td>22</td>
<td>42</td>
</tr>
<tr>
<td>Upp</td>
<td>Microarray</td>
<td>19</td>
<td>150</td>
</tr>
<tr>
<td>Vdx</td>
<td>Microarray</td>
<td>80</td>
<td>134</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>432</td>
<td>916</td>
</tr>
</tbody>
</table>

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.

```r
> 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 [2020-10-29 21:18:39] 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)
MIGSA: Massive and Integrative Gene Set Analysis

[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 = SEAparsm(treat_lfc = 1.45)
> )
> summary(rnaSeqIgsaInput)

INFO [2020-10-29 21:18:45] 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 gettingPbmcData 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)
+ })
>
> 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.

```r
> ## 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)
```
> dim(bcMigsaRes)
[1] 20425 11

> summary(bcMigsaRes)

INFO [2020-10-29 21:18:57] Gene sets enriched in 0 experiments: 18191
INFO [2020-10-29 21:18:57] Gene sets enriched in 1 experiments: 921
INFO [2020-10-29 21:18:57] Gene sets enriched in 5 experiments: 104
INFO [2020-10-29 21:18:57] Gene sets enriched in 7 experiments: 113

$consensusGeneSets

0 1 2 3 4 5 6 7 8
18191 921 377 231 150 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
vdx 485 508 532 480 605 679 1117 596

> ## 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:

```r
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:

```r
table(consensusGsets$GS_Name)
```

<table>
<thead>
<tr>
<th>BP</th>
<th>CC KEGG_2015</th>
<th>MF</th>
</tr>
</thead>
<tbody>
<tr>
<td>287</td>
<td>49</td>
<td>18</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>BP</th>
<th>CC</th>
<th>KEGG_2015</th>
<th>MF</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>3</td>
<td>1</td>
<td>17</td>
</tr>
</tbody>
</table>

> ## 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 10
2    7 13 22 21 12  6  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

<table>
<thead>
<tr>
<th>id</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>652</td>
<td>652</td>
</tr>
<tr>
<td>123872</td>
<td>123872</td>
</tr>
<tr>
<td>6422</td>
<td>6422</td>
</tr>
<tr>
<td>388389</td>
<td>388389</td>
</tr>
<tr>
<td>55036</td>
<td>55036</td>
</tr>
<tr>
<td>2099</td>
<td>2099</td>
</tr>
</tbody>
</table>

> ## 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)
> head(intGenesMigsa)

<table>
<thead>
<tr>
<th>id</th>
<th>Name</th>
<th>GS_Name</th>
<th>mainz</th>
<th>nki</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>GO:0000070 mitotic sister chromatid segregation</td>
<td>BP TRUE TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>41</td>
<td>GO:0000075 cell cycle checkpoint</td>
<td>BP TRUE TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>47</td>
<td>GO:0000086 G2/M transition of mitotic cell cycle</td>
<td>BP TRUE TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>69</td>
<td>GO:0000132 establishment of mitotic spindle orientation</td>
<td>BP TRUE TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>87</td>
<td>GO:0000166 nucleotide binding</td>
<td>MF FALSE FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>116</td>
<td>GO:0000226 microtubule cytoskeleton organization</td>
<td>BP TRUE TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

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

Session Info

> sessionInfo()

R version 4.0.3 (2020-10-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04.5 LTS

Matrix products: default
BLAS: /home/biocbuild/bbs-3.12-bioc/R/lib/libRblas.so
LAPACK: /home/biocbuild/bbs-3.12-bioc/R/lib/libRlapack.so

locale:
[1] LC_CTYPE=en_US.UTF-8 LC_NUMERIC=C
[3] LC_TIME=en_US.UTF-8 LC_COLLATE=C
[5] LC_MONETARY=en_US.UTF-8 LC_MESSAGES=en_US.UTF-8
[7] LC_PAPER=en_US.UTF-8 LC_NAME=C
[9] LC_ADDRESS=C LC_TELEPHONE=C

attached base packages:
[1] stats4 parallel stats graphics grDevices utils datasets
[8] methods base
other attached packages:
[1] edgeR_3.32.0 MIGSAdata_1.14.0 MIGSA_1.14.1
[4] mGSZ_1.0 ismev_1.42 mgcv_1.8-33
[7] nlme_3.1-150 MASS_7.3-53 limma_3.46.0
[10] GSA_1.03.1 BiocParallel_1.24.0 GSEABase_1.52.0
[13] graph_1.68.0 annotate_1.68.0 XML_3.99-0.5
[16] AnnotationDbi_1.52.0 IRanges_2.24.0 S4Vectors_0.28.0
[19] Biobase_2.50.0 BiocGenerics_0.36.0

loaded via a namespace (and not attached):
[1] ggplot2_3.3.2 bit64_4.0.5
[4] jsonlite_1.7.1 splines_4.0.3 RBL_1.66.0
[7] blob_1.2.1 Category_2.56.0 pillar_1.4.6
[10] RSQLite_2.2.1 lattice_0.20-41 glue_1.4.2
[13] digest_0.6.27 colorspace_1.4-1 Matrix_1.2-18
[16] plyr_1.8.6 pkgconfig_2.0.3 geneNet_1.72.0
[19] purrr_0.3.4 xtable_1.8-4 GO.db_3.12.0
[22] scales_1.1.1 tibble_3.0.4 farver_2.0.3
[25] generics_0.0.2 ggplot2_3.3.2 ellipsis_0.3.1
[28] survival_3.2-7 magrittr_1.5 crayon_1.3.4
[31] memoise_1.1.0 GOstats_2.56.0 vegan_2.5-6
[34] tools_4.0.3 data.table_1.13.2 org.Hs.eg.db_3.12.0
[37] formatR_1.5.0 lifecycle_0.2.0 matrixStats_0.57.0
[40] stringr_1.4.0 munsell_0.5.0 locfit_1.5-9.4
[43] cluster_2.1.0 lambda.r_1.2.4 compiler_4.0.3
[46] rlang_0.4.8 futile.logger_1.4.3 grid_4.0.3
[49] RCurl_1.98-1.2 AnnotationForge_1.32.0 labeling_0.4.2
[52] bitops_1.0-6 gtable_0.3.0 DBI_1.1.0
[55] reshape2_1.4.4 R6_2.5.0 dplyr_1.0.2
[58] bit_4.0.4 futile.options_1.0.1 permute_0.9-5
[61] Rgraphviz_2.34.0 stringi_1.5.3 Rcpp_1.0.5
[64] vctrs_0.3.4 tidyselect_1.1.0

References


**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