MTGOsc: MTGO analysis on single cells

Nelson Nazzicari, Simone Marini, Danila Vella

2019-07-22

Contents

Example data: bladder basal epithelial cells

MTGOsc comes with some example data from the Mouse Atlas in the form of a (simplified for storage reasons) Seurat object. Let's start with loading Seurat and MTGO:

```
library(Seurat)
library(MTGOsc)
```

Loading MTGOsc also loads three objects: *bladder*, *markers*, and *mouse.pathways*.

```
#this is a (simplified) Seurat object
print(bladder)
```

```
## An old seurat object
##  269 genes across 1028 samples
```

```
#this come from applying Seurat function FindAllMarkers() to the bladder dataset
head(markers)
```

```
##                p_val avg_logFC pct.1 pct.2     p_val_adj
## krt15  1.523860e-202  1.959457 0.994 0.269 2.070317e-198
## igfbp2 3.880320e-196  1.857702 1.000 0.322 5.271803e-192
## trf    2.219344e-186  1.747788 0.896 0.189 3.015200e-182
## krt5   1.329655e-180  1.730088 0.976 0.276 1.806470e-176
## gsdmc2 1.235398e-172  1.711701 0.908 0.208 1.678412e-168
## gsto1  2.506894e-172  1.629429 0.985 0.299 3.405866e-168
##                                cluster   gene
## krt15  Basal epithelial cell(Bladder)  krt15
## igfbp2 Basal epithelial cell(Bladder) igfbp2
## trf    Basal epithelial cell(Bladder)    trf
## krt5   Basal epithelial cell(Bladder)   krt5
## gsdmc2 Basal epithelial cell(Bladder) gsdmc2
## gsto1  Basal epithelial cell(Bladder)  gsto1
```

```
#this is a simple gene -> pathway dictionary
head(mouse.pathways)
```

```
##    gene                                  pathway
## 1 43160                               Metabolism
## 2 43160                    Biological oxidations
## 3 43160 Phase I - Functionalization of compounds
## 4 43161                               Metabolism
```

```
## 5 43161                    Biological oxidations
## 6 43161 Phase I - Functionalization of compounds
```

Bladder object contains three clusters:

```
table(bladder@ident)
```

```
##
## Basal epithelial cell(Bladder) Stromal cell_Dpt high(Bladder)
##                            327                             651
##         Umbrella cell(Bladder)
##                             50
```

Calling MTGOsc

For this tutorial we'll focus on Basal epitelial cells cluster, which is of intermediate size. We create two "selector" arrays, one for cells and one for genes.

```
cells.selected = bladder@ident == 'Basal epithelial cell(Bladder)'
genes.selected = subset(markers, cluster == 'Basal epithelial cell(Bladder)')$gene

my.bladder = bladder
my.bladder@data = my.bladder@data[genes.selected, cells.selected]
```

MTGOsc needs a folder where to save temporary files and final results.

```
root = tempdir() #change this to your preferred local path
dir.create(root, recursive = TRUE, showWarnings = FALSE)
```

We are now ready to start with MTGOsc:

```
#building a genes-pathways dictionary
dict = write.dictionary(genes=mouse.pathways$gene,
                        terms = mouse.pathways$pathway, outfolder = root)

#computign gene coexpression (default function is 'cor')
coexp = write.coexpressionMatrix(geneExpression = my.bladder@data, outfolder = root)

#thinning coexpression network via scale free criterion
edges = write.edges(coexpression = coexp, outfolder = root,
                    keep.weights = FALSE, fun = thinning_scale_free)
```

```
## gamma: 2.98456061941471 (target:2)
## threshold: 0.9
## network edges: 18
```

```
#writing a parameter file, useful for MGTO
write.paramFile(outfolder = root)
```

All the groundwork is done and we can invoke MTGO:

```
#actual call to MTGO
#call.MTGO(outfolder = root, verbose = TRUE) #this prints several log messages

#building and saving representation of resulting network
#network.collapsed = export.network.modules(infolder = root, collapse.modules = TRUE)
#network.full = export.network.modules(infolder = root, collapse.modules = FALSE)
```

At this point networks are saved on disk, in "Network" subfolder, in the form of two html files. In the first one (ClusterNetwork.html) each functional module is collapsed to a single node, the second one (FullNetwork.html) where each gene is represented and functional modules are color coded.

Gene module enrichment on Reactome

Here we look for Reactome pathway enrichment of the genes constituting the thinned network. This procedure is complementary to the exctraction of Reactome pathways by MTGO-SC.

```
# load libraries for gene enrichment on Reactome (those are on Bioconductor, not CRAN)
library(ReactomePA)
library(clusterProfiler)
library(org.Mm.eg.db)

#a support function to take care of gene upper/lower case convention
firstup = function(x) {
  x = tolower(x)
  substr(x, 1, 1) = toupper(substr(x, 1, 1))
  return(x)
}

#the list of all genes involved in the cluster
genes = unique(c(as.character(edges$gene1), as.character(edges$gene2)))

#correct casing of gene names
genes = firstup(genes)

#translating gene names to ENTREZID via org.Mm.eg.db database
genes = bitr(genes, fromType="SYMBOL", toType="ENTREZID", OrgDb="org.Mm.eg.db")

#the actual enrichment
enriched = enrichPathway(gene=genes$ENTREZID, pvalueCutoff=0.05,
                         readable=T, organism = "mouse", pAdjustMethod = "BH")
```

Comparing network extraction methods

In the previous example we extracted the gene interaction network that is fed to MTGO via a two step approach:

1. computing gene coexpression (default function: *cor*)
2. network thinning (we used the function *scale_free_threshold*)

MTGOsc package offers several options for both steps, and it is always possible for the user to write and use their own functions. As such we propose an objective approach based on an (external) ground truth to select the best combination of functions.

In general terms, we'll select the method that maximise the *Affinity Score* (AS), which is a metric of how close the extracted network is to a selected external ground truth. As such, the first thing we'll need is said ground truth, which luckily is already saved in MTGOsc:

```
#gGT is an iGraph object derived from four different sources:
# - the CORUM database [1]
# - a protein interaction map [2]
```

```
# - String and Reactome databases [3]
summary(gGT)
```

```
## IGRAPH 2229188 UN-- 14324 1007972 --
## + attr: name (v/c)
```

For more details see the following references:

1 Giurgiu M, Reinhard J, Brauner B, Dunger-Kaltenbach I, Fobo G, Frishman G, Montrone C, Ruepp A. CORUM: the comprehensive resource of mammalian protein complexes-2019. Nucleic Acids Res. 2018 Oct 24. doi: 10.1093/nar/gky973. [Epub ahead of print] 30357367

2 Ramani, Arun K., et al. "A map of human protein interactions derived from co-expression of human mRNAs and their orthologs." Molecular systems biology 4.1 (2008): 180.

3 Skinnider, Michael A., Jordan W. Squair, and Leonard J. Foster. "Evaluating measures of association for single-cell transcriptomics." Nature methods 16.5 (2019): 381.

For the sake of this tutorial we'll compare two methods for extracting the interaction network:

- METHOD 1:
  - Gene coexpression computed via Pearson's correlation (default *cor* function)
  - Network thinning to maximise scale free fit (*thinning_scale_free* function)
- METHOD 2:
  - Gene coexpression computed via proportionality metrics phi (*coexpr_propr* function)
  - Network thinning to keep the top ten percent of all gene-gene interactions (*thinning_percentile* function)

Extracted networks must be transformed in *iGraph* object, starting from the edges object, as follow:

```
#Edges for Method 1, computed above
edges.M1 = edges

#Edges for Method 2, on the same data
coexp.M2 = write.coexpressionMatrix(
  geneExpression = my.bladder@data, outfolder = root, fun = coexpr_propr, metric = 'phs', symmetrize = '
edges.M2 = write.edges(coexpression = coexp.M2, outfolder = root,
                   keep.weights = FALSE, fun = thinning_percentile, overwrite = TRUE)

#transforming into iGraph
network.M1 = igraph::graph_from_edgelist(as.matrix(edges.M1),directed = FALSE)
network.M2 = igraph::graph_from_edgelist(as.matrix(edges.M2),directed = FALSE)
```

The Affinity Scores can be directly computed as follows:

```
#support function to compute Affinity Score between two networks
compute_AS = function(N1, N2){
  E.N1 = length(igraph::E(N1))
  E.N2 = length(igraph::E(N2))
  overlap = length(igraph::E(igraph::intersection(N1, N2)))

  #the formula is [overlap^2] / (edges1 * edges2)
  #but for better computation we implement it as
  res = (overlap / E.N1) * (overlap / E.N2)

  #and we are done
  return(res)
}
```

```
#ready to compute Affinity Scores for the examples
AS.M1 = compute_AS(network.M1, gGT)
AS.M2 = compute_AS(network.M2, gGT)
```

However they cannot be directly compared yet. For a proper comparison we should compute for each extracted network a random population of scrambled networks and then compare each AS to the distribution of scores. This will allow us to compute Z-scores and consequently P values associated to each extracted method. We can then (finally!) compare the logarithms of the P-values.

```
#numerosity for each random population
N = 100

#room for Affinity Scores for the two scrambled networks populations
AS.distr.M1 = c()
AS.distr.M2 = c()

#doing the computation
for (i in 1:100){
  #creating a random scramble of both networks
  random.network.M1 = rewire(network.M1,with=keeping_degseq(niter = vcount(network.M1) * 10,loops = FALS
  random.network.M2 = rewire(network.M2,with=keeping_degseq(niter = vcount(network.M2) * 10,loops = FALS

  #computing the new Affinity Scores
  AS.distr.M1[i] = compute_AS(random.network.M1, gGT)
  AS.distr.M2[i] = compute_AS(random.network.M2, gGT)
}

#Computing Z scores
Z.M1 = (AS.M1 - mean(AS.distr.M1)) / sd(AS.distr.M1)
Z.M2 = (AS.M2 - mean(AS.distr.M2)) / sd(AS.distr.M2)

#Computing -log(P values)
logP.M1 = -log(pnorm(-Z.M1))
logP.M2 = -log(pnorm(-Z.M2))
```

At this point the methods can be compared directly. In our case Method 2 shows higher values and should thus be preferred.