Using the `attract` Package To Find the Gene Expression Modules That Represent The Drivers of Kauffman's Attractor Landscape

Jessica Mar

May 2, 2019

1 Introduction

A mammalian organism is made up of over 200 types of specialized cells. Each cell type carries out a specific task integral to maintaining homeostasis of the organism. Cell types can vary by morphology, structure, lifespan, functional ability and much more. Despite such remarkable diversity, all cells within an organism are derived from an original precursor cell, and in most cases, share the same genome. Diversity comes about largely through differential expression programs where cells regulate the abundance of certain gene transcripts and their downstream molecules such as proteins and microRNAs. While epigenetic regulation also plays a role, finding the gene expression signatures represents the key to understanding the mechanisms underlying the unique properties that cells acquire. The attractor hypothesis proposed by Stuart Kauffman [1] describes how cell fate transitions between cell types occur through coordinated changes in genome-wide gene expression. In our approach [2], we show how by identifying the biological processes that are differentially activated between cell types, we can find the drivers of Kauffman's attractor landscape.

2 Filter Data Set

Before starting the analysis you may want to filter your data if using an RNA sequencing data set. To do this use the funtion filterDataSet on a data frame

of expression data. To filter we remove all rows (genes) where the percentage of samples with an expression value of 0 is 75 percent or higher. Then we add 1 to all the expression values and take the log base 2 of the data. You may also change the minimum percentage of samples with an expression value of 0 to remove genes with the filterPerc parameter.

```
> library(attract)
> filteredData <- filterDataSet(data, filterPerc=0.75)
```

3 Experimental Data Set

To illustrate our approach, we demonstrate the functions in the `attract` package on a gene expression data set published by Müller et al. [3] (NCBI GEO accession number GSE11508). The primary goal of this study was to elucidate the transcription profiles characterizing different stem cell lines and their progenitors. For our purposes, we have selected a subset of four cell lines - embryonic stem cells, neural stem cells, neural progenitors and teratoma-differentiated cells. These lines span a spectrum of pluripotent abilities and have also been derived from a range of different tissue sources.

The Müller expression data set is stored as a matrix object `exprs.dat`, and the corresponding cell line information is in the data.frame `samp.info`.

```
> library(attract)
> data(exprs.dat)
> data(samp.info)
```

The functions in our package operate off Bioconductor `ExpressionSet` objects.

```
> loring.eset <- new("ExpressionSet")
> loring.eset@assayData <- new.env()
> assign("exprs", exprs.dat, loring.eset@assayData)
> p.eset <- new("AnnotatedDataFrame", data=samp.info)
> loring.eset@phenoData <- p.eset
```

The first step of the analysis is to find the core pathway modules that show the most differential expression changes between the four different cell types. These core pathways represent the drivers of the attractor landscape.

4 Finding Core Attractor State Pathway Modules

The pathway modules as defined by KEGG or reactome and identified using the GSEAlm algorithm. These modules represent the pathways with the expression profiles that discriminate between the different celltypes or experimental groups of interest.

```
> attractor.states <- findAttractors(loring.eset,
+     "celltype", annotation = "illuminaHumanv1.db",
+     database = "KEGG", analysis = "microarray",
+     databaseGeneFormat = NULL, expressionSetGeneFormat = NULL)
```

You can also use MSigDB gene sets as well to find core attractor state pathway modules. When doing this, you must specify the databaseGeneFormat argument and the expressionSetGeneFormat argument. These arguments specify the type of gene identifier in the expression data set and the MSigDB gene sets. Since micro array data is being used expressionSetGeneFormat is set to PROBEID. Since the MSigDB genes are gene symbols, databaseGeneFormat is set to ENTREZID. To get the full range of options of what databaseGeneFormat and expressionSetGeneFormat can be set to, use the command columns(<yourAnnotationPackage>) to get the options for expressionSetGeneFormat and keytypes(<yourAnnotationPackage>) to get the options for databaseGeneFormat. When using RNAseq data sets, the argument expressionSetGeneFormat must always be declared. To get the options of what to put in, use keytypes(<yourAnnotationPackage>). Examples are "ENSEMBL", "ENTREZID", or "SYMBOL"

```
> columns(illuminaHumanv1.db)
```

```
 [1] "ACCNUM"       "ALIAS"        "ENSEMBL"
 [4] "ENSEMBLPROT"  "ENSEMBLTRANS" "ENTREZID"
 [7] "ENZYME"       "EVIDENCE"     "EVIDENCEALL"
[10] "GENENAME"     "GO"           "GOALL"
[13] "IPI"          "MAP"          "OMIM"
[16] "ONTOLOGY"     "ONTOLOGYALL"  "PATH"
[19] "PFAM"         "PMID"         "PROBEID"
```

```
[22] "PROSITE"      "REFSEQ"       "SYMBOL"
[25] "UCSCKG"       "UNIGENE"      "UNIPROT"

> keytypes(illuminaHumanv1.db)

 [1] "ACCNUM"       "ALIAS"        "ENSEMBL"
 [4] "ENSEMBLPROT"  "ENSEMBLTRANS" "ENTREZID"
 [7] "ENZYME"       "EVIDENCE"     "EVIDENCEALL"
[10] "GENENAME"     "GO"           "GOALL"
[13] "IPI"          "MAP"          "OMIM"
[16] "ONTOLOGY"     "ONTOLOGYALL"  "PATH"
[19] "PFAM"         "PMID"         "PROBEID"
[22] "PROSITE"      "REFSEQ"       "SYMBOL"
[25] "UCSCKG"       "UNIGENE"      "UNIPROT"

> MSigDBpath <- system.file("extdata", "c4.cgn.v5.0.entrez.gmt",
+     package = "attract")
> attractor.states.cutsom <- findAttractors(loring.eset,
+     "celltype", annotation = "illuminaHumanv1.db",
+     database = MSigDBpath, analysis = "microarray",
+     databaseGeneFormat = "ENTREZID", expressionSetGeneFormat = "PROBEID")
```

The output of the `findAttractors` object is an S4 class `AttractorModuleSet` object.

It contains the following slots:

```
> class(attractor.states)

[1] "AttractorModuleSet"
attr(,"package")
[1] "attract"

> slotNames(attractor.states)

[1] "eSet"            "cellTypeTag"     "incidenceMatrix"
[4] "rankedPathways"
```

where:

- `eSet` - is the `ExpressionSet` object that was supplied as input to the `findAttractors` function.
- `cellTypeTag` - is the character string denoting which variable in the `pData(eSet)` object stores the cell type or experimental group of interest info. Note that `cellTypeTag` must be one of `colnames(pData(eSet))`.
- `incidenceMatrix` - is incidence matrix where rows correspond to KEGG or reactome pathways, columns correspond to genes. An entry of 1 at location (X,Y) in the matrix indicates membership of gene Y in pathway X (0 indicates non-membership).
- `rankedPathways` - is a `data.frame` object that lists the KEGG or reactome pathway modules, ranked from most to least significant. The permutation P-values represent over-enrichment for each KEGG or reactome pathway from the `GSEAlm` package.

5 Removing Flat or Uninformative Genes

We next remove genes that show no significant expression changes across the four cell types.

```
> remove.these.genes <- removeFlatGenes(loring.eset,
+     "celltype", contrasts = NULL, limma.cutoff = 0.05)
```

This step runs a LIMMA model which tests for differences in expression between any of the four cell types. More specific differences can be tested instead by inputing these as a set of contrasts and supplying this object to the `contrasts` argument. See `?removeFlatGenes` for more info.

6 Finding the Synexpression Groups

Different cell or tissue types acquire their diversity by driving differentially coordinated expression patterns through interacting gene networks that are broadly captured by the pathways listed in the KEGG or reactome database. We next focus on elucidating what this repertoire of transcriptionally-coherent expression patterns being sustained within a pathway are, in other words the synexpression groups.

A synexpression group contains genes that share similar expression profiles across the four groups.

As an example, we use the MAPK signaling pathway module:

```
> mapk.syn <- findSynexprs("04010", attractor.states,
+     "celltype", remove.these.genes)

[1] "04010"
```

The output of `findSynexpress` is an S4 class `SynExpressionSet` object

```
> class(mapk.syn)

[1] "SynExpressionSet"
attr(,"package")
[1] "attract"

> slotNames(mapk.syn)

[1] "groups"   "profiles"
```

where:

- `groups` - is a `list` object containing the genes in each synexpression group (each component corresponds to a different synexpression group).
- `profiles` - is a `matrix` object that stores the average expression profiles for each synexpression group. The rows of the matrix correspond to the synexpression groups, the columns correspond to the samples in the data set.

```
> length(mapk.syn@groups)

[1] 8

> sapply(mapk.syn@groups, length)

[1] 42 23 28 28  9 20  8  8
```

The number of synexpression groups is determined by an informativeness metric [4]. There are 3 synexpression groups for the MAPK pathway module.

Using the same `findSynexprs` function, we can find the synexpression groups for the top 5 pathway modules:

```
> top5.syn <- findSynexprs(attractor.states@rankedPathways[1:5,
+     1], attractor.states, "celltype", removeGenes = remove.these.genes)
```

```
[1] 03010
218 Levels: 00010 00020 00030 00040 00051 00052 ... 05416
[1] 04512
218 Levels: 00010 00020 00030 00040 00051 00052 ... 05416
[1] 04510
218 Levels: 00010 00020 00030 00040 00051 00052 ... 05416
[1] 05146
218 Levels: 00010 00020 00030 00040 00051 00052 ... 05416
[1] 00190
218 Levels: 00010 00020 00030 00040 00051 00052 ... 05416
```

Note in this case, the output object of `findSynexprs` is an `environment` variable.

The keys are given as:

```
> ls(top5.syn)
```

```
[1] "pway00190synexprs" "pway03010synexprs"
[3] "pway04510synexprs" "pway04512synexprs"
[5] "pway05146synexprs"
```

Each of the values is stored as an individual `SynExpressionSet` object:

```
> class(get(ls(top5.syn)[1], top5.syn))
```

```
[1] "SynExpressionSet"
attr(,"package")
[1] "attract"
```

7 Visualizing the Synexpression Groups for a Core Attractor Pathway

We can visualize the average expression profiles of the synexpression groups using base R's `plot` functions or alternatively using `plotsynexprs`.

```
> par(mfrow = c(2, 2))
> pretty.col <- rainbow(3)
> for (i in 1:3) {
+     plotsynexprs(mapk.syn, tickMarks = c(6, 28,
+         47, 60), tickLabels = c("ESC", "PRO",
+         "NSC", "TER"), vertLines = c(12.5, 43.5,
+         51.5), index = i, main = paste("Synexpression Group ",
+         i, sep = ""), col = pretty.col[i])
+ }
```

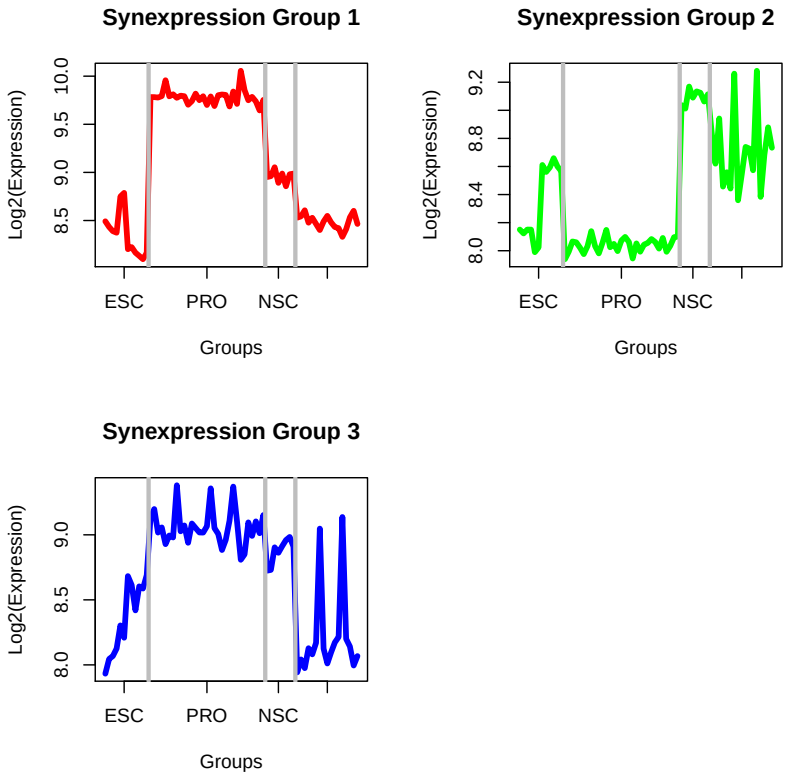

Figure 1: Average Expression Profiles of the Synexpression Groups

8 Finding Correlated Partners of the Synexpression Groups

In our approach so far, the attractor pathway modules and their following synexpression groups have been inferred from information restricted to well-curated sources like KEGG. As a result, these inferences are of high quality and we have strong confidence in their applicability to the system under study. These benefits in accuracy clearly come at the expense of low coverage because we are ultimately only describing a small proportion of the genome.

However by using the synexpression groups to pick up genes with highly correlated expression profiles, we can then extrapolate out to make inferences about the entire set of genes in the genome. Genes with highly correlated profiles to the synexpression groups (e.g. $R > 0.85$) are also likely to be integral in maintaining cell type-specific differences, however due to their lack of inclusion in resources like KEGG, would not have been picked up by the first GSEA step.

For the MAPK synexpression groups, we can find out what other genes on the chip share similar expression profiles.

```
> mapk.cor <- findCorrPartners(mapk.syn, loring.eset, remove.these.genes)
```

The output of `findCorrPartners` is a list which stores vectors of genes that are highly correlated with at least one gene in the synexpression expression group. More or less stringent criterion can be applied by changing the `cor.cutoff` argument (default setting is 0.85).

```
> sapply(mapk.cor@groups, length)
```

```
[1] 1513  356  343 1460    3   99   75   28
```

9 Functional Enrichment Analysis of the Synexpression Groups

For each of these correlated sets, we look for functional enrichment using GO terms to learn about any trends in common roles that these genes may potentially share.

```
> mapk.func <- calcFuncSynexprs(mapk.syn, attractor.states,
+     "CC", annotation = "illuminaHumanv1.db", analysis = "microarray",
+     expressionSetGeneFormat = NULL)
```

10 References

[1] Kauffman S. 2004. A proposal for using the ensemble approach to understand genetic regulatory networks. J Theor Biol. 230:581.

[2] Mar JC, Matigian NA, Quackenbush J, Wells CA. 2011. attract: A method for identifying core pathways that define cellular phenotypes. PLos One. 6(10):e25445

[3] Müller F et al. 2008. Regulatory networks define phenotypic classes of human stem cell lines. Nature. 455(7211): 401.

[4] Mar JC, Wells CA, Quackenbush J. 2011. Defining an informativeness metric for clustering gene expression data. Bioinformatics. 27(8): 1094.

11 Session Information

```
R version 3.6.0 (2019-04-26)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04.2 LTS

Matrix products: default
BLAS:   /home/biocbuild/bbs-3.9-bioc/R/lib/libRblas.so
LAPACK: /home/biocbuild/bbs-3.9-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
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
```

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

other attached packages:
[1] GO.db_3.8.2                illuminaHumanv1.db_1.26.0
[3] org.Hs.eg.db_3.8.2         attract_1.36.0
[5] AnnotationDbi_1.46.0       IRanges_2.18.0
[7] S4Vectors_0.22.0           Biobase_2.44.0
[9] BiocGenerics_0.30.0

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.1              compiler_3.6.0
 [3] XVector_0.24.0          bitops_1.0-6
 [5] tools_3.6.0             zlibbioc_1.30.0
 [7] digest_0.6.18           bit_1.1-14
 [9] annotate_1.62.0         RSQLite_2.1.1
[11] memoise_1.1.0           lattice_0.20-38
[13] pkgconfig_2.0.2         png_0.1-7
[15] graph_1.62.0            Matrix_1.2-17
[17] DBI_1.0.0               Category_2.50.0
[19] Rgraphviz_2.28.0        curl_3.3
[21] cluster_2.0.9           genefilter_1.66.0
[23] httr_1.4.0              Biostrings_2.52.0
[25] GOstats_2.50.0          bit64_0.9-7
[27] grid_3.6.0              GSEABase_1.46.0
[29] R6_2.4.0                survival_2.44-1.1
[31] RBGL_1.60.0             XML_3.98-1.19
[33] limma_3.40.0            reactome.db_1.68.0
[35] blob_1.1.1              splines_3.6.0
[37] KEGGREST_1.24.0         AnnotationForge_1.26.0
[39] xtable_1.8-4            RCurl_1.95-4.12
```