An Introduction to the R package `Rolemodel`

Zhishi Wang, Aimee Teo Broman, Subhrangshu Nandi, Bret Larget, Karl Broman and Michael Newton

Contents

1 Introduction

The `Rolemodel` package is intended to implement the model-based gene-set enrichment analysis method described in Wang *et al.* (2015). We call our method multi-functional analyzer, or MFA for short. Our method is based on the role model, first proposed by Bauer *et al.* (2010). In the role model, we have a finite number of wholes and parts; each whole is a set of parts. In the gene set analysis problem, gene sets can be treated as wholes and genes as parts; each gene set is a set of unordered genes. Two computational advances are developed to do the posterior inference: one is the integer linear programming (MFA-ILP), from which we can obtain the MAP (maximum a posteriori) estimate of active gene sets; the other is the penalized MCMC (MFA-MCMC), from which we can obtain the posterior active probability for each gene set. We would like to use MFA-ILP as our primary tool, as it gives a summary functional decoding of the gene list, while posterior probabilities from the MCMC computation provide a measure of confidence in the inferred sets.

In this articel, we describe how to use the functions in the `Rolemodel` package. The function `sequentialRM` performs the MFA-ILP computation, and the function `bp` performs the MFA-MCMC. Generally, the input for MFA include a list of experimentally derived genes (*e.g.,*

differentially expressed genes under different cell type conditions), the 0-1 incidence matrix characterizing the inclusion relationship between wholes and parts, the system parameters α (the false positive rate), γ (the true positive rate) and p (the prior active probability for whole nodes), where α, γ and p are all numeric values between 0 and 1, and α is supposed to be less than γ. We can estimate the values of α and γ from the gene-level data (see details in Wang *et al.*, 2015), and then estimate the value of p via the R package `mgsa`. In this article, let's assume that α, γ and p are known parameters.

2 Setup

Say we have a list of human genes, and we are interested in their gene-set enrichment. For this example, I will sample a random set of 100 genes from the `org.Hs.eg.db` database.

```
> library(org.Hs.eg.db)
> genes <- sample(mappedRkeys(org.Hs.egSYMBOL),100)
> head(genes)
```

```
[1] "ACADL"        "PEG3"         "KPNA3"        "OR5AQ1P"
[5] "LOC102723330" "RBMY1A3P"
```

Load the `Rolemodel` library.

```
> library(Rolemodel)
```

The `gs2edge` function will take this vector of genes, and create input objects for the `ILP` or `sequentialRM`, and `bp` functions. Gene-set enrichment is based on Entrez-ID gene IDs; if the gene list is composed of gene symbols, these will be converted to Entrez ID. `n.upp` and `n.low` are the maximum and minimum gene-set sizes to consider in the enrichment analysis. We recommend an `n.upp` <= 50 for `ILP` and `sequentialRM` due to computing time and memory allocation.

```
> gs <- gs2edge(genes, n.upp=30, idtype="SYMBOL", lib="org.Hs.eg")
> I <- gs$I
> I[1:6,1:6]
```

```
      GO:0000002 GO:0000012 GO:0000014 GO:0000028 GO:0000030 GO:0000038
100            0          0          0          0          0          0
1000           0          0          0          0          0          0
10000          1          0          0          0          0          0
10003          0          0          0          0          0          0
10005          0          0          0          0          0          0
10006          0          0          0          0          0          0
```

```
> y <- gs$y
> head(y)

  100  1000 10000 10003 10005 10006
    0     0     0     0     0     0

> edge <- gs$edge
> head(edge)

         go_id gene_id
3   GO:0001867       2
18  GO:0002921       2
34  GO:0007597       2
75  GO:0030449       2
108 GO:0045824       2
109 GO:0045916       2
```

3 Gene-set Enrichment

We will demonstrate the use of the `ILP`, `sequentialRM`, and `bp` using the `t2d` dataset, which is included in the `Rolemodel` library. To speed processing time, we consider gene-sets with 5 to 20 genes. Use the `subRM` function to subset the `I` matrix.

```
> data(t2d)
> dim(t2d$I)

[1] 10626  6037

> sum(t2d$y)

[1] 58

> newI <- subRM(t2d$I, 5, 20)
> newy <- t2d$y[rownames(newI)]
> dim(newI)

[1] 8437 4449

> sum(newy)

[1] 52
```

As mentioned before, we need to specify the values of α, γ and p before running MFA. The `p` parameter can be estimated using the `mgsa` library, via the `estP` function. In this example, we are already given `p=0.00331`. As an exercise, however, we estimate p:

```
> alpha <- 0.00019
> gamma <- 0.02279
> pest <- estP(I=newI, y=newy, alpha, gamma)
> pest

[1] 0.004495392
```

An internal study indicates that the algorithm is not overly sensitive to the exact settings of these parameters, but to their range.

3.1 MFA-ILP: MAP estimate via integer linear programming

We use the R package `Rglpk` to do the integer linear programming, which is an R interface to GNU Linear Programming Kit intending to solve large-scale linear programming problems. The function `ILP` in our package is a wrapper designed for the specific gene-set problem by invoking the function `Rglpk_solve_LP` in `Rglpk` directly.

```
>  alpha <- 0.00019
>  gamma <- 0.02279
>  p <- 0.00331
> # do not run
> # res <- ILP(newI, newy, alpha, gamma, p)
```

It is very time-consuming to run `ILP` directly when the dimension of the incidence matrix is kind of large. So one suggestion is that using the function `sequentialRM` rather than `ILP` to do the ILP related computation. `SequentialRM` is a wrapper to implement the two approaches (shrinkage and sequential strategy) developed in Wang *et al.* (2013) to speed up the computation.

```
> res <- sequentialRM(newI, newy, nupstart=10, by=1, alpha, gamma, p)

[1] 11
[1] 12
[1] 13
[1] 14
[1] 15
[1] 16
[1] 17
[1] 18
[1] 19
[1] 20
```

```
> str(res)

List of 2
 $ onwholes: chr [1:8] "GO:0001714" "GO:0045725" "GO:2000679" "GO:0006983" ...
 $ sol     :List of 3
  ..$ optimum : num 44.1
  ..$ solution: Named num [1:122] 0 0 0 0 0 0 1 1 1 1 ...
  .. ..- attr(*, "names")= chr [1:122] "GO:0032460" "GO:0032461" "GO:0032462" "GO:003246
  ..$ status  : int 0
```

Here `nupstart` indicates the starting upper bound of the number of genes in a set, which is used to subset the incidence matrix, and `by` indicates the increment of the upper bound for each iteration in the sequential strategy. One suggestion is to set `nupstart` ≤ 10, and `by` $= 1$. The output `res` is a list consisting of two parts. The first part `onwholes` includes the names of identified active GO terms (gene sets), *i.e.,* the MAP estimate. In this example, there are totally 8 GO terms identified as active. The second part `sol` is extracted directly from the output of function `Rglpk_solve_LP` in the package `Rglpk`, where `optimum` is the value of the objective function at the optimum, `solution` is the vector of optimal coefficients (0-1vector), and `status` is an integer with status information about the solution returned. It will return 0 for the optimal solution being found, and non-zero otherwise. Actually, the MAP estimate is just the whole nodes whose values are taking 1 in the optimal solution.

```
> names(res$sol$solution)[res$sol$solution == 1][1:8]

[1] "GO:0001714" "GO:0045725" "GO:2000679" "GO:0006983" "GO:0031017"
[6] "GO:0070365" "GO:0005638" "GO:0060395"
```

3.2 MFA-MCMC: the posterior active probability

The function `bp` is used to do the MCMC sampling. The input for `bp` is a bit different from that in `sequentialRM` and is listed as follows:

- `whole` Vector of character strings with names of whole nodes
- `part` Vector of 0's and 1's indicating active part nodes; names are the names of the part nodes
- `edge` Matrix with two columns; each row indicates an edge between a whole node and a part node
- `alpha, gamma, p` the same as before
- `nburn` Number of burn-in generations
- `ngen` Number of sample generations

- `sub` Subsample rate for burn-in and sample files
- `penalty` Penalty per illegal node to loglikelihood
- `initial` Initial states of whole nodes, which can take one of three values: `inactive` - all whole nodes inactive; `random` - all whole nodes active with probability `p`, no illegal nodes; or `high` - all nodes with proportion of connected part nodes with response equal to 1 above 0.4 are active, no illegal nodes.

```
> eout <- I2edge(newI)
>  bp.out <- bp(whole=colnames(newI), part=newy, edge=eout, alpha, gamma, p,
  nburn=1000000, ngen=10000000, sub=1000, penalty=5, initial="random")
>  str(bp.out)

'data.frame':        4449 obs. of  6 variables:
 $ Name             : chr  "GO:0000002" "GO:0000012" "GO:0000028" "GO:0000042" ...
 $ ActiveProbability: num  0 0.00264 0.00224 0 0.00169 ...
 $ Count            : int  0 23899 20238 0 15233 0 54438 0 17841 20482 ...
 $ Sample           : int  9040135 9040135 9040135 9040135 9040135 9040135 9040135 90401
 $ Degree           : int  13 8 6 14 11 10 10 11 5 12 ...
 $ Response         : int  0 0 0 0 0 0 0 0 0 0 ...
 - attr(*, "samples")= num [1:11000, 1:10] 1.91e-05 2.96e-03 2.70e-03 5.20e-12 1.37e-05
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr  "AcceptanceProb" "LogLikelihood" "LogPrior" "ActiveOne" ...
```

Among the output of `bp` (see the help file for a full explanation), we are most interest in the `ActiveProbability`, which is the posterior active probability for each whole node. Generally, cut-off value 0.5 is used to pick GO terms as active.

```
> bp.out$Name[bp.out$ActiveProbability>0.5]

[1] "GO:0001714" "GO:0006983" "GO:0031017" "GO:0045725" "GO:0060612"
[6] "GO:0005638"
```

As a result, these GO terms can be seen as the active ones inferred from the MCMC sampling.

3.3 An integrative analysis function: summary of the result

As mentioned before, we would like to use MFA-ILP as our primary tool, while posterior probabilities from the MCMC computation provide a measure of confidence in the inferred sets. Of course, one can use the MCMC sampling result independently.
We would like to report the MAP estimate in the decreasing order of the posterior active probabilities.

```
> tmp <- bp.out$ActiveProbability[bp.out$Name%in%res$onwholes]
> ord <- order(tmp, decreasing = T)
> output<- data.frame(MAP = res$onwholes[ord], P.MFA = tmp[ord])
> print(output)

         MAP     P.MFA
1 GO:0006983 0.9844145
2 GO:2000679 0.9158180
3 GO:0045725 0.8785884
4 GO:0060395 0.8645420
5 GO:0001714 0.5389740
6 GO:0031017 0.4746733
7 GO:0070365 0.3299888
8 GO:0005638 0.1852368
```

On the other hand, we have also created an overall function `rmTable` to do both MFA-ILP and MFA-MCMC analysis simultaneously. Thus users can take total control of the parameters and interpret the result in an easier way. The input here is just a list of genes of interest, e.g., which can be a list of differentially expressed genes according to some different cellular conditions. The type of this gene list can be in Entrez ids, gene symbols, Ensembles, etc. The output includes four parts: the first part is a summary data frame about the active gene sets, including their GO terms, set size, number of associated genes included, and so on; the second part is the solution of ILP calculation (the same as the output of `sequentialRM`); the third part is the solution of MCMC calculation (the same as the output of `bp`); and the last part is the incidence matrix and the response vector based on the input gene list. Notice that since the up-to-date data source is used here, the result is slightly different from the one obtained previously.

```
> data(T2D_genelist)
> idlist <- idlist[-57] ## no Entrez id for "KLHDC5"
> res <- rmTable (idlist, lib="org.Hs.eg", n.upp=20, n.low=5, nupstart=10, by=1, alpha

[1] 11
[1] 12
[1] 13
[1] 14
[1] 15
[1] 16
[1] 17
[1] 18
[1] 19
[1] 20

> str(res)
```

```
List of 4
 $ rm.table     :'data.frame':        6 obs. of  7 variables:
  ..$ Term                   : chr [1:6] "lamin filament" "axolemma" "positive regulatio
  ..$ Ontology               : chr [1:6] "CC" "CC" "BP" "BP" ...
  ..$ MAP                    : Factor w/ 6 levels "GO:0005638","GO:0030673",..: 1 2 3 5
  ..$ P.MFA                  : num [1:6] 0.995 0.978 0.902 0.896 0.86 ...
  ..$ set.size               : num [1:6] 5 12 15 20 7 6
  ..$ number.associated.genes: num [1:6] 2 2 5 3 2 4
  ..$ genes                  : Factor w/ 6 levels "AKT2;GCK;INS;INSR;IRS1",..: 4 2 1 6 3
 $ rm.sol       :List of 2
  ..$ onwholes: chr [1:6] "GO:0045725" "GO:0060261" "GO:0005638" "GO:2000675" ...
  ..$ sol     :List of 3
  .. ..$ optimum : num 50.8
  .. ..$ solution: Named num [1:178] 0 0 0 0 0 0 0 1 1 1 ...
  .. .. ..- attr(*, "names")= chr [1:178] "GO:0010898" "GO:0032000" "GO:0055015" "GO:006
  .. ..$ status  : int 0
 $ bp.sol       :'data.frame':        5146 obs. of  6 variables:
  ..$ Name             : chr [1:5146] "GO:0000002" "GO:0000012" "GO:0000014" "GO:0000028
  ..$ ActiveProbability: num [1:5146] 0 0.00589 0.00222 0.0023 0.00166 ...
  ..$ Count            : int [1:5146] 0 52556 19774 20474 14821 10684 44930 16304 0 0 ..
  ..$ Sample           : int [1:5146] 8919158 8919158 8919158 8919158 8919158 8919158 89
  ..$ Degree           : int [1:5146] 20 8 6 8 17 15 5 11 13 6 ...
  ..$ Response         : int [1:5146] 0 0 0 0 0 0 0 0 0 0 ...
  ..- attr(*, "samples")= num [1:11000, 1:10] 3.67e-14 2.96e-03 1.86e-05 2.70e-03 1.91e-
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:10] "AcceptanceProb" "LogLikelihood" "LogPrior" "ActiveOne" ...
 $ incidence.mat:List of 3
  ..$ I   : num [1:9397, 1:5146] 0 0 1 0 0 0 0 0 0 0 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:9397] "100" "1000" "10000" "10003" ...
  .. .. ..$ : chr [1:5146] "GO:0000002" "GO:0000012" "GO:0000014" "GO:0000028" ...
  ..$ y   : Named num [1:9397] 0 0 0 0 0 0 0 0 0 0 ...
  .. ..- attr(*, "names")= chr [1:9397] "100" "1000" "10000" "10003" ...
  ..$ edge:'data.frame':        51180 obs. of  2 variables:
  .. ..$ go_id  : chr [1:51180] "GO:0001867" "GO:0002921" "GO:0007597" "GO:0045916" ...
  .. ..$ gene_id: chr [1:51180] "2" "2" "2" "2" ...
```

3.4 A plot function

In order to visualize the active gene sets, we make a plot function `rmPlot` to display an image about the enriched gene sets and the associated genes contained by them. At the top row of the plot, it's the gene set having the highest overlap with the input gene list; then this gene set and genes in this set are removed from the system. The second row will plot the gene set

having the highest overlap with the remainder, and so on.

4 Discussion

The R package `Rolemodel` mainly implements two computational approaches in the gene-set analysis problem: MFA-ILP and MFA-MCMC. When the system in consideration is large, MFA-ILP usually has a long running time and a high requirement on the memory size, due to the complexity of integer linear programming. We intend to develop an approximate algorithm to compute a local maximum of the likelihood of role model. The local maximum should be close to the real MAP estimate, while the computation time will be much faster. At that time, a new wrapper will be added to the package to implement the approximate algorithm. On the other hand, the estimation of parameters α, γ and p can be a potential problem. We have developed some pipelines to estimate the values of α and γ; to develop an approach to estimate p without the help of MGSA can contribute to make MFA as a standalone gene-set enrichment method.

Reference

Bauer, S., Gagneur, J. and Robinson, P. N. (2010). GOing Bayesian: model-based gene set analysis of genome-scale data. *Nucleic Acids Research*, 38 (11): 3523-3532.

Wang, Z., He, Q., Larget, B. and Newton, M. (2015). A multi-functional analyzer uses parameter constaints to improve the efficiency of model-based gene-set analysis.

Andrew P. M. and others: Large-scale association analysis provides insights into the genetic architecture and pathophysiology of type 2 diabetes (2012). Nature Genetics, Volume 44-9.

```
> rmPlot(res[[4]]$I, res[[4]]$y, res[[2]]$onwholes)
```

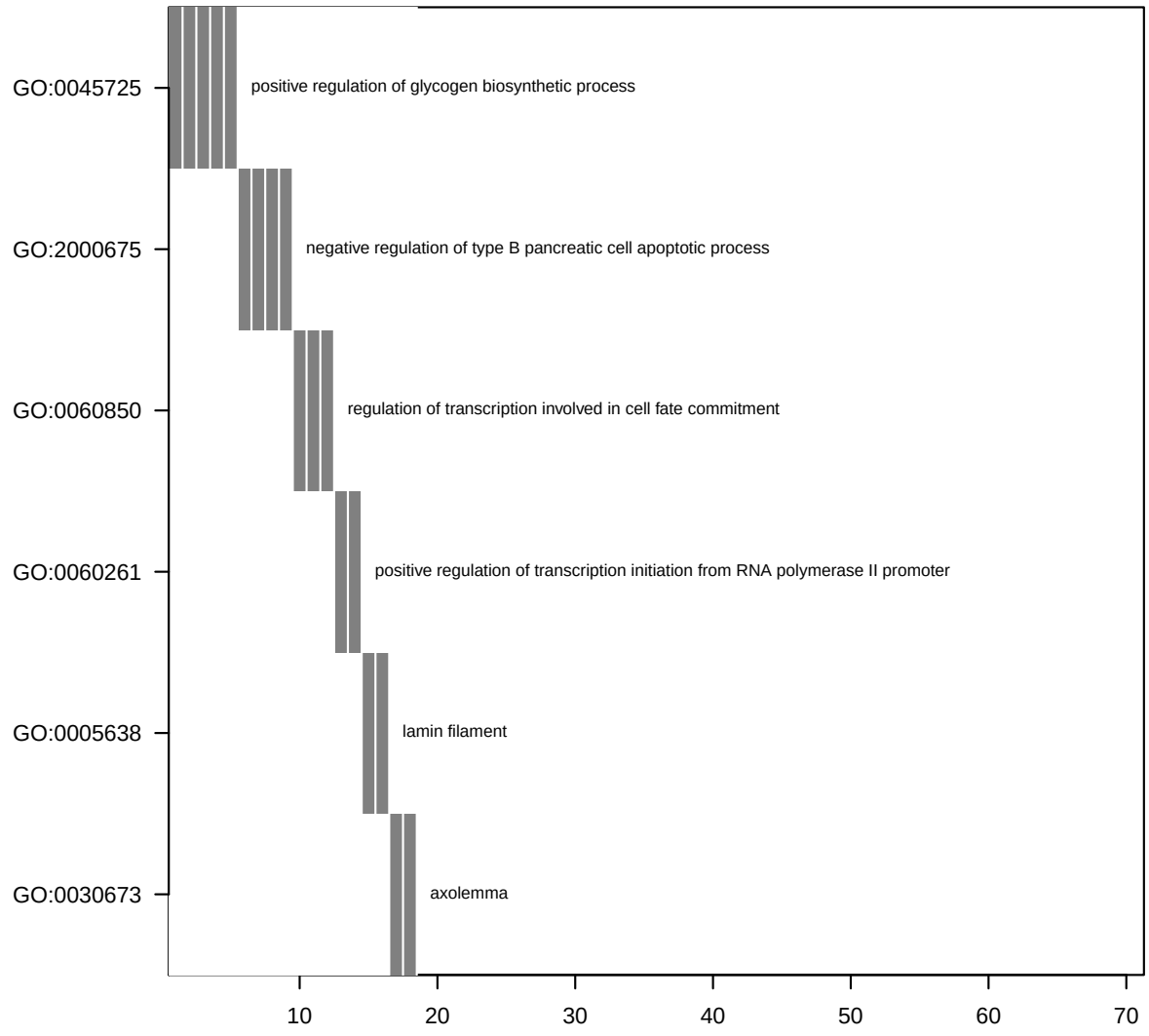

Figure 1: Display the active gene sets. Top row is the gene set having the highest overlap with the input gene list; then this gene set and genes in this set are removed from the system. The second row is the gene set having the highest overlap with the remainder, and so on.