SigMod user manual

Yuanlong LIU

December 18, 2017

Contents

1 Brief description of SigMod **2**

2 Performing SigMod analysis **2**

2.1 Load R functions/packages required by SigMod 2

2.2 Prepare a node-scored network 3

2.3 Identify gene module(s) 6

2.3.1 A new module identification strategy 6

2.3.2 Description of the `SigMod_bisection` function 7

2.3.3 A demo analysis 9

Appendices **13**

A Automatic construction of a node-scored gene network from the STRING database 13

B An example analysis of the updated SigMod using the STRING network 15

C Description of the selection path of SigMod 16

1 Brief description of SigMod

(Copied from the abstract of [Liu et al., 2017a]) Apart from single marker-based tests classically used in genome-wide association studies (GWAS), network-assisted analysis has become a promising approach to identify a set of genes associated with disease. To date, most network-assisted methods aim at finding genes connected in a background network, whatever the density or strength of their connections. This can hamper the findings as sparse connections are non-robust against noise from either the GWAS results or the network resource

We present SigMod, a novel and efficient method integrating GWAS results and gene network to identify a strongly interconnected gene module enriched in high association signals. Our method is formulated as a binary quadratic optimization problem, which can be solved exactly through graph min-cut algorithms. Compared to existing methods, SigMod has several desirable properties: (i) edge weights quantifying confidence of connections between genes are taken into account, (ii) the selection path can be computed rapidly, (iii) the identified gene module is strongly interconnected, hence includes genes of high functional relevance, and (iv) the method is robust against noise from either the GWAS results or the network resource.

The description of how to use SigMod is given in the following sections.

2 Performing SigMod analysis

2.1 Load R functions/packages required by SigMod

First, load the igraph R package. SigMod requires igraph 1.0.0 or above, which should be installed in advance

```
> require( igraph )
> if(packageVersion("igraph") < "1.0.0") {
+     stop("Need to install igrah version 1.0.0 or above")
+ }
```

Then, load SigMod functions

```
> # setwd('path/to/SigMod')
> source('../R/construct_scored_network.R')
> source('../R/convert_pvalues_2_scores.R')
> source('../R/remove_isolated_nodes.R')
> source('../R/SigMod_core.R')
> source('../R/selection_path.R')
> source('../R/selection_path_fast.R')
> source('../R/SigMod_bisection.R')
```

2.2 Prepare a node-scored network

SigMod works on a node-scored network, in which the nodes represent proteins/genes, and edges represent their interactions (or other types of relationships, depending on the analysis background). We implemented the `construct_scored_net` function to build a node-scored gene network (we also implemented the `construct_string_net` function to provide automatic construction of a node-scored network via online retrieving gene/protein interaction information from the STRING database [Szklarczyk et al., 2017]. Details are given in section A).

The arguments of `construct_scored_net` include:

- `network_data`: A data frame that contains n rows and m columns. Two of its columns represent the interactions, additional columns represents the attributes of interactions such as interaction weight, interaction detection methods, etc.
- `interaction_indices`: A two-element numeric vector specifying the index of columns which represent the interactions. For example, if the interactions are represented by column 3 and 4, then `interaction_indices = c(3,4)`
- `weight_index`: The index of column which specifies the weight of the interactions. For example, if column 5 specifies the interaction weight, then set `weight_index = 5`. If there is no weight informaiton in the `network_data`, or if the weight information exists but the user does not want to use this informaiton in the analysis, set `weight_index = NULL`
- `gene_scores`: A data frame specifying the genes and their associated scores. This data frame should include a “gene” column and a “score” column. Extra columns can exist but will be ignored

In the following we give an example of using the `construct_scored_net` function to construct a node-scored gene network. We downloaded the protein-protein interaction data (saved as `"../data/network/PINA_net.tab"`) from the PINA platform (http://omics.bjcancer.org/pina/) (Go to the website ⇒ Home ⇒ Statistics & Download ⇒ Homo sapiens (Last update: May 21, 2014) ⇒ MITAB format). The gene score data (saved as `"../data/gene_p_score/gene_scores.tab"`) were the z-scores converted from gene association p-values as descrbed in [Liu et al., 2017b] (using the META1 outcomes).

To use the `construct_scored_net` function to build a node-scored gene network, first read the soucre gene score data into R as a data frame:

```
> gene_scores_file = '../data/gene_p_score/gene_scores.tab'
> gene_scores = read.table(gene_scores_file, header = TRUE)
> ## Convert all gene names to upper case to prevent case mismatch:
> gene_scores$gene = toupper(gene_scores$gene)
> head(gene_scores)
```

```
       gene    score
1     GSDMB 4.396983
2    ORMDL3 4.271456
3 LOC728129 4.099674
4     ZPBP2 4.081476
5     GSDMA 4.041130
6      PNMT 4.006452
```

Similarly, read the source network data into R as a data frame:

```
> network_data_file = '../data/network/PINA_net.tab'
> network_data = read.table(network_data_file,
+                            header = TRUE, fill=TRUE,
+                            stringsAsFactors = FALSE,
+                            sep='\t', quote='"')
> ## Convert all gene names to upper case to prevent case mismatch:
> network_data[,3] = toupper(network_data[,3])
> network_data[,4] = toupper(network_data[,4])
```

Then, use the `construct_scored_net` function to build a node-scored gene network. The two columns that represent the interation list are column 3 and 4, therefore specify `interaction_indices = c(3,4)`. As no interaction weight is provided by the PINA network, specify `weight_index = NULL`

```
> net = construct_scored_net(network_data, interaction_indices=c(3,4),
+                            weight_index=NULL, gene_scores )
> ## an overview of the constructed net:
> net
```

```
IGRAPH UNW- 12110 119620 --
+ attr: name (v/c), weight (v/n), weight (e/n)
+ edges (vertex names):
 [1] ELAVL1--KHSRP     ELAVL1--ARID1A    ELAVL1--RALA
 [4] ELAVL1--RPA3      ELAVL1--PARK2     ELAVL1--RNPS1
 [7] ELAVL1--UBE4B     ELAVL1--FN1       ELAVL1--UBE2C
[10] ELAVL1--GABARAPL2 ELAVL1--ZNF638    ELAVL1--HNRNPA1
[13] ELAVL1--RAB7A     ELAVL1--RHOQ      ELAVL1--CTTN
[16] ELAVL1--VCAM1     ELAVL1--CUL1      ELAVL1--CANX
[19] ELAVL1--ETS1      ELAVL1--NCL       ELAVL1--CHRM2
[22] ELAVL1--YWHAB     ELAVL1--CUL3      ELAVL1--RAN
+ ... omitted several edges
```

Note, in many studies, a gene p-value data instead of a gene score data is provided. We implemented the `p2score` function to convert gene p-values to gene scores. The conversion is based on the equation $score = \Phi^{-1}(1-p\_value)$, where Φ^{-1} is the inverse of the Cumulative Distribution Function of the standard normal distribution $N(0, 1)$. The following gives an example of how to convert gene p-values into scores:

```
> gene_ps_file = '../data/gene_p_score/gene_ps.tab'
> gene_ps = read.table(gene_ps_file, header = TRUE)
> head(gene_ps)
```

```
       gene            p
1     GSDMB 5.488287e-06
2    ORMDL3 9.710045e-06
3 LOC728129 2.068662e-05
4     ZPBP2 2.237532e-05
5     GSDMA 2.659708e-05
6      PNMT 3.081884e-05
```

```
> ## convert gene p-values to scores using the p2score function:
> gene_scores = p2score( gene_ps )
> head(gene_scores)
```

```
      gene            p    score
1    GSDMB 5.488287e-06 4.396983
2   ORMDL3 9.710045e-06 4.271456
3 LOC728129 2.068662e-05 4.099674
4    ZPBP2 2.237532e-05 4.081476
5    GSDMA 2.659708e-05 4.041130
6     PNMT 3.081884e-05 4.006452
```

Some notes about constructing a node-scored gene network:

1^o SigMod only handles undirected networks. In the case there is direction over the interaction in the source network data, this information will be ignored

2^o SigMod only handles simple networks. A simple network is a network which does not contain loops and multiple edges. We used the `simplify` funciton to convert a network into a simple one in the `construct_scored_net` function. We also suggest the user to convert their source network into a simple network before calling the `construct_scored_net` function

3^o Many network databases describe protein-protein interactions instead of gene interactions, such as the PINA database. In such case the protein identifier should be mapped exactly to the gene identifiers as is used in the gene score file

2.3 Identify gene module(s)

2.3.1 A new module identification strategy

The module selection based on the SigMod formulation depends on the value of the λ and η parameter. To identify a disease-associated module, we described a hierarchical approach in the SigMod article [Liu et al., 2017a]. This approach included a major step as the computation of the selection path for k values of λ. These λ values are equally spaced in a range $[\lambda_{min}, \lambda_{max}]$ that can be specified by the user. One limitation of this approach is that some (or many) of these λ values can have the same (or slightly different) selection path, thus leading to redundant computation. Also, different values of λ_{min}, λ_{max}, and especially k can result in different outcomes.

Here, we implemented an alternative approach that may avoid these problems.

This approach identifies a disease-associated module via finding the smallest λ value (denote as λ_{opt}) that has the following property:

> In the part of its selection path $\mathcal{P} = \langle \varnothing, S_1, ..., S_k \rangle$, there is at least one size jump greater than $maxjump$ (Definition of selection path and size jump please refer to Appendices C). Here S_k is the largest selection whose number of genes is $\leq nmax$ ($nmax$ is a user defined parameter that represents the maximum amount of genes to be considered).

Using this criterion, λ and η become intermediate parameters that do not need to be specified explicitly. Instead, the module selection will be directly determined by parameters $nmax$ and $maxjump$, which have more intuitive meaning and are more manipulable than λ and η.

To find λ_{opt}, we implemented a bisection (binary) search algorithm, similar to that described in https://en.wikipedia.org/wiki/Binary_search_algorithm. Though λ_{opt} can lie anywhere within $[0, +\infty]$, we provide a choice to set the range $[\lambda_{min}, \lambda_{max}]$ for searching as properly small so that to achieve reducing of computing time [1]. The flowchart describing this algorithm is given in Figure 1. We implemented this strategy as the `SigMod_bisection` funciton that will be described in the next section

2.3.2 Description of the `SigMod_bisection` function

Parameters of the `SigMod_bisection` function:

- `lambda_min`: The lower bound of the interaval in which to search for λ_{opt}. We set `lambda_min = 0` as default
- `lambda_min`: The upper bound of the interaval in which to search for λ_{opt}. We set `lambda_max = 1` as default
- `nmax`: The maximum number of genes to be considered for each lambda value
- `maxjump`: A threhold that indicates whether a lambda value is big enough to lead selection of strongly interaconnected genes. We

[1] A warning informaiton will be given to the user to increase the range if λ_{opt} is not found in this range

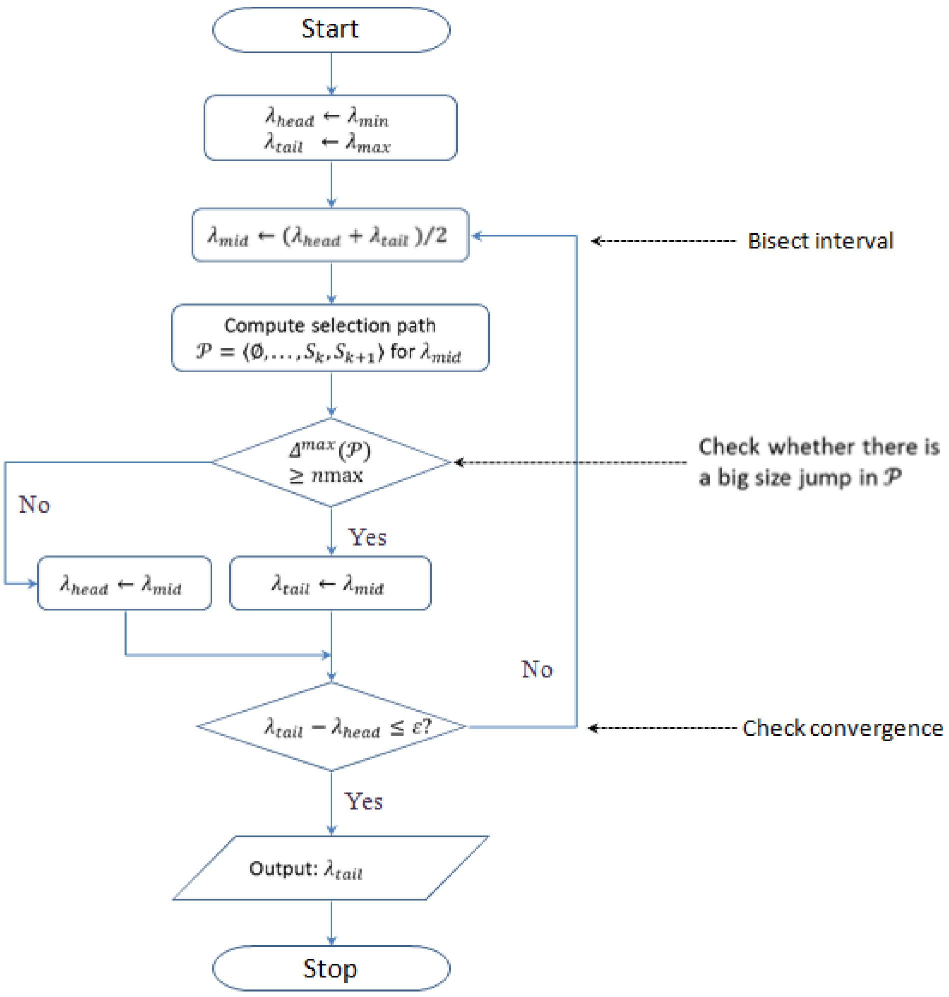
Start
$\lambda_{head} \leftarrow \lambda_{min}$
$\lambda_{tail} \leftarrow \lambda_{max}$
$\lambda_{mid} \leftarrow (\lambda_{head} + \lambda_{tail})/2$
Bisect interval
Compute selection path
$\mathcal{P} = \langle \emptyset, \ldots, S_k, S_{k+1} \rangle$ for λ_{mid}
$\Delta^{max}(\mathcal{P}) \geq n\max$
Check whether there is a big size jump in $\mathcal{P}$
No
Yes
$\lambda_{head} \leftarrow \lambda_{mid}$
$\lambda_{tail} \leftarrow \lambda_{mid}$
No
$\lambda_{tail} - \lambda_{head} \leq \varepsilon$?
Check convergence
Yes
Output: λ_{tail}
Stop

Figure 1: Flowchart of the binary search algorithm to find λ_{opt}

set `maxjump = 10` by default. Increasing this threhold may lead to seleting a module that has more strong interconntion, but the overal module score can be smaller

Output of the function:

The output is a list that contains various module selection information, including:

- `opt_module`: a list that contains two modules. The first module is the optimal module that is selected (`opt_module[[selected]]`), while the second module is the subsequent module in the same selection path (`opt_module[[selected_next]]`). Though the user can choose directly `opt_module[[selected]]` as the final selected module, `opt_module[[selected_next]]` can include additional genes and deserves to have a look
- `lambdas`: the λ values that have been computed
- `paths`: the selection path of each computed λ value
- `opt_index`: the index of λ_{opt} in `lambdas`

2.3.3 A demo analysis

The following code gives an example of identifying module(s) using SigMod in the node-scored network constructed in Section 2.2:

```
> res_info = SigMod_bisection( net,
+                              lambda_min=0, lambda_max=1,
+                              nmax=300, maxjump=10)
```

```
> names( res_info )
```

```
[1] "opt_module" "lambdas"    "paths"      "opt_index"
```

To see the information of optimal module(s):

```
> res_info$opt_module

$selected
IGRAPH UNW- 160 332 --
+ attr: name (v/c), weight (v/n), weight (e/n)
+ edges (vertex names):
 [1] UBD--RPS2      UBD--TUBB      UBD--FBL
 [4] UBD--SFPQ      UBD--HSPA1A    UBD--RPS20
 [7] UBD--TRIM27    UBD--HNRNPA2B1 UBD--SYNCRIP
[10] UBD--EPB41L2   UBD--PSMD4     UBD--KIAA0664
[13] UBD--FLOT1     UBD--IKBKAP    UBD--YBX1
[16] UBD--VARS      UBD--USO1      UBD--ARF1
[19] UBD--TRIM56    UBD--PSMD13    UBD--PRPF19
[22] UBD--TRIM10    UBD--HIST1H2BL UBD--SH3GLB1
+ ... omitted several edges

$selected_next
IGRAPH UNW- 439 6180 --
+ attr: name (v/c), weight (v/n), weight (e/n)
+ edges (vertex names):
 [1] ELAVL1--RPA3    ELAVL1--PARK2   ELAVL1--FN1
 [4] ELAVL1--HNRNPA1 ELAVL1--VCAM1   ELAVL1--CUL1
 [7] ELAVL1--NCL     ELAVL1--CUL3    ELAVL1--YWHAQ
[10] ELAVL1--ACTB    ELAVL1--TARDBP  ELAVL1--EED
[13] ELAVL1--TP53RK  ELAVL1--TUBB    ELAVL1--CDK9
[16] ELAVL1--SUMO2   ELAVL1--SFPQ    ELAVL1--SUZ12
[19] ELAVL1--SUMO1   ELAVL1--CAND1   ELAVL1--HSPA1A
[22] ELAVL1--GRB2    ELAVL1--PPP2R1A ELAVL1--VCP
+ ... omitted several edges
```

To obtain the selected genes, use the following code:

```
> selected_module = res_info$opt_module[['selected']]
> selected_genes = V(selected_module)$name
> ## order the genes by gene score:
> selected_genes = selected_genes[order( V(selected_module)$weight,
+                                        decreasing=TRUE ) ]
> head(selected_genes)

[1] "GSDMB"  "ORMDL3" "GSDMA"  "PSMD3"  "MICA"   "ERBB2"
```

To plot the selected module, use the following code:

```
> set.seed(1)
> par(mar=c(0,0,0,0))
> plot( selected_module, vertex.size = 5, vertex.label=NA )
```

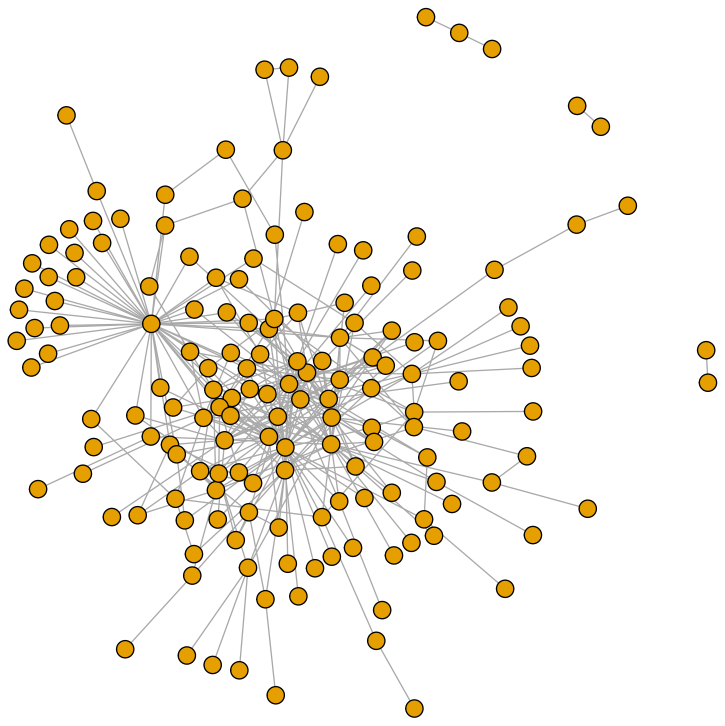

Figure 2: Plot of the selected module. Gene names are not shown

```
> set.seed(1)
> par(mar=c(0,0,0,0))
> plot( selected_module, vertex.size = 5, vertex.label.cex=0.5 )
```

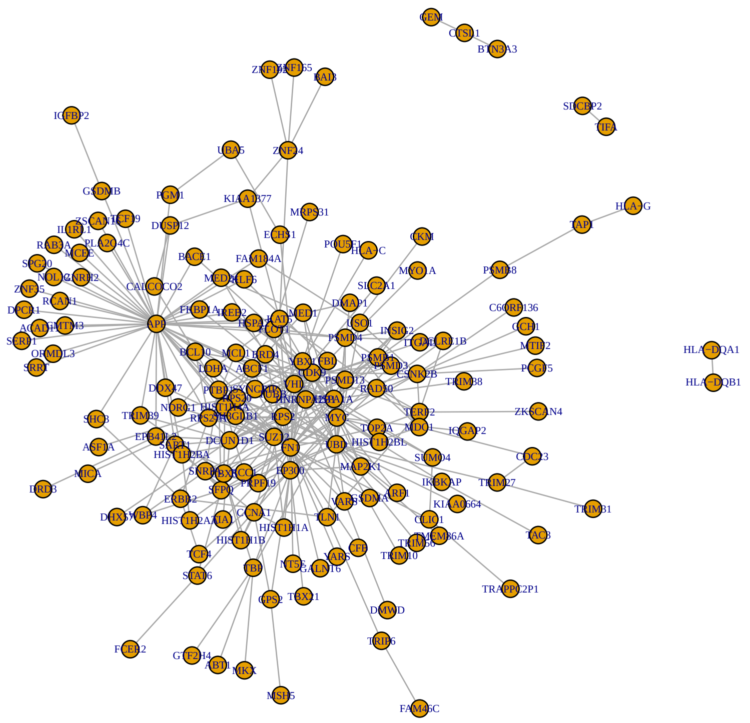

Figure 3: Plot of the selected module. Gene names are shown

Appendices

A Automatic construction of a node-scored gene network from the STRING database

Besides using the construct_scored_net function as described in section 2.2, we also implemented the construct_string_net function to provide automatic construction of a node-scored network via online retrieving gene/protein interaction information from the STRING database [Szklarczyk et al., 2017]. STRING is a comprehensive database which describes various types of interaction information between genes or their coding products (proteins). The full description of the STRING database can be found in https://string-db.org/.

To use the construct_string_net function, the STRINGdb R package need be installed. Details of installing and using STRINGdb can be found on the Bioconductor website https://bioconductor.org/packages/release/bioc/html/STRINGdb.html.

The arguments of construct_string_net include:

- gene_ps A dataframe specifying the genes and their associated p-values. This data frame should include a "gene" column and a "p" column. Extra columns can exist but will be ignored
- gene_scores A dataframe specifying the genes and their associated scores. This data frame should include a "gene" column and a "score" column. Extra columns can exist but will be ignored. Either a gene_scores or a gene_ps is should be provided to run the construct_string_net function. If gene_ps is provided, the p-values will be automatically converted to scores by calling the p2score function
- evidence2exclude The interaction evidence to be excluded for constructing the network for your analysis. Way of assignment: evidence2exclude = c("evidence_1", "evidence_2", ...); or evidence2exclude = NULL if no evidence wants to be excluded.

  The STRING database includes various types of gene/protein interaction evidence, including:

  "neighborhood", "neighborhood_transferred", "fusion",

  "cooccurence", "homology", "coexpression", "database", "coexpression_transferred", "experiments_transferred", "experiments", "database_transferred", "textmining", "textmining_transferred"

The exact name and decription of each interaction evidence please refer to the STRINGdb package. Some of these evidences may not be wanted for your network analysis thus can be excluded. We set evidence2exclude = c("textmining_transferred", "textmining") as default, thus to exclude the interaction evidence obtained using text-mining methods. When excluding some evidences, the combined confidence score will be recomputed based on the remained evidences, according to the formula $S_{combined} = \prod_i (1 - S_i)$ as was described in the STRING article [von Mering et al., 2005], where S_i is the score of an individual evidence

- confidence A combined interaction between two genes is only retained if the combined confidence score is above a confidence threshold, which can be set as any value between 0 and 1. We set confidence = 0.7 as default. This results in a high-confidence network according to the categorization described in the STRING article

- genes2exclude The genes to be excluded for performing the network analysis. Removing the genes that are (likely to be) irrelevant to the trait of study can reduce false positives and decrease computational complexity. genes2exclude can be assigned with three types of value:

  – genes2exclude = "OR_genes"(set as default): genes of the Olfactory Receptors (OR) gene family are excluded. Description of the OR gene family can be found in: "../data/OR_genes/Olfactory_receptors_readme.txt". The OR genes form a huge hairball structure (a group of genes having dense interconnections) in the STRING network. It can hamper the power of SigMod. We suggest to remove these genes from the STRING network as long as they are (likely to be) irrelevant to the phenotype under study

  – genes2exclude = "./xx.xx": genes included in the "./xx.xx" file will be excluded. An example of ./xx.xx file is given in "../data/genes2exclude/gene_to_exclude.tab"

  – genes2exclude = NULL: no gene will be excluded

The following code gives an example to construct a node-scored STRING network when a gene p-values data is provided:

```
> require(STRINGdb)
> gene_ps_file = '../data/gene_p_score/gene_ps.tab'
> gene_ps = read.table(gene_ps_file, header = TRUE)
> ## construct the string-based scored-network:
> string_net = construct_string_net(
+   gene_ps = gene_ps,
+   evidence2exclude = c('textmining_transferred', 'textmining'),
+   genes2exclude = "OR_genes",
+   confidence=0.7 )
> ## an overview of the constructed string_net:
> string_net
```

The following code gives an example to construct a node-scored STRING network when a gene score data is provided:

```
> require(STRINGdb)
> gene_scores_file = '../data/gene_p_score/gene_scores.tab'
> gene_scores = read.table(gene_scores_file, header = TRUE)
> string_net = construct_string_net(
+   gene_scores = gene_scores,
+   evidence2exclude = c('textmining_transferred', 'textmining'),
+   genes2exclude = "OR_genes",
+   confidence=0.7 )
> ## an overview of the constructed string_net:
> string_net
```

B An example analysis of the updated SigMod using the STRING network

```
> require(STRINGdb)
> gene_ps_file = '../data/gene_p_score/gene_ps.tab'
> gene_ps = read.table(gene_ps_file, header = TRUE)
> ## construct the string-based scored-network:
> string_net = construct_string_net(
+   gene_ps = gene_ps,
+   evidence2exclude = c('textmining_transferred', 'textmining'),
+   genes2exclude = "OR_genes",
+   confidence=0.7 )
```

```
> res_info = SigMod_bisection( net=string_net,
+                              lambda_min=0, lambda_max=1,
+                              nmax=300, maxjump=10)
```

To see the information of optimal module(s):

```
> res_info$opt_module
```

To obtain the selected genes, use the following code:

```
> selected_module = res_info$opt_module[['selected']]
> selected_genes = V(selected_module)$name
> ## order the genes by gene score:
> selected_genes = selected_genes[order( V(selected_module)$weight,
+                                        decreasing=TRUE ) ]
> head(selected_genes)
```

C Description of the selection path of SigMod

SigMod is based on solving a maximization problem:

$$\arg\max_{\boldsymbol{u}} f(\boldsymbol{u})\boldsymbol{z}^\intercal \boldsymbol{u} + \lambda \boldsymbol{u}^\intercal A \boldsymbol{u} - \eta \|\boldsymbol{u}\|_0, \tag{1}$$

where $\boldsymbol{z}$ are gene scores, A is the adjacency matrix of the gene network, $\boldsymbol{u}$ are indicator variables that takes a value of either 0 or 1. λ is the interconnection tuning parameter and η is the sparsity tuning parameter.

For a given λ value, denote the gene selection with sparsity parameter η as $S(\eta)$. We define the selection path $\mathcal{P}$ of a sparsity range $[\eta_{\min}, \eta_{\max}]$ as the sequence of unique gene selections by moving η from $\eta_{\max}$ to $\eta_{\min}$, i.e. $\mathcal{P} = \langle S_1, ..., S_m \rangle$, where $S_1, ..., S_m$ are the unique selections (see Figure S1 as an example). The selection path for the whole sparsity range $[0, +\infty]$ is therefore $\mathcal{P} = \langle \varnothing, S_1, ..., S_m, S_{m+1}, ..., S_{max} \rangle$, where $\varnothing$ is the selection of no genes, S_{max} is the selection that has maximal possible amount of genes.

The size jump is defined as the number of differently selected genes between two consecutive selections S_k and S_{k+1}.

References

Y. Liu, M. Brossard, D. Roqueiro, P. Margaritte-Jeannin, C. Sarnowski, E. Bouzigon, and F. Demenais. Sigmod: an exact and efficient method to identify a strongly interconnected disease-associated module in a gene network. *Bioinformatics*, 33(10):1536–1544, 2017a. doi: 10.1093/bioinformatics/btx004. URL +http://dx.doi.org/10.1093/bioinformatics/btx004.

Y. Liu, M. Brossard, C. Sarnowski, A. Vaysse, M. Moffatt, P. Margaritte-Jeannin, F. Llinares-López, M. Dizier, M. Lathrop, W. Cookson, et al. Network-assisted analysis of gwas data identifies a functionally-relevant gene module for childhood-onset asthma. *Scientific Reports*, 7(1):938, 2017b.

D. Szklarczyk, J. H. Morris, H. Cook, M. Kuhn, S. Wyder, M. Simonovic, A. Santos, N. T. Doncheva, A. Roth, P. Bork, L. J. Jensen, and C. von?Mering. The string database in 2017: quality-controlled proteiníc protein association networks, made broadly accessible. *Nucleic Acids Research*, 45(D1):D362–D368, 2017. doi: 10.1093/nar/gkw937. URL +http://dx.doi.org/10.1093/nar/gkw937.

C. von Mering, L. J. Jensen, B. Snel, S. D. Hooper, M. Krupp, M. Foglierini, N. Jouffre, M. A. Huynen, and P. Bork. String: known and predicted proteiníc protein associations, integrated and transferred across organisms. *Nucleic Acids Research*, 33:D433–D437, 2005. doi: 10.1093/nar/gki005. URL +http://dx.doi.org/10.1093/nar/gki005.