SigMod v2 user manual

Yuanlong LIU

January 7, 2018

Contents

1 Introduction of the SigMod method (v2)

1.1 Mathematical formulation

SigMod identifies a module that is enriched in high score genes and tend to be strongly interconnected from a node-scored network based on solving a maximization problem:

$$\arg\max_{u_1,u_2,\cdots,u_n} f(u_1,u_2,\cdots,u_n,\lambda,\eta), \qquad (1)$$

where the objective function f is defined as

$$f(u_1,u_2,\cdots,u_n,\lambda,\eta) = \underbrace{\sum_i z_i u_i}_{Association} + \underbrace{\lambda \sum_{i\neq j} A_{ij} u_i u_j}_{Interconnectivity} - \underbrace{\eta \sum_i I(u_i = 1)}_{Sparsity}.$$

In f, z_i is the score of the i th gene. A_{ij} is the interaction weight between the i th and j th gene (A_{ij} = 0 if there is no interaction). u_i is an indicator variable indicating whether a gene is selected ($u_i = 1$) or not ($u_i = 0$). λ is the interconnection tuning parameter. Bigger λ values will give more focus on the interconnectivity, thus lead to select genes that have stronger interconnection. η is the sparsity tuning parameter. A bigger η value will lead to select less genes.

For a fixed λ and η value, solving the maximization problem defined by Equation (1) will determine the value of each u_i, thus determine whether a gene is selected or not. We have shown that the maximization problem can be solved exactly using a graph-cut algorithm [Liu et al., 2017].

As different values of λ and η will lead to select different modules, they need to be set properly in order to select a gene module that is enriched in high score genes and tend have strong interconnection. We have proposed an hierarchical strategy for parameter setting in [Liu et al., 2017]. Here, we present an updated parameter setting strategy that have several advantages over the original one. To better understand the strategy, we first recall the definition of a "selection path" and the "size jump" in the selection path, which are useful terminologies already defined in our paper.

1.2 Recall the definition of selection path and size jump

1.2.1 The selection path

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 obtained by moving η from $\eta_{\max}$ to $\eta_{\min}$, i.e., $\mathcal{P} = \langle S_1, ..., S_m \rangle$ if $S_1, ..., S_m$ are these unique selections. The whole selection path, defined on the sparsity range $[0, +\infty]$, is therefore

$\mathcal{P}^{whole} = \langle \varnothing, S_1, ..., S_k, S_{k+1}, ..., S_{max} \rangle$, where $\varnothing$ is the empty selection of no genes (corresponding to $\eta = +\infty$); S_{max} is the selection that has maximal possible amount of genes (corresponding to $\eta = 0$). Figure 1 gives an example of a selection path.

We have developed a path algorithm that can compute the selection path over any sparsity range $[\eta_{\min}, \eta_{\max}]$ [Liu et al., 2017].

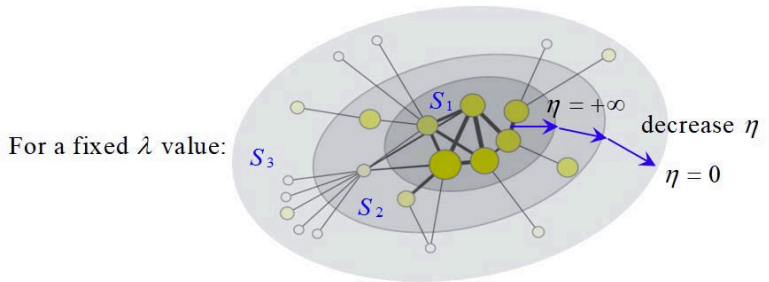

Figure 1: An example of selection path. In this example, the scored network contains 22 genes. The node size is proportional to the gene score. For a fixed λ value, we consider the sparsity range $[\eta_{\min}, \eta_{\max}] = [0, +\infty]$. There are four possible selections by moving η from $+\infty$ to 0: S_0 (an empty selection of 0 genes), S_1, S_2, S_3 (represented by each contour). These selections are nested ($S_0 \subseteq S_1 \subseteq S_2 \subseteq S_3$). The numbers of genes in these selections are $\#S_0 = 0$, $\#S_1 = 6$, $\#S_2 = 10$, and $\#S_3 = 22$ respectively. The size jump, defined as $\Delta_i = \#S_i - \#S_{i-1}$, are $\Delta_1 = 6 - 0 = 6$, $\Delta_2 = 10 - 6 = 4$, $\Delta_3 = 22 - 10 = 12$. The maximum size jump in this selection path is $\Delta_3 = 12$.

1.2.2 The size jump

It is of note that in a given selection path, the different selections are nested (i.e., $S_{k-1} \subseteq S_k$), thus there are at most n different selections (n is the number of genes in the scored-network). For proof, please refer to [Liu et al., 2017].

Besides, the amount of different genes between two consecutive selections $\#S_k$ and $\#S_{k-1}$ is not necessarily 1 (as illustrated in Figure 2). We call this amount as the size jump, and denote it as Δ_k: $\Delta_k = \#S_k - \#S_{k-1}$. We define the maximum size jump Δ_{max} in a selection path as the maximum value of all size jumps.

One interesting property of the size jump is that, for a bigger λ value, there is usually bigger size jumps in its selection path, as revealed in Figure 2. This is because bigger λ values will give more importance to the interconnection, thus some strongly interconnected genes are non-separable and are selected together. Therefore, we can determine whether a λ value is big enough, by checking whether there is a big size jump in its selection path. This property has been used in our previous strategy to select a strongly interconnected module, and will be used again in our new strategy, as will be introduced in the following part.

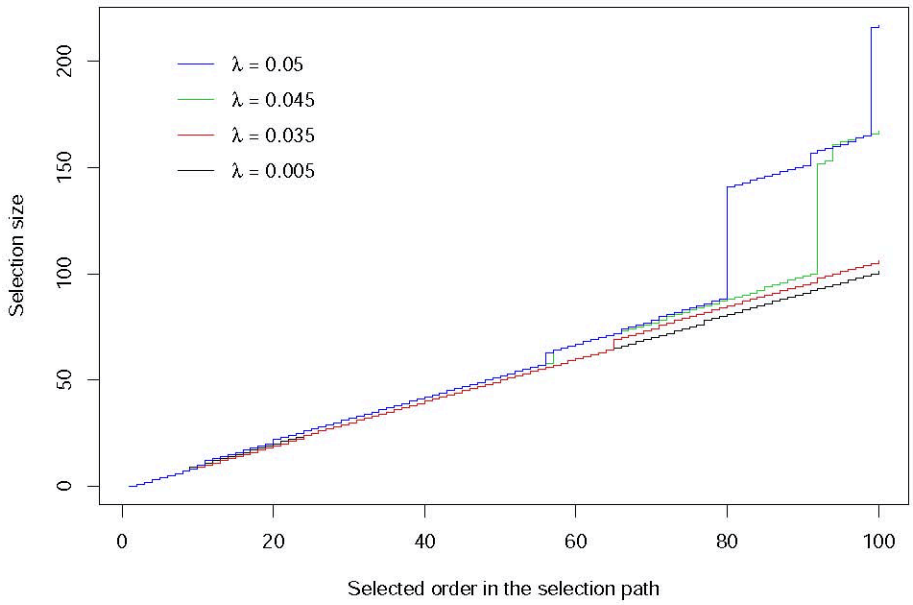

Figure 2: The number of genes in each selection (S) in a selection path ($\mathcal{P}$). Four paths corresponding to four λ values are shown. The x-axis represents the order of each selection in the path. The y-axis represents the number of genes in each selection. In a given path, a big size jump between two consecutive selections indicates some strongly interconnected genes are non-separable and are selected together. For larger λ values, the jumps are usually bigger, such as for $\lambda = 0.05$ (blue line).

1.3 An updated parameter setting / module identification strategy

The module selection depends on the value of the λ and η. To identify a disease-associated module, we previously described a hierarchical strategy. This strategy 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 strategy 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 propose a new strategy that may overcome some of these limitations. This strategy identifies a disease-associated module via finding the smallest λ value (denote as λ_{opt}) that has the following property:

The maximum size jump in its selection path $\mathcal{P}_k$ is greater than or equal to a user defined threshold *maxjump* , where $\mathcal{P}_k = \langle \varnothing, S_1, ..., S_k \rangle$ is the part of the whole selection path $\mathcal{P}^{whole} = \langle \varnothing, S_1, ..., S_k, S_{k+1}, ..., S_{max} \rangle$ such that S_k is the smallest selection satisfying $\#S_k \geq nmax$. *maxjump* and *nmax* are thresholds that will be specified by the user.

This criterion includes two aspects. First, we consider the subpath $\mathcal{P}_k = \langle \varnothing, S_1, ..., S_k \rangle$ so that to select approximately *nmax* genes. Second, we want λ to be big enough so that to encourage selecting strongly connected genes. To determine whether a λ value is big enough, we check whether the maximum size jump in $\langle \varnothing, S_1, ..., S_k \rangle$ is greater than a threshold *maxjump* . The idea behind was illustrated in section 1.3.

Using this criterion, λ and η become intermediate parameters that do not need to be specified explicitly. Instead, the module selection will be determined by two alternative 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 search algorithm, similar to that described in https://en.wikipedia.org/wiki/Binary_search_algorithm, which has the goal of finding the position of a target value within a sorted array. Though λ_{opt} can lie anywhere in $[0, +\infty]$, we set the searching interval as [0, 1] by default. The flowchart describing our algorithm is given in Figure 3. One desirable property of the bisection search algorithm is that, for each iteration, the length of the interval to be searched will be shrunk to half of the length of interval in the previous iteration, thus can reduce considerably the number of λ values to be computed.

Once λ_{opt} is found (therefore $\langle \varnothing, S_1, ..., S_k \rangle$), if $\#S_k = nmax$, we choose S_k but remove the genes that do not have any interaction with other genes in S_k to get the final module; if $\#S_k > nmax$, we choose S_{k-1} but remove the genes that do not have any interaction with other genes in S_{k-1} to get the final module.

We implemented this strategy in the SigMod_bisection R function that will be described in section 2.

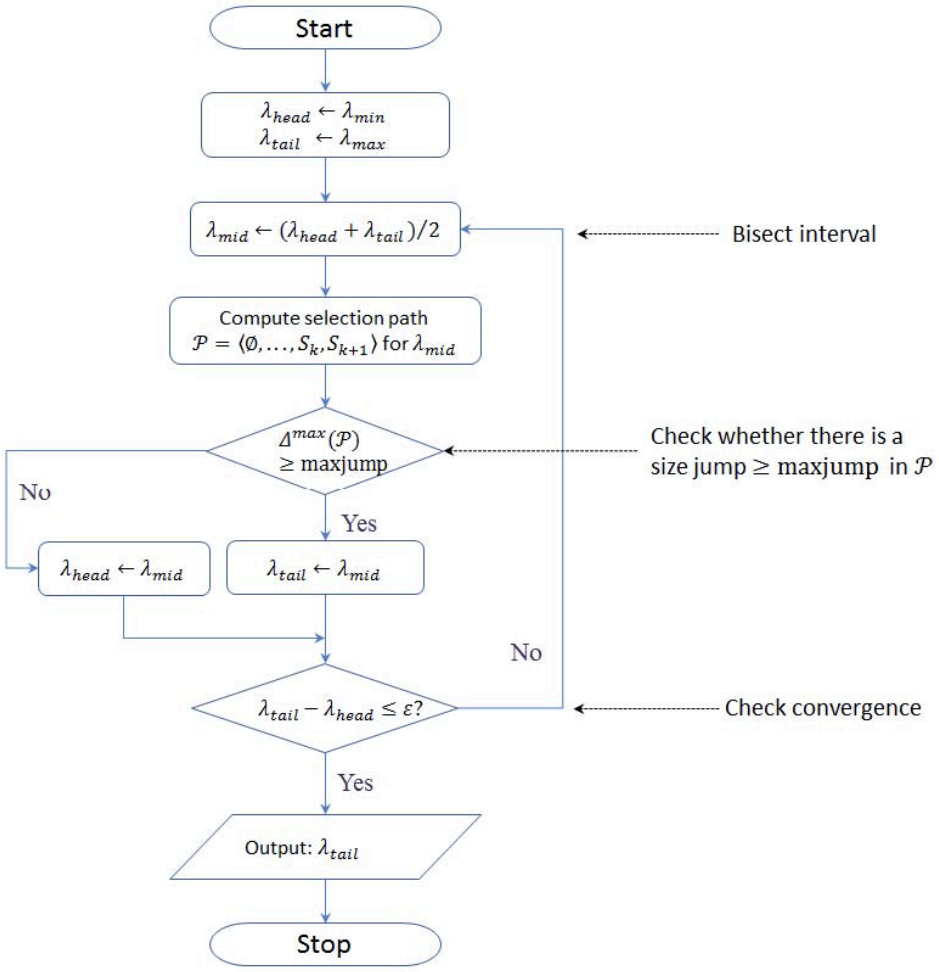

Figure 3: Flowchart of the bisection search algorithm to find λ_{opt}. The interval to search for λ_{opt} is initialized as $[\lambda_{min}, \lambda_{max}]$ ($[0,1]$ by default). The bisection search procedure terminates if the length of the searching interval becomes smaller than $\epsilon = 10^{-5}$

2 Implementation and usage

We have implemented various R functions for performing SigMod analysis. These include functions to build a scored network, and functions to identify a disease-associated module using the strategy introduced in section 1.3. These functions are included in the SigMod_v2\R folder. In the following we introduce how to conduct a network analysis using these functions.

2.1 Load the required R functions/packages

First, load the igraph R package. SigMod requires igraph 1.0.0 or above, which should be installed ahead of analysis:

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

Then, load all R functions in the SigMod_v2\R folder using the following code (the install.packages approach is not applicable currently)

```
R_dir = '../R' ## the path to the SigMod_v2\R folder
## load all R functions in this folder
file.sources = list.files(R_dir, pattern='*.R$',
                          full.names=TRUE, ignore.case=TRUE)
tmp=sapply(file.sources, source, .GlobalEnv)
```

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 for general cases, and the construct_string_net function for automatic construction of a node-scored network via on line retrieving gene/protein interaction information from the STRING database [Szklarczyk et al., 2017].

2.2.1 Description of the construct_scored_net function

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, optional additional columns represent the attributes of interactions such as interaction weight, interaction detection methods, etc. The following example shows how network_data looks like. In this example, the first two columns represent the interactions, other columns represent attributes of the interactions:

```
> network_data
  Inteactor_A Inteactor_B Interaction_attri_1 Interaction_attri_2 Interaction_attri_n
1      gene_1      gene_3                 ...                 ...                 ...
2      gene_2      gene_4                 ...                 ...                 ...
3      gene_3      gene_5                 ...                 ...                 ...
4      gene_4      gene_6                 ...                 ...                 ...
5         ...         ...                 ...                 ...                 ...
```

- interaction_indices: A two-element numeric vector specifying the index of columns which represent the interactions. In the above example the interactions are represented by column 1 and 2, thus assign interaction_indices = c(1,2)

- weight_index: The index of column which specifies the weight of the interactions. If column *k* specifies the interaction weight, then weight_index = *k*. If there is no weight information in the network_data, or if the weight information exists but the user does not want to use this information in the analysis, set weight_index = NULL. When interaction weights are included in the analysis, no missing value is allowed in the weight values

- gene_ps: A data frame specifying the genes and their associated p-values. This data frame should have a “gene” column and a “p” column. Extra columns can exist but will be ignored. The following example shows how gene_ps looks like:

```
> 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
```

- 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. The following example shows how gene_scores looks like:

```
> 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
```

To run the construct_scored_net function, either a gene_scores or a gene_ps should be provided. If gene_ps is provided, the p-values will be automatically converted to scores by calling the p2score function. This 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 $N(0,1)$

- 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 two types of value:

  1) genes2exclude = NULL: no gene will be excluded (set as default)

  2) genes2exclude = your_genes_to_exclude: genes included in the your_genes_to_exclude will be excluded, where your_genes_to_exclude is a vector of genes. The following example shows how your_genes_to_exclude looks like:

```
> your_genes_to_exclude
[1] "gene_1" "gene_2" "..."    "gene_n"
```

Return of this function:

The return of this function is a node-scored network object in the igraph network format

Example usage:

```
############## Read network data ##############
## For example, your network data has a tab delimited text format
network_data_file = 'your_network_data.tab'

## read network_data_file into R as a data frame
network_data = read.table(network_data_file,
                          header = TRUE, stringsAsFactors = FALSE,
                          sep='\t', quote='"')

############## Read gene p-values data ##############
gene_ps_file = 'your_gene_p_values_data.tab'
gene_ps = read.table(gene_ps_file, stringsAsFactors = FALSE, header =
TRUE)

############## Construct a scored network ##############
## assign correct value to interaction_indices
interaction_indices = c(x,y)

## assign correct value to weight_index
weight_index = NULL

## construct a scored-network
scored_net = construct_scored_net(network_data=network_data,
                  interaction_indices= interaction_indices,
                  weight_index=weight_index, gene_ps=gene_ps )
```

2.2.2 Description of the construct_string_net function

We also implemented the construct_string_net function to provide automatic construction of a node-scored network via on line retrieving gene/protein interaction information from the STRING database (version 10) [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 and loaded. Details of installing and using STRINGdb can be found on the Bioconductor website https://bioconductor.org/packages/release/bioc/html/STRINGdb.html. Also, **make sure that you are connected to the internet**.

The arguments of construct_string_net include:

- gene_ps: A data frame 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 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. Either a gene_scores or a gene_ps 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. 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 $N(0,1)$

- evidence2exclude: The interaction evidence to be excluded for constructing the network for your analysis. Way of assignment: evidence2exclude = NULL if no evidence to be excluded; or evidence2exclude = c("evidence_1", "evidence_2", ...). 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 description of each interaction evidence please refer to the STRINGdb R package. Some of these evidences may not be relevant to your network analysis thus can be excluded. In construct_string_net, we set evidence2exclude = c("textmining_transferred", "textmining") as default, thus to exclude the interaction evidence obtained using the text-mining methods, whose accuracy remains to be improved. When excluding some evidences, the combined confidence score (=interaction weight) will be recomputed based on the retained 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 a retained evidence

- confidence: A combined interaction between two genes is only retained if the combined confidence score is above confidence, which can be set as any value between 0 and 1. Removing interactions of low confidence can exclude false information and reduce the computational time. In construct_string_net, we set confidence = 0.7 as default. This results in a high-confidence network according to the categorization described in the STRING article [von Mering et al., 2005] (low confidence: scores < 0.4; medium: 0.4 to 0.7; high: > 0.7)

- 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:

1) genes2exclude = NULL: no gene will be excluded

2) 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 https://www.genenames.org/cgi-bin/genefamilies/set/141. The OR genes form a huge hairball structure (a group of genes having very 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

3) genes2exclude = your_genes_to_exclude: genes included in the your_genes_to_exclude will be excluded, where your_genes_to_exclude is a vector of genes

Return of this function:

The return of this function is a node-scored network object in the igraph network format

Example usage **(be sure you are connected to the internet**):

```
############### Read gene p-values data ###############
## For example, your p-values data has a tab delimited text format
gene_ps_file = 'your_gene_p_values_data.tab'
gene_ps = read.table(gene_ps_file, stringsAsFactors = FALSE, header =
TRUE)

## construct the string-based scored-network:
scored_net = construct_string_net( gene_ps = gene_ps )
```

2.2.3 Some notes about constructing a scored gene network

1) A gene score (or p-value) can be overloaded to the network only if the gene is also in the network (with exactly the same name, case sensitive). In both construct_scored_net and construct_string_net, we have converted all gene identifiers into upper case to avoid mismatch

2) 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 identifier as is used in the gene p-value/score file

3) SigMod only handles undirected networks. In the case which there is direction over the interaction in the source network data, this information is ignored

4) SigMod only handles simple networks. A simple network is a network which does not contain loops and multiple edges. We used the simplify function to convert a network into a simple one in construct_scored_net and construct_string_net. We also suggest the user to convert their source network into a simple network before calling these functions

2.3 Identify a gene module from the scored-network

We implemented the SigMod_bisection function to identify a gene module from scored-network constructed by construct_scored_net or construct_string_net. This function implements the strategy that was described in section 1.3. Description of the SigMod_bisection function is given below.

Parameters of the SigMod_bisection function:

- net: The scored-network that is used for searching the module. net should be a network object in the igraph network format. net can be constructed using the construct_scored_net function or construct_string_net function
- lambda_max: The upper bound of the interval 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 λ value (described in section 1.3). It should be noticed that the amount of finally selected genes is usually smaller than this number, because in the last step of the algorithm the genes that do not have any interaction with other selected genes will be removed. We set nmax = 300 by default. We suggest the user also to try different values, for example, 200, 300, 400, 500..., and to compare their results
- maxjump: A threshold that indicates whether a λ value is big enough to lead selection of strongly interconnected genes (described in section 1.3). We set maxjump = 10 by default. Increasing this threshold may lead to selecting a module that has more strong interconnection, but the overall module score can be smaller. We suggest the user to try different values of maxjump, for example, 5, 10, 20, 30, 50, and to compare their results

Return of this function:

The return of this function is a list that contains various module selection information, including:

- net: the inputted net

- opt_module: a list that contains two modules. The first module is the optimal module that is selected (opt_module[[1]]), while the second module is the subsequent module in the same selection path (opt_module[[2]]). Though the user can choose directly opt_module[[1]] as the final selected module, opt_module[[2]] 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

The analysis results will be saved automatically to the working folder, which include:

- SigMod_computation_details.Rdata: the return of the SigMod_bisection as described above
- selected_genes.tab: a tab delimited text le that contains the genes in the selected module
- selected_genes_next.tab: a tab delimited text le that contains the genes in the subsequent module of the selected module that deserves to have a look
- selected_module.pdf: the plot of the selected module
- selected_module_next.pdf: the plot of the subsequent module
- hist_selected_genes_p_values.pdf: a histogram of the p-values of the genes in the selected module and the genes in the whole network. This plot will be created if a gene p-values data is used to construct the scored network
- hist_selected_genes_next_p_values.pdf: a histogram of the p-values of the genes in the subsequent module and the genes in the whole network. This plot will be created if a gene p-values data is used to construct the scored network
- selected_module.pdf: the plot of the selected module
- selected_module_next.pdf: the plot of the subsequent module
- selected_module_interactions.tab: the interactions in the selected module
- selected_module_next_interactions.tab: the interactions in the subsequent selected module

Example usage:

```
## identify module from the scored_net
## scored_net is the return of construct_scored_net function or
construct_string_net function

## results will be saved under the working folder
res_info = SigMod_bisection( net=scored_net )
```

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, 2017. doi: 10.1093/bioinformatics/btx004. URL +http://dx.doi.org/10.1093/ bioinformatics/btx004.

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 protein lcprotein 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 protein lcprotein 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.