

dmGWAS 2.0: dense module searching for genome-wide association studies in protein-protein interaction network

Peilin Jia^{1,2}, Siyuan Zheng¹ and Zhongming Zhao^{1,2,3}

¹Department of Biomedical Informatics, ²Department of Psychiatry, ³Vanderbilt-Ingram Cancer Center, Vanderbilt University, Nashville, TN 37232, USA

December 3, 2010

Author note: Improvement in version 2.0 compared to 1.0 is described in page 4, box 1.

1. Introduction

Genome-wide association studies (GWAS) have greatly expanded our knowledge of common diseases by discovering many susceptibility common variants. Several gene-set based methods that are complementary to the typical single marker / gene analysis have recently been applied to GWAS datasets to detect the combined effect of multiple variants within a pathway or functional group. These methods include Gene Set Enrichment Analysis (GSEA), which was adapted from microarray expression data analysis (Wang *et al.*, 2007; Perry *et al.*, 2009), SNP ratio test (O'Dushlaine *et al.*, 2009), and hypergeometric test. One of the potential issues of these methods is that by sorting genes into classical pathways or functional categories the results might be much limited to *a priori* knowledge (e.g., pre-defined gene groups) in certain cases and make it difficult or ineffective for identifying a meaningful combination of genes (Ruano *et al.*, 2010). Another limitation is the incomplete annotations of pathways or GO annotations.

dmGWAS is designed to identify significant protein-protein interaction (PPI) modules and, from which, the candidate genes for complex diseases by an integrative analysis of GWAS dataset(s) and PPI network. It implements a dense module searching method previously developed for gene expression data analysis (Ideker *et al.*, 2002). We adapted the method specifically for GWAS datasets, including data preparation, integration, searching, and validation in GWAS permutation data. Specifically, we proposed two strategies to select modules for single GWAS and multiple GWAS datasets. In the later case, additional GWAS dataset(s) can be used for evaluation/validation of the modules identified by the primary (discovery) GWAS dataset.

Compared with pathway-based approaches, this method introduces flexibility in defining a gene set and can effectively utilize local PPI information. Our applications of *dmGWAS* in schizophrenia and breast cancer have shown that *dmGWAS* is sensitive and effective in identifying candidate modules and genes for the disease (Jia *et al.*, 2010). Moreover, the functions for GWAS data preparation implemented in *dmGWAS* are applicable for other cases of gene set based analysis, such as SNP-gene mapping and gene-based association computation.

2. Methods and Implementation

2.1 Workflow

dmGWAS implements several comprehensive methods for the analysis of GWAS and network data. The workflow of *dmGWAS* is illustrated in Fig. 1 and is described as follows.

Step 1. SNP-gene mapping: The association results from GWA studies (e.g., assoc file from PLINK) are used as the input of *dmGWAS*. SNPs are mapped to genes according to an annotation file containing SNP-gene coordinate information.

Step 2. Assigning statistical values for SNPs to genes: A summary *P* value for each gene is computed in this step. Several options are available, including using the most significant SNP, by the Simes method (Chen et al., 2006), by Fisher's method, or using the smallest gene-wise FDR value (Peng et al., 2010).

Step 3. Dense module searching: The function *dms* is executed to perform the following analyses: (1) constructing a node-weighted PPI network, (2) performing dense module searching and generating simulation data from the random networks, (3) normalizing module scores using the simulation data, (4) removing unqualified modules, and (5) ranking the generated modules according to their significance values. The primary results are generated in this step.

Step 4. Module selection: For multiple GWAS datasets, a strategy to mutually evaluate two GWAS datasets is implemented, i.e., using one as the discovery dataset and the other as the evaluation dataset, and vice versa. Thus, the modules significant in both GWAS datasets can be selected as the candidate modules. This strategy is strongly recommended by the authors of *dmGWAS* since it provides complementary analysis for multiple GWAS datasets. Alternatively, if a single GWAS dataset is under investigation a function is provided to select the top ranked modules.

Step 5. Evaluation by permutation: The selected modules are further evaluated using the permutation data based on the original GWAS dataset(s).

Finally, the selected modules are combined to construct a subnetwork specific for the disease under investigation. It can be graphically displayed in the environment of R.

The input and output for each function in each step are straightforward, and data preparation is easy. Thus, it is easy for the users to incorporate their own data or to integrate some functions in *dmGWAS* into their own programs.

2.2 Methodology

Dense module searching algorithm

The score for a module is defined as

$$Z_m = \frac{\sum z_i}{\sqrt{k}}$$

where *k* is the number of genes in the module and z_i is transferred from *P* value according to $z_i = \Phi^{-1}(1 - p_i)$ (Φ^{-1} denotes the inverse normal distribution function.) (Ideker *et al.*, 2002).

Module score is further normalized by

$$Z_N = \frac{Z_m - \text{mean}(Z_m(\pi))}{\text{sd}(Z_m(\pi))}$$

where $Z_m(\pi)$ is generated by randomly selecting the same number of genes in a module from the whole network 100,000 times.

Searching strategy

The following steps perform greedy searching iteratively using each gene in the network as a seed gene.

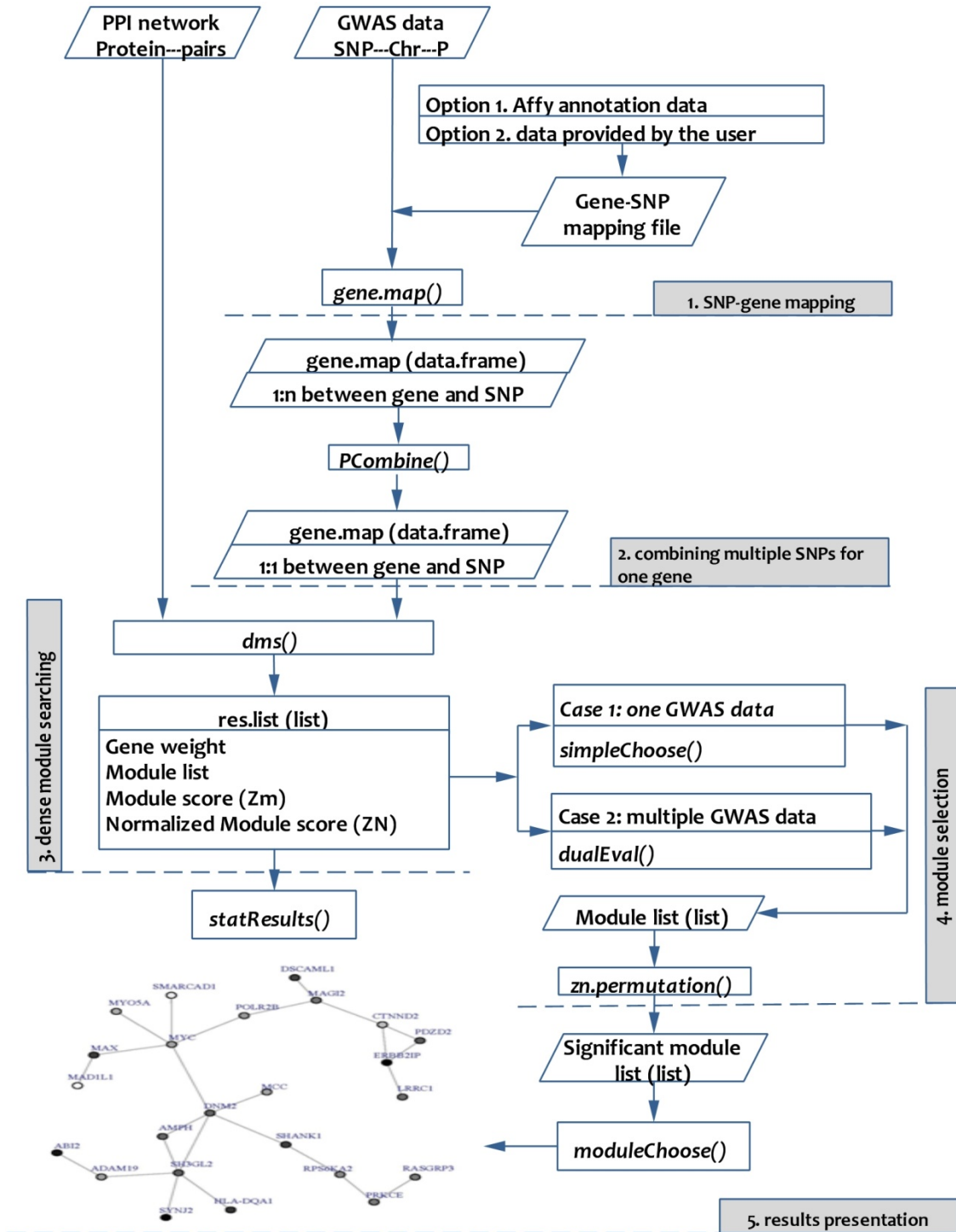


Fig. 1. Flowchart of *dmGWAS*. PPI: protein-protein interaction. Chr : chromosome.

Step 1: A seed module is assigned. In the beginning, the seed module contains only the seed gene. Z_m is computed for the current seed module.

Step 2: Identify neighborhood interactors, which are defined as nodes whose shortest path to any node in the module is shorter or equal to a pre-defined distance d (e.g., $d = 2$).

Step 3: Examine the neighborhood genes defined in step 2 and find the genes that can generate the maximum increment of Z_m . Nodes will be added if the increment is greater than $Z_m \times r$, where r is the rate of proportion increment. That is, the expanded module has a score Z_{m+1} greater than $Z_m \times (1+r)$.

A substantial improvement has been made in version 2.0 (see Box 1 below). Because of this update, the execution time of dmGWAS version 2.0 is longer than in version 1.0: it takes ~2 hours on a server (3.00 GHz Quad Core Intel® Xeon® Processor X5450 and 16.0 GB of RAM, one thread) in version 1.0, but 3-4 hours in version 2.0. However, the results are more accurate in version 2.0.

Step 4: Repeat steps 1-3 until adding any neighborhood nodes cannot yield an increment that is greater than $Z_m \times r$.

We suggest that modules with less than 5 genes not be considered. When multiple modules have the same component genes but are generated by different seed genes, only one is kept. In our following analysis, we used $d = 2$ and $r = 0.1$. The results using other d and r values are described in Box 2.

Box 1. Substantial improvement in version 2.0

In the case when $d > 1$, neighborhood genes will include both direct and indirect interactors. For example, in Figure 1, the direct interactors of the seed module are G6 and G7 ($d=1$), and the indirect nodes with $d=2$ to the seed module is G8. When none of the direct interactors can generate $Z_{m+1} \geq Z_m \times (1+r)$, we will move on to the nodes with $d=2$, or, G8. In our version 1.0, we only take into account the node G8 itself, i.e., when G8 is included, a shortest path between G8 and the seed module will be randomly selected to connect them. In version 2.0, we consider gene weights along the paths and only the path with the strongest weight will be included. In this case, the path G8-G6-G5 will be selected and included into the module. This helps us identify more relevant genes.

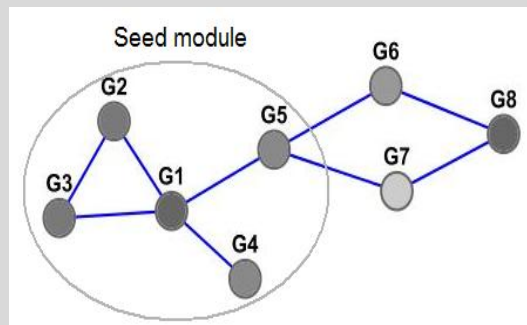


Figure 1. Node selection in module expansion process. Node color is positively proportional to the node weight.

3. Example

Step 1. SNP-gene mapping

Input file 1 - the association result file from GWAS. The direct output file from PLINK, .assoc file, is workable. If you use other software, make sure the file contains two columns: SNPs and their P values.

Input file 2 - annotation file. To map SNPs to genes, a file containing the coordinate information of genes and SNPs needs to be prepared in advance. The file has to be space or tab-delimited and contain SNP ID, the distance of the SNP to the gene, and gene name. A header line is required to indicate the columns for "SNP", "Dist", and "Gene". For example,

```
$ head -5 GenomeWideSNP_5.na30.annot.AffyID
SNP          Dist      Gene
AFFX-SNP_10000979 0          ABCA8
AFFX-SNP_10009702 282266    RPS18
AFFX-SNP_10009702 73369     HS3ST3B1
AFFX-SNP_10009702 76596     HS3ST3B1
```

Users can do this by using PLINK (e.g., --gene-report) and format it to a file containing the three columns.

Alternatively, we provide two annotation files for two Affymetrix chips, which can be downloaded from our website. They are the annotation files from Affymetrix website for their most popular arrays: **Affymetrix Genome-Wide Human SNP Array 5.0** (GenomeWideSNP_5.na30.annot.csv.zip) and **Affymetrix Genome-Wide Human SNP Array 6.0** (GenomeWideSNP_6.na30.annot.csv.zip). Our annotation files online contain only the necessary columns for the use of *dmGWAS*. Annotation data for other platforms (e.g., Illumina) will be available soon.

To map SNPs in the GWAS data to the corresponding genes:

```
> gene.map=SNP2Gene.match(assoc.file="gwas.assoc",
snp2gene.file="GenomeWideSNP_5.na30.annot.AffyID", boundary=20)
```

where boundary is the distance a SNP is included for a gene, e.g., boundary=20 means 20kb border is added to the start and end of each gene listed in the gene file.

```
> head(gene.map,3)
  Gene      SNP      P
1 MAGI2 SNP_A-1780270 0.7350
2 UQCC  SNP_A-1780274 0.2165
3 CAPS2 SNP_A-1780277 0.6848
```

Step 2. Assign *P* values of SNPs to genes

Since there are multiple SNPs for each gene and *dmGWAS* requires one summary *P* value for a gene, we implement several methods in *dmGWAS* to combine multiple SNPs for a gene:

- 1) using the most significant SNP (*method="smallest"*);
- 2) the Simes method (Chen et al., 2006) (*method="simes"*);
- 3) Fisher's method (*method="fisher"*);
- 4) using the smallest gene-wise FDR value (Peng et al., 2010) (*method="gwbon"*).

For example,

```
> gene2weight = PCombine(gene.map, method="smallest")
> head(gene2weight, 3)
      gene      weight
A2BP1  A2BP1 0.0003139
A2M    A2M   0.8806000
A2ML1  A2ML1 0.1080000
```

The object, *gene2weight*, is simply an object of data.frame with two columns, gene and weight (*P* value). If the users prefer their own way of assigning weight to genes, they can provide a data frame that contains the same information as is returned from *SNP2Gene.match*, i.e., a data frame containing two columns "gene" and "weight". Note that the weight must be *P* values since our algorithm takes *P* value as input and transfers it to Z-score.

Step 3. Dense module searching:

A PPI network is needed in the format of protein interaction pair. For example,

```
> network = read.table("network.txt")
> head(network, 3)
      V1      V2
1  TAF4  SAP130
2  HDAC3  HDAC1
3  TAF1   TAF15
```

Next, one single function, *dms*, performs all the analysis necessary for dense module searching. The detail of the algorithm can be found in our manuscript (please see the References section on our web site). Four input parameters are necessary for the execution of this function:

gene2weight: generated in step 2.

network: pair-wise PPI data.

d: defines searching area in the network topology, i.e., within neighborhood genes that are defined as those with the shortest path to any node in the module less than or equal to *d*.

r: the cutoff for incensement during module expanding process. The score improvement of each step is required as passing $Z_{m+1} > Z_m \times (1+r)$ for the inclusion of any neighborhood genes, where Z_{m+1} is the module score by recruiting a neighborhood node.

Box 2 explains how to choose values for *d* and *r*. The command line to execute *dms* is:

```
> res.list = dms(network, gene2weight, d=2, r=0.1)
```

The resultant object, *res.list*, contains all the results from the searching process, including the node-weighted network used for searching, the resultant dense modules and their component genes, the module score matrix containing Z_m and Z_N , and the randomization data. A resultant file called **RESULT.RData** will also be generated for future recalling.

```
> names(res.list)
[1] "GWPI"          "graph.g.weight"  "genesets.clear"  "genesets.length.null.dis"
"genesets.length.null.stat" "zi.matrix"      "zi.ordered"
```

#GWPI, an object of igraph class, is the node-weighted PPI network.

#graph.g.weight, an object of data.frame, contains the gene in GWPI and the weight (z_i , transformed from *P*).

#genesets.clear, an object of list, contains all the valid modules. The name of each record is the seed gene.

#genesets.length.null.dis, an object of list, contains the randomization data of for each size of modules.

#genesets.length.null.stat, an object of list, contains the statistic values of randomization data of for each size of modules.

#zi.matrix, an object of matrix, contains data for each module: gene (seed gene), Z_m (module score), Z_n (normalized module score).

#zi.ordered, ordered matrix of zi.matrix based on Z_n .

Box 2. How to choose d and r

In our analyses, d has a marginal effect on the results; thus, it's recommended that d is set as the default value 2.

The parameter r impedes restriction on the score of the module. When r is small, it imposes loose restriction during the module expanding process; thus, unrelated nodes with lower z_i scores (higher P values) might be included. On the other hand, when r is large, strict restriction is imposed and only those nodes with very high z_i scores (very low P values) could be included. As a result, it may miss some informative nodes that have moderate association P values.

In our application in a schizophrenia GWAS dataset, we compared the performance using four r values (0.05, 0.1, 0.15, and 0.2). Fig. 2 shows the results. It can be seen that when r is small, the size of modules tends to be large (e.g., when $r=0.05$, average number of module genes is 19.1). When r is large, the size of the modules becomes small (e.g., when $r=0.2$, it is 6.6, slightly larger than our minimum number of genes for a module, 5). Our results using $r=0.1$ worked well and sensitively (average number of genes is 11.1). However, it is recommended that the users try different r values on their own data to decide the most appropriate value.

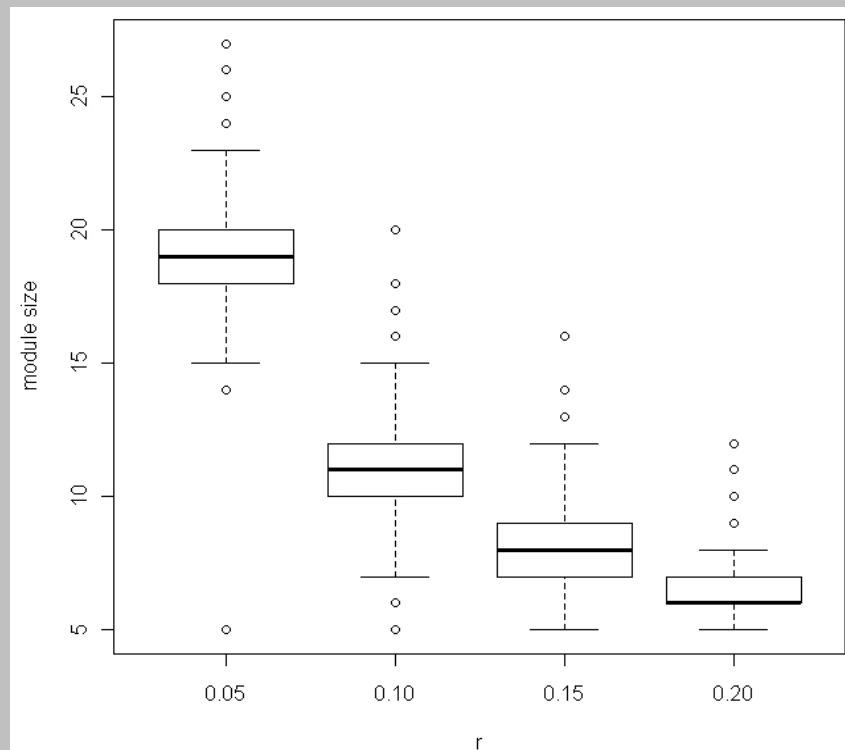


Fig. 2. Comparison of module size with different r values in dense module searching using schizophrenia GAIN GWAS dataset.

To have an overview of the generated modules, the function `statResults` is provided to briefly show the network features of top ranked modules (Fig. 3):

```
> statResults(res.list, top=100)
```

This function walks along the module list (i.e., walks along the matrix `res.list$zi.ordered` from top), ranked by their normalized score, Z_N , from the highest to the lowest, and add one module at each step to generate a combined subnetwork. In each step, the number of genes involved in the modules, clustering coefficient of the subnetwork, average shortest path, and average degree are plotted to give an overview of the top ranked modules.

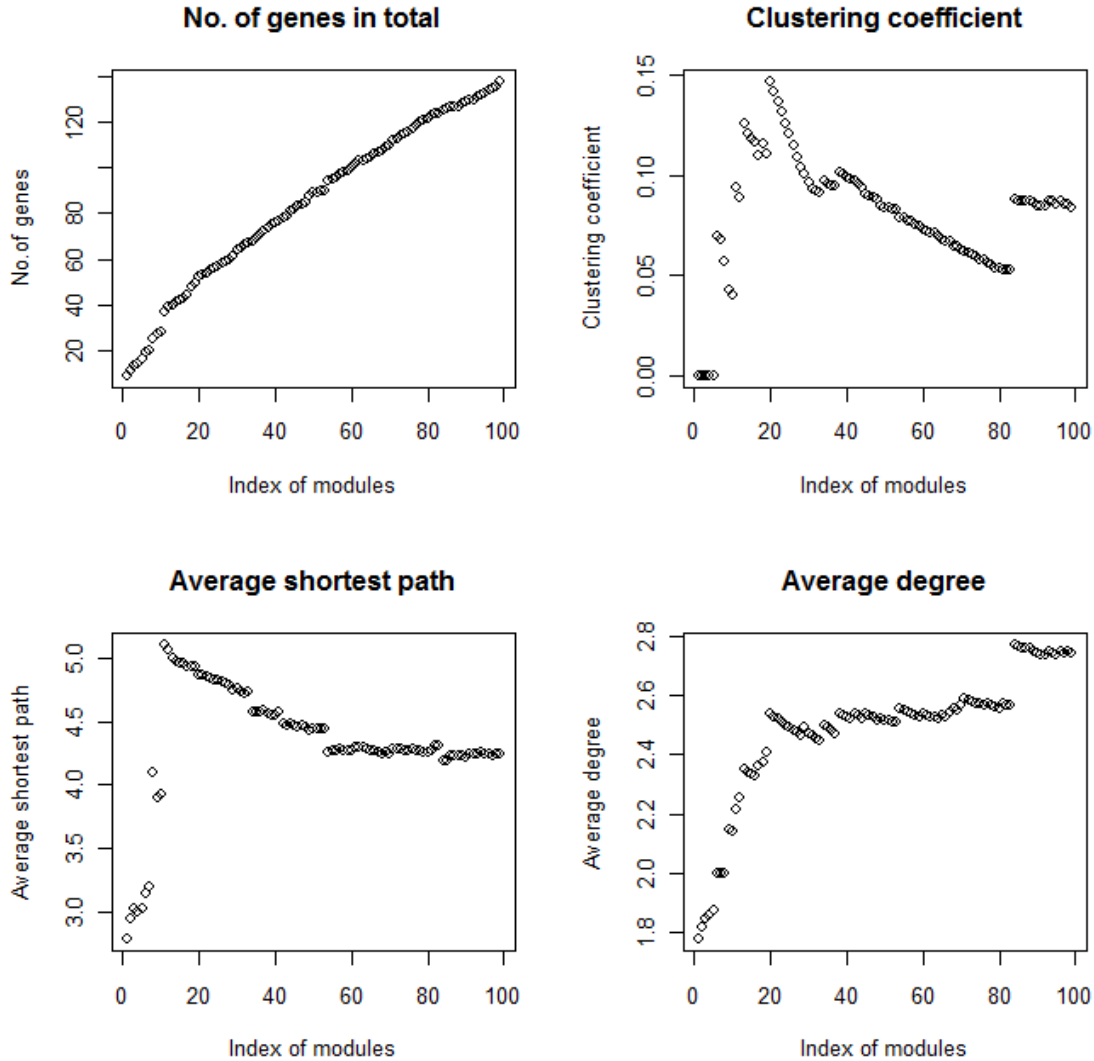


Fig. 3. Network features of the top ranked modules.

Step 4. Module selection

Modules are ranked and selected by Z_N . Theoretically, and also based on our application, each gene has a local module; thus, there may be thousands of modules generated with extensive overlap between modules because of the complex structure of the human PPI network. We propose two strategies for module selection.

Strategy 1. Module selection for single GWAS dataset:

```
> selected = simpleChoose(res.list, top=100, plot=T) #for single GWAS dataset
```

Strategy 2. Module selection for multiple GWAS datasets:

```
> zod.res = dualEval(resfile1, resfile2) #for two GWAS dataset
> names(zod.res)
[1] "zod"      "zod.sig"
> head(zod.res$zod.sig) #the significant modules
      gene   zi.dis   za.dis nominalP.dis   zi.eval   za.eval nominalP.eval
4293 ATP2B2 10.25564 6.882083         0 8.993793 3.090531    0.00076
7234 GRID1  10.18871 6.809900         0 8.950589 3.047315    0.00085
1824 FNBP1  10.13180 6.748524         0 8.825186 2.921878    0.00121
1928 GRIN2A 10.11515 6.730566         0 8.612028 2.708662    0.00287
1555 GUCY1A2 10.07223 6.684281         0 8.655341 2.751987    0.00253
4052 NRXN3  10.30150 6.673032         0 9.801202 3.564755    0.00012
```

where *resfile1* and *resfile2* are generated independently using *dms* for dense module searching. In *dualEval*, *resfile1* is used as discovery dataset and *resfile2* is used as evaluation dataset.

Alternatively, the following function is provided for users' convenience so that modules can be selected by a user-specified seed gene list:

```
> selected = moduleChoose(seedgenes, res.list, plot=T) # choose modules by user-specified "seed gene" list
```

Step 5. Evaluation by permutation

Modules generated in the above steps need to be validated by permutation of the original GWAS dataset. Permutation data can be generated using PLINK; thus, they are in the same format as being used in the real case. Each round of permutation will generate a file. All files are required to be saved in the same folder.

The selected modules will then be subjected to permutation data for testing associations with the disease of interest.

```
> module.list = res.list$genesets.clear[1:10] #an example to show how to get the list of modules
> res = zn.permutation(module.list, gene2snp, gene2snp.method= "smallest", original.file, permutation.dir)
```

where *original.file* is the location of the original assoc file and *permutation.dir* saves all the permutation files. The other parameters, *gene2snp* and *gene2snp.method*, need to be the same as used in steps 1 and 2.

Both the empirical P values and the module scores computed by permutation data are returned to *res*:

```
> names(res)
[1] "Zn.EmpiricalP"  "Zn.Permutation"
> head(res$Zn.EmpiricalP)
      Seed      Zm EmpiricalP
1   TAF4  0.108389063519163    0
2  HDAC3  0.0969500793246838    0
3   TAF1  0.146814408612484    0
4 HSP90AA1 0.110336728792522    0
5  L3MBTL2 0.184730745613557    0
6   FGF3  0.138878891433732    0
```

Finally, use the function *moduleChoose* to combine the significant modules and show the resulting graph. A resulting graph is shown in Fig. 4.

```
> subnetwork=moduleChoose(zod.res$zod.sig[,1], res.list, plot=T)
```

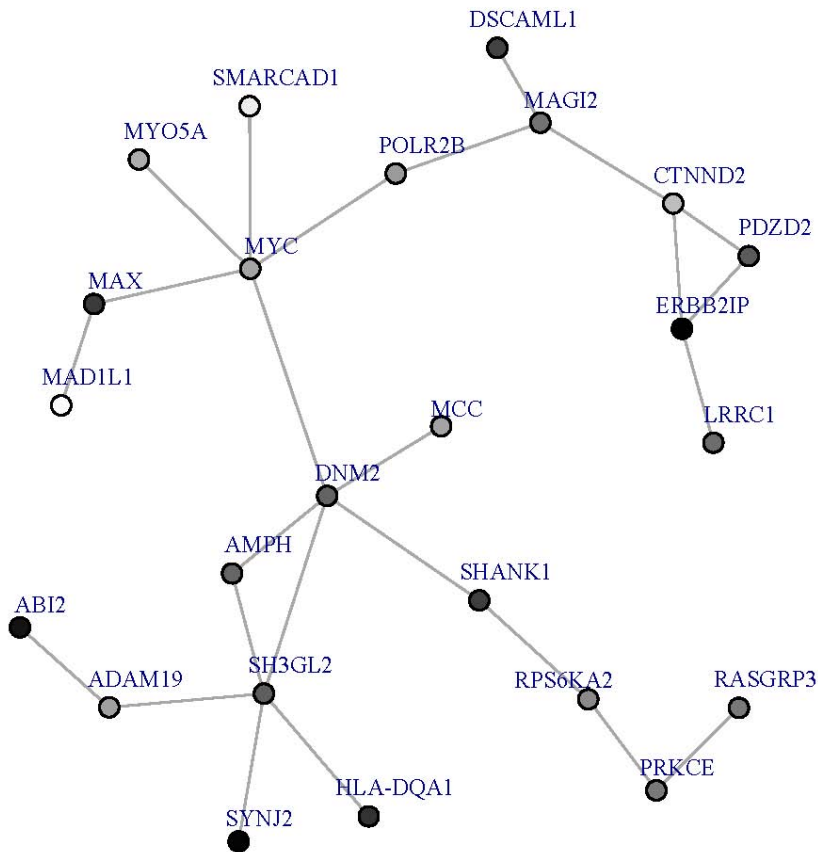


Fig. 4. An example of subnetwork generated by function *moduleChoose*.

References

- Ideker, T. *et al.* (2002) Discovering regulatory and signalling circuits in molecular interaction networks. *Bioinformatics*, **18 Suppl 1**, S233-240.
- Jia, P. *et al.* (2010) dmGWAS: dense module searching for genome-wide association studies in protein-protein interaction networks. *Bioinformatics*.
- O'Dushlaine, C. *et al.* (2009) The SNP ratio test: pathway analysis of genome-wide association datasets. *Bioinformatics*, **25**, 2762-2763.
- Perry, J.R. *et al.* (2009) Interrogating type 2 diabetes genome-wide association data using a biological pathway-based approach. *Diabetes*, **58**, 1463-1467.
- Ruano, D. *et al.* (2010) Functional gene group analysis reveals a role of synaptic heterotrimeric G proteins in cognitive ability. *Am J Hum Genet*, **86**, 113-125.
- Wang, K. *et al.* (2007) Pathway-Based Approaches for Analysis of Genomewide Association Studies. *The American Journal of Human Genetics*, **81**, 1278-1283.