Gene Co-expression Visualization

Introduction

Gene co-expression network is an undirected graph in which each node corresponds to a gene and each edge represents the co-expression relation between a pair of genes (van Dam *et al.*, 2017). The co-expression relationship is usually identified by calculating the correlation coefficient between the expression levels of a pair of genes. Based on these relationships, the researchers could identify the groups of genes with correlated expression levels and study the functions they are collectively involved in, which provides different insights other than the studies focus on individual genes.

In this project, the program written in Go programming language will take a gene expression profile provided by the user, identify the co-expressed gene groups, and enable the users to visualize the co-expression network interactively. Besides, the identified co-expressed gene cluster can also be used for Gene Ontology (GO) (Gene Ontology Consortium, 2004) or pathway enrichment analysis, and an example using the web application shinyGO (Ge & Jung, 2018) is provided in this report.

Methods

(a) Data processing

Data filtering

To filter out the genes with very low expression levels, those with expression levels below a certain threshold (1 was used in the example) in all samples will be deleted, and the remaining expression profile will be used in further analysis steps.

Similarity matrix

In order to generate a matrix containing the similarity scores between each pair of genes in the input expression profile, the Pearson correlation coefficients between each pair are calculated by the function CalculateCorrelation() according to Formula 1.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

(https://en.wikipedia.org/wiki/Pearson_correlation_coefficient)

Formula 1

Graph adjacency matrix

Based on the similarity matrix generated in the previous step, each pair of genes will be evaluated according to their similarity score. If the similarity score between two genes (g_i and g_j) has value are greater than the threshold defined by the user (e.g 0.85) the cell (i,j) in the graph adjacency matrix will have the value of 1; if the similarity is smaller than the opposite number of the threshold, cell (i,j) will have value of -1, otherwise 0.

Graph generation and traverse

A co-expression graph is built according to the adjacency matrix where an edge

between the two genes g_i and g_j corresponds to the value 1 or -1 in the cell (*i,j*) and stored in a CoExprsTable object in Go.

The complete co-expression graph could have multiple connected components disjoint from each other, each represents one group of co-expressed genes. In order to identify and separate them for further use, the Depth-First Search algorithm (Formula 2) is implemented, and the resulting separated components are stored in a slice of CoExprsTable objects.

```
DFS-iterative (G, s):                                    //Where G is graph and s is source vertex
  let S be stack
  S.push( s )              //Inserting s in stack
  mark s as visited.
  while ( S is not empty):
      //Pop a vertex from stack to visit next
      v  =  S.top( )
     S.pop( )
     //Push all the neighbours of v in stack that are not visited
    for all neighbours w of v in Graph G:
        if w is not visited :
                  S.push( w )
                 mark w as visited
```

(https://www.hackerearth.com/zh/practice/algorithms/graphs/depth-first-search/tutorial/)

Formula 2

(b) Results Generation

Text-based results

In the “text” mode, the user needs to give three addition parameters: input file name, output file name, and correlation threshold. After entering the "text" mode, a function pipe() which pipes all functions involved in matrices generation, graph generation and graph traversal, passing the parameters to relevant functions, and writing the results to the user-determined file. The results will be written into a single text file, containing a list of co-expressed gene groups and the gene names in each group.

Web application

A web app is built and served by Go to enable users to upload files, choose their interested list of genes and visualize the network interactively.

There are three handler functions written in Go to serve the three webpages of the application:

1. upload_handler() is the handler function that serves the index page of the app, where the users start the analysis by uploading the gene expression profile file. It examines whether the uploaded file has legit size, if yes, it will store it to a temporary text file that will be deleted before running a new analysis.

2. GraphContentHandler() serves the web page where all co-expressed gene clusters are displayed in a table and the user can choose one to visualize interactively in a new tab.

3. GraphPageHandler() serves the web page "/Graph" which will be opened in a new tab after clicking the "Generate Graph" button in the "/GraphList" page. The interactive graph is generated using D3.js, and an open-source force-direct graph model developed in D3.js is used as a prototype (https://bl.ocks.org/heybignick/3faf257bbbbc7743bb72310d03b86ee).

Gene Ontology (GO) enrichment analysis

The GO enrichment analysis was performed on an external website shinyGO (http://bioinformatics.sdstate.edu/go/).

Results

The example dataset was downloaded from GEO database under accession ID GSE121017, in which the gene counts for 26,485 genes had already been normalized but not filtered. The filtering threshold set up for this dataset is 1, and there are 10460 remain after being filtered. However, analyzing the whole dataset will take several minutes, so 200 genes randomly selected from the unfiltered original dataset will be used as the example in the following paragraphs (/Data/sample_200_rand.txt).

When running the program in text mode using the command "`$ ./Go_codes.exe text Data/sample_200_rand.txt Results/test_200_Dec_5.txt 0.85`", the progress of analysis can be seen in the terminal as shown in Figure 1.

```
Correlation coeffients calculated!
Co-expression relation decide!
Co-expression graph generate! Now finding gene clusters...
Gene clusters found!
Results generate! Please see your results in:  Results/test_200_Dec_5.txt
```

Figure 1

The results of the analysis written in the text file are shown below.

```
The input file is:  Data/sample_200_rand.txt
There are 6 samples
There are 200 genes initially

There are 75 genes after filtering

This is the co-exprs graph:
No. of nodes in the graph: 75
No. edges in the graph: 287
There are 147 positive relations and 140 negative relations

There are  2  co-expressed gene clusters in the graph
```

```
The  0 -th cluster contains 66 genes:  [TMEM126B PRR19 TYMS CCDC24 LIMD1
IKBKB FBXO9 ALDH9A1 INTS12 OXSM USMG5 C1RL-AS1 CDK2AP1 MAP1LC3B FBXO44 PIK3R4
MAU2 DIEXF RPAP3 PSMD4 PPP6C STIM2 LINC00963 HTRA2 ADD3 TAF7 FAM83H PRICKLE3
MSH6 ZNF276 PTGR2 CCAR2 TNFRSF10A TMEM161A OXSR1 LGALS8 FAM120A OFD1 CFAP20
METTL17 MTMR6 ZNF589 PRSS21 GRINA MCM6 C11orf80 RPS29 CARD16 GATB PEX19 RBM7
MAZ SF3B5 QRSL1 PLCB3 ILK OGT IER3 TMEM132A MRPS15 RIPK1 MAP2K5 ELP3 DYNC2LI1
PPP1R2 MNAT1]
The  1 -th cluster contains 2 genes:  [DCUN1D4 SMC2]
```

When running the program in web mode, the only parameter the user needs to give is "web", then the server will be started and the web app could be seen at http://localhost:8000.

After opening the web app, there will be an upload selection form and a submit button for the user to provide the gene expression profile (Figure 2).

Figure 2

After setting the correlation cutoff to 0.85 (same as we used in text mode), selecting the Data/sample_200_rand.txt file, and clicking on the submit button, the web page will be redirected to http://localhost:8000/GraphList , where a table containing the two co-expressed gene clusters will be shown as in Figure 3.

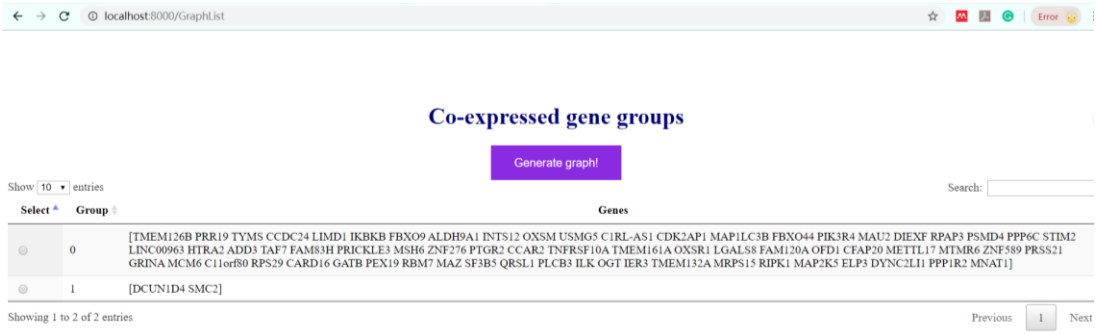

Figure 3

In this webpage, the user can use the radio buttons to select the group of genes they are interested in and visualize the co-expression network by clicking the "Generate graph" button. Then a webpage will be opened automatically in a new tab at http://localhost:8000/Graph, where an interactive graph will be shown as in Figure 4 (Group 0 was chosen).

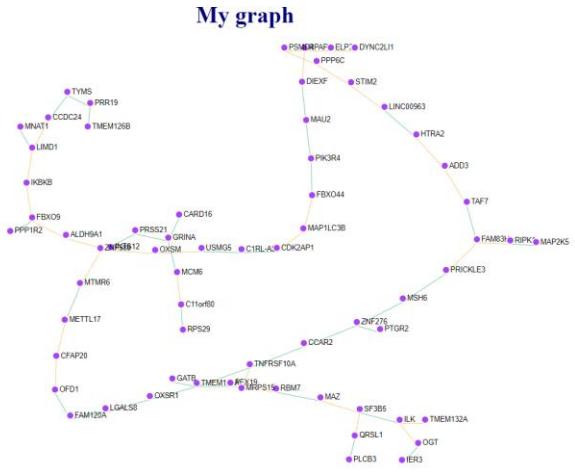

Figure 4

This interactive graph enables users to see the gene names and the relationship between each pair: a positively correlated gene pair has an orange edge between them, while a negatively correlated pair is connected by a green edge.

The gene list of the first cluster was then copied and pasted to the web application

shinyGO (http://bioinformatics.sdstate.edu/go/) for GO enrichment analysis, and the result is shown in Figure 5. It is shown that this group of genes are mainly involved in cell death and apoptotic process.

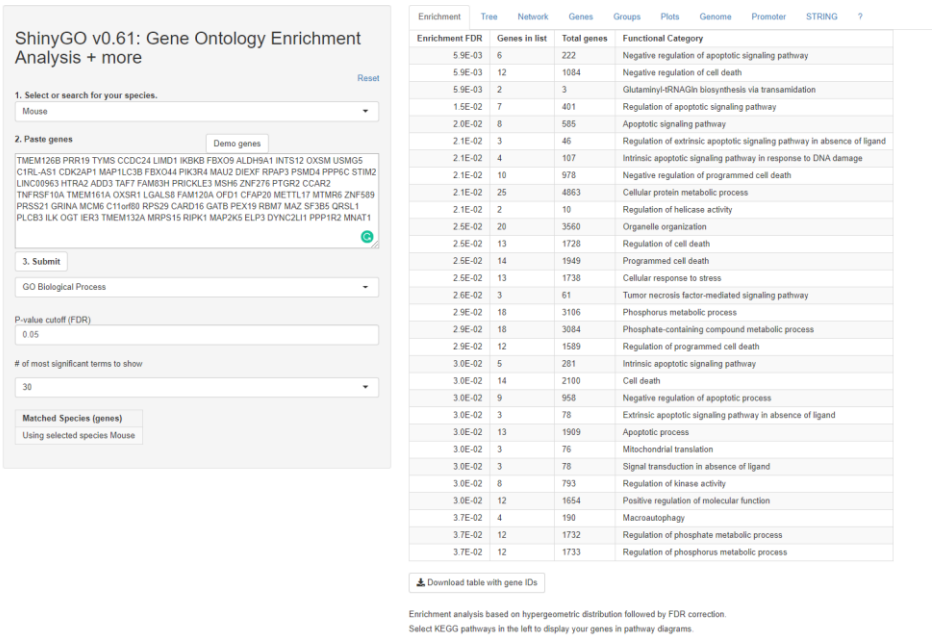

Figure 5

Discussion

There are still several aspects to be improved in this program for a better user experience.

ID conversion function

The original gene expression matrix downloaded from a public database may use gene symbols or gene accession ids for a specific database. For this program, the gene list written to the text file as well as the label on the graph will be whatever provided by the row names of the original file. Thus, it would be helpful that if the user can select the original identifier type they provided and the program can convert them to another id type that the user what to show in the graph.

Built-in GO enrichment analysis in text mode

It will be more convenient if GO enrichment could be performed locally and the results could be directly included in the output text file. I did not figure out the way to do this in Golang, but it should be applicable to trigger a system-level command for calling an R script to handle the enrichment analysis and have the results written in a single file. However, due to time constraints, this was not included in the current product.

Automatic redirection to shinyGO

In the web mode, the user needs to copy the gene list from the page to the external website, which might be inconvenient. Thus, if this step could be automated, that by clicking a button, the webpage will be redirected to the other website and the selected

gene list will be copied automatically, the web app may become easier to use.

Parallelism

When large datasets (e.g. the full profile of GSE121017 dataset) are given, the program could be running for 5 to 10 minutes before generating the results, which is not efficient. Thus, if some steps of the analysis process, such as similarity score calculation and graph traverse could be divided and assigned to multiple processors, the efficiency of the program would be increased to a great extent.

Current bugs

Some bugs are not solved yet because of time constraints. This first problem is that the expression level cutoff is hardcoded and could not be customized by the user, which would be better if making it customizable.

The other current problem is in the web part, which is that I did not figure out how to pass a value from the current handler function to the one that the web page will be redirected to. Thus, although .json files for all found co-expressed gene groups will be generated, the GraphHandler could not read the index value that was selected on the GraphList page and choose the .json file to read accordingly. I did not have enough time to figure out how to fix this in the end, so I just hardcoded the index as "0", that's why currently whichever cluster the user selected, in the Graph page it always shows the graph for the first group of genes.

Conclusion

In conclusion, for the final project, a program that can run in two modes to either generate text-based results locally containing information about co-expressed gene groups in the original expression profile or run the analysis and interactively visualize the results on a web app was developed. Also, HTML, CSS, and JavaScript scripts are involved in generating the web app. It is functional at this point, but there are still several issues to be solved and aspects could be further improved in the future.