

Software Project, summer 2012

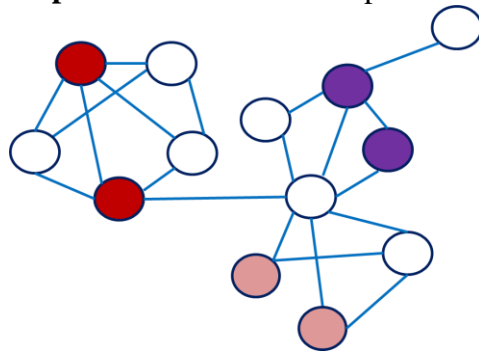
Due by 13.9.2012, 23:59

This is your final software project, it builds on the infrastructure developed in ex. 2 and we recommend using that code in your project, with the necessary alterations to the interface and body. In your final project you will cluster a biological network and detect functionality in the clusters according to you findings.

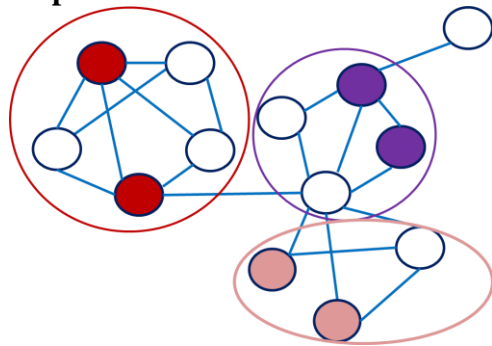
1. Introduction

Over the last decade high throughput technologies for measuring biological data have become available, changing the landscape of biological research. This enables, for the first time, a global, simultaneous view of proteins behavior and function. Proteins are the main building blocks of the cell and are responsible for its structure and function. They work by interacting with one another to build molecular machines. One can describe these interactions via a **network** $G=(V,E)$, where the nodes V represent proteins and the edges E represent the interactions. The goal of this project is to try and infer the function of unknown proteins based on the functions of known proteins that are “close” to them in the network. This is based on the observation that interacting, or close by proteins tend to share the same function. The basic idea is described by the following 3-step process:

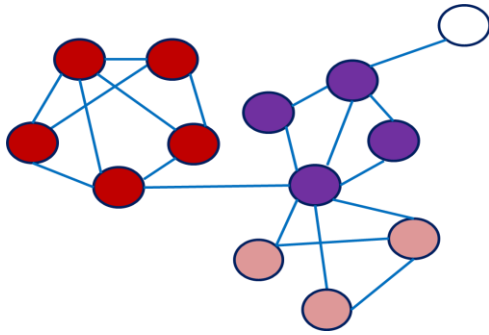
Step 1. Mark the network's proteins whose function is known:



Step 2. Cluster the network to maximize the “strength” of interactions within clusters:



Step 3. For uncharacterized proteins, infer their functions according to the functions (if known) of their cluster members:



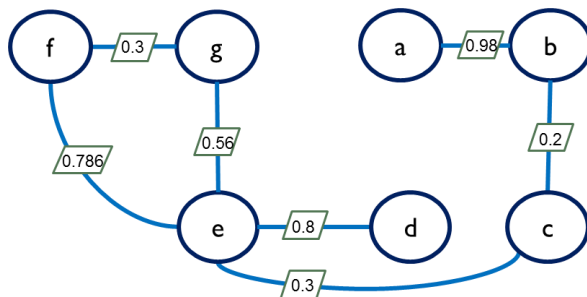
2. The Max K-Clustering Problem

The max K-clustering problem, related to the known min K-cut problem, is defined as follows: Given a network $G=(V,E)$ and a parameter K , partition the nodes into K clusters $C_1 \dots C_k$ (i.e., every node belongs to a single cluster and none of the clusters is empty) such that the weight of edges within clusters is maximized. Formally, if we denote by $E(C_i)$ the set of edges that connect nodes in cluster i then the goal is to

$$\text{maximize: } \sum_{i=1}^k \sum_{e \in E(C_i)} W_e.$$

2.1. Running Example

The example below demonstrates the problem definition, memory implementation and output.



If we had clustered this example using the random clustering in exercise 2 we could have gotten the score (sum of edge weights within clusters) of 2.826:

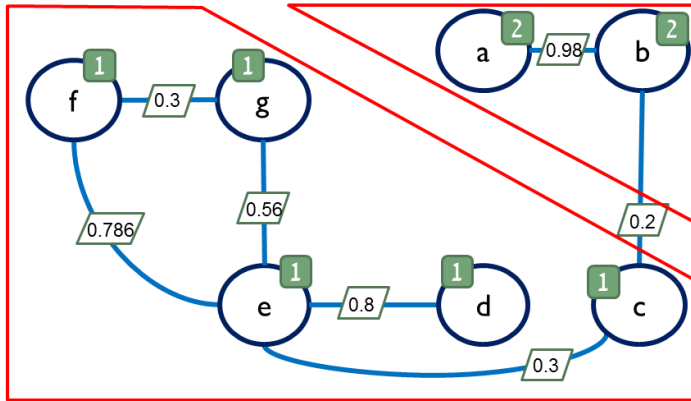


(Notice the example is not identical to the example given for exercise 2, nor is the definition of score).

However, if we were to follow the K-clustering problem, we might create the following 2 clusters, with a higher score of 3.726.

Cluster 1 score: 2.746

Cluster 2 score: 0.98



Max k-clustering score is 3.726

2.2. From K-Clustering to Integer Linear Programming

In your program you shall translate the K-clustering problem to an integer linear program, and use CPLEX c interface to solve it.

Linear programming is a technique for the optimization of a linear objective function, subject to linear constraints. For a vector x of variables and coefficient vectors a, b, c, \dots a linear program would look as follows:

maximize(minimize) cx
 subject to $ax \leq b$
 \dots
 \dots

} linear equality and
inequality constraints

If some of the unknown variables are required to be integers, then the problem is called **integer linear programming (ILP)**. In contrast to linear programming, which can be solved efficiently, integer programming is NP-hard. Nevertheless, efficient solvers exist for it that scan in an efficient way the solution space which may be exponential in size. The formulation we will use is based on [Johnson et.al. 1992; journal of Mathematical Programming, 62:133-151] and its details follow.

2.2.1 Binary Variables:

- i. Z_e^k - indication of whether edge e is in cluster k . There are $|E| \cdot K$ variables of this type.
- ii. X_i^k - indication of whether vertex i is in cluster k . There are $|V| \cdot K$ variables of this type.

2.2.2 Objective:

Maximize the sum of edges residing inside a cluster

$$\max \sum_{k=1}^K \sum_{e \in E} (W_e \cdot Z_e^k)$$

2.2.3 Constraints:

- i. For each edge $e=(i,j)$ and cluster k :

$$Z_e^k \leq X_i^k, X_j^k$$

The purpose of these constraints is to make sure that an edge is in a cluster only if its two vertices are in the cluster.

- ii. For each edge $e=(i,j)$ and cluster k :

$$Z_e^k - X_i^k - X_j^k \geq -1$$

The purpose of these constraints is to force an edge to be in a cluster if both its vertices are there.

- iii. For each vertex i :

$$\sum_{k=1}^K X_i^k = 1$$

That is, each vertex is assigned to a single cluster.

- iv. For each cluster k :

$$\sum_{i \in V} X_i^k \geq 1$$

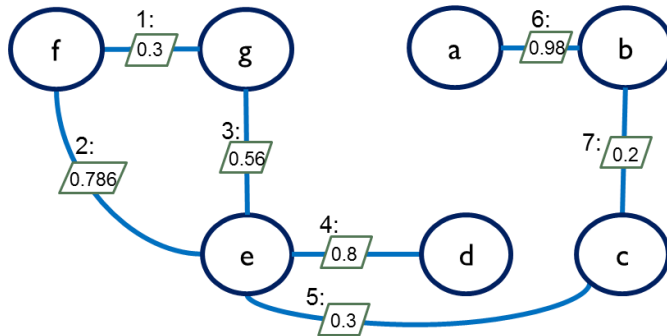
That is, each cluster has at least one vertex.

For more details see the presentation in class.

2.2.1. Running example K-clustering after translation

$|V| = 7, |E| = 7, K = 2$:

Here the nodes are labeled with numbers and edges with letters:



2.3.1 Variables:

- i. 14 variables of type Z_e^k :

$$Z_1^1, Z_2^1, Z_3^1, Z_4^1, Z_5^1, Z_6^1, Z_7^1, Z_1^2, Z_2^2, Z_3^2, Z_4^2, Z_5^2, Z_6^2, Z_7^2$$

- ii. 14 variables of type X_i^k :

$$X_a^1, X_b^1, X_c^1, X_d^1, X_e^1, X_f^1, X_g^1, X_a^2, X_b^2, X_c^2, X_d^2, X_e^2, X_f^2, X_g^2$$

2.3.2 Objective:

Maximize the sum of edges residing inside a cluster

$$\begin{aligned} \max & [(0.3 * Z_1^1) + (0.786 * Z_2^1) + (0.56 * Z_3^1) + (0.8 * Z_4^1) + (0.3 * Z_5^1) + (0.98 \\ & * Z_6^1) + (0.2 * Z_7^1) + (0.3 * Z_1^2) + (0.786 * Z_2^2) + (0.56 * Z_3^2) + (0.8 \\ & * Z_4^2) + (0.3 * Z_5^2) + (0.98 * Z_6^2) + (0.2 * Z_7^2)] \end{aligned}$$

2.3.3 Constraints:

- i. 28 constraints of type i:

For edge 1(f,g): $Z_1^1 - X_f^1 \leq 0$, $Z_1^1 - X_g^1 \leq 0$,
 $Z_1^2 - X_f^2 \leq 0$, $Z_1^2 - X_g^2 \leq 0$

For edge 2(f,e): $Z_2^1 - X_f^1 \leq 0$, $Z_2^1 - X_e^1 \leq 0$,
 $Z_2^2 - X_f^2 \leq 0$, $Z_2^2 - X_e^2 \leq 0$

For edge 3(g,e): $Z_3^1 - X_g^1 \leq 0$, $Z_3^1 - X_e^1 \leq 0$,
 $Z_3^2 - X_g^2 \leq 0$, $Z_3^2 - X_e^2 \leq 0$

For edge 4(e,d): $Z_4^1 - X_e^1 \leq 0$, $Z_4^1 - X_d^1 \leq 0$,
 $Z_4^2 - X_e^2 \leq 0$, $Z_4^2 - X_d^2 \leq 0$

For edge 5(e,c): $Z_5^1 - X_e^1 \leq 0$, $Z_5^1 - X_c^1 \leq 0$,
 $Z_5^2 - X_e^2 \leq 0$, $Z_5^2 - X_c^2 \leq 0$

For edge 6(a,b): $Z_6^1 - X_a^1 \leq 0$, $Z_6^1 - X_b^1 \leq 0$,
 $Z_6^2 - X_a^2 \leq 0$, $Z_6^2 - X_b^2 \leq 0$

For edge 7(b,c): $Z_7^1 - X_b^1 \leq 0$, $Z_7^1 - X_c^1 \leq 0$,
 $Z_7^2 - X_b^2 \leq 0$, $Z_7^2 - X_c^2 \leq 0$

ii. 14 constraints of type ii:

For edge 1(f,g): $Z_1^1 - X_f^1 - X_g^1 \geq -1$, $Z_1^2 - X_f^2 - X_g^2 \geq -1$

For edge 2(f,e): $Z_2^1 - X_f^1 - X_e^1 \geq -1$, $Z_2^2 - X_f^2 - X_e^2 \geq -1$

For edge 3(g,e): $Z_3^1 - X_g^1 - X_e^1 \geq -1$, $Z_3^2 - X_g^2 - X_e^2 \geq -1$

For edge 4(e,d): $Z_4^1 - X_e^1 - X_d^1 \geq -1$, $Z_4^2 - X_e^2 - X_d^2 \geq -1$

For edge 5(e,c): $Z_5^1 - X_e^1 - X_c^1 \geq -1$, $Z_5^2 - X_e^2 - X_c^2 \geq -1$

For edge 6(a,b): $Z_6^1 - X_a^1 - X_b^1 \geq -1$, $Z_6^2 - X_a^2 - X_b^2 \geq -1$

For edge 7(b,c): $Z_7^1 - X_b^1 - X_c^1 \geq -1$, $Z_7^2 - X_b^2 - X_c^2 \geq -1$

iii. 7 constraints of type iii:

For vertex a: $X_a^1 + X_a^2 = 1$

For vertex b: $X_b^1 + X_b^2 = 1$

For vertex c: $X_c^1 + X_c^2 = 1$

For vertex d: $X_d^1 + X_d^2 = 1$

For vertex e: $X_e^1 + X_e^2 = 1$

For vertex f: $X_f^1 + X_f^2 = 1$

For vertex g: $X_g^1 + X_g^2 = 1$

iv. 2 constraints of type iv:

For cluster 1: $X_a^1 + X_b^1 + X_c^1 + X_d^1 + X_e^1 + X_f^1 + X_g^1 \geq 1$

For cluster 2: $X_a^2 + X_b^2 + X_c^2 + X_d^2 + X_e^2 + X_f^2 + X_g^2 \geq 1$

3. High level description of the project

The project consists of three parts:

- Solving the max K-clustering problem by interfacing the CPLEX library. Then outputting the clustering results and statistics of the clustered solutions.
- Outputting the clustering results as an XGMML (graphic XML) file to allow further analysis and visualization.
- (Bonus) Testing the method on real data using the Cytoscape environment.

3.1. General Requirements

The main program of the project will get 4 arguments from the user:

- i. Input folder (relative path + slash)
- ii. Output folder (relative path + slash)
- iii. Lower bound for the number of clusters: L
- iv. Upper bound for the number of clusters: U

Notice that the Linux operating system expects forward slash '/'.

Input

The program shall open the files *nodes* and *edges* (see all files format below) in *input folder*, and read the network from them. Maximum line length in a file is 500.

The network is defined as in exercise 2:

The network is undirected. Ids are automatically assigned to vertices and edges by the program. Each vertex holds a name, degree, and a doubly linked list of edges. Each edge holds a non-negative double weight. The same validity checks as in exercise 2 apply in the project, including safe programming checks.

The following adjustments to exercise 2 are to be made:

There is no need to support deletion of vertices or edges (although you may leave the corresponding functions in the code, if you wish to). The Id of a vertex is defined according to the order in which it was fed to the network, i.e. the first vertex will be assigned Id = 1, the second Id = 2 and so on (notice, the definition is different than exercise 2, in which Ids started at 0). If a second vertex arrives with a name that already exists in the network, a warning message will be issued to the screen and the new vertex will not be added to the network. If a self-loop or duplicated edge arrives, a warning message will be issued to the screen and it will not be added to the network. Since the network is undirected an edge (u,v) will be considered as duplicated, if either (u,v) or (v,u) is already in the network.

For those additional checks you may reach $O(\text{num_of_vertices})$ for adding a new vertex and $O(\min(r_1, r_2))$, where r_1, r_2 are the degrees of the first and second vertices of the edge, for adding a new edge.

Terminate the program only if no other course of action exists: An input/output file cannot be found or opened, an OS operation failed (as malloc, printf, scanf, etc) or cplex failed.

You may and should alter necessary code sections from exercise 2, notice that the altered exercise 2 is also part of the project, and will be graded as well.

The program will solve the max K-clustering problem using k in the given range (from lower bound L to the upper bound U, inclusive).

Output

1 file *results*:

- The score of each $\langle k \rangle$ -clustering, in the range from L to U , inclusive.
- For the clustering solution of $k=U$:
 - The clustered network
 - Average weight of an edge within the clusters
 - Average weight of an edge between the clusters
 - For each cluster in the network (clusters organized by size, from biggest cluster to smallest, in terms of the number of vertices residing in it):
 - The cluster's score (the sum of weights over the edges residing in it).
 - The cluster's diameter (diameter of a graph or sub graph is defined as the "longest shortest path" traversing its vertices)

$(U - L + 1)$ files with the name $\langle k \rangle$ *clustering_solution.xgmml*:

- For each k , the clustering solutions will be stored in $\langle k \rangle$ -*clustering_solution* file in xgmml format, readable by Cytoscape.

1 file *best_clusters.xgmml*:

- For $k=U$ the 5 biggest clusters will be stored in *best_clusters* file in xgmml format, readable by Cytoscape. Cluster size is the number of vertices residing in it.

All these files will be stored in *Output folder*.

Pay close attention to the format and make sure you match the examples files given.

Files Formats

Input files:

3.1.1. *nodes* file

protein: $\langle \text{vertex_name} \rangle$

...

protein: $\langle \text{vertex_name} \rangle$

3.1.2. *edges* file:

interaction: $\langle \text{first_vertex_name} \rangle$ - $\langle \text{second_vertex_name} \rangle$ $\langle \text{weight} \rangle$

...

interaction: $\langle \text{first_vertex_name} \rangle$ - $\langle \text{second_vertex_name} \rangle$ $\langle \text{weight} \rangle$

Output files:

3.1.3. *results* file:

Clustering with $k=\langle L \rangle$: $\langle \text{score} \rangle$

...

...

...

...

Clustering with $k=\langle U \rangle$: $\langle \text{score} \rangle$

$\langle \backslash n \rangle$

The clustered network for $k=U$:

```

<num_of_vertices> vertices:
<id>: <vertex_name> <cluster_no>
<id>: <vertex_name> <cluster_no>
<num_of_edges> edges:
<id>: <first_vertex_name>-<second_vertex_name> <weight>
<id>: <first_vertex_name>-<second_vertex_name> <weight>
<\n>
Clustering statistics for <U>:
Average weight of an edge within clusters: <weight>
Average weight of an edge between clusters: <weight>
Cluster <cluster_no>: score - <cluster_score> diameter - <longest shortest
path>
Cluster <cluster_no>: score - <cluster_score> diameter - <longest shortest
path>

```

Cluster number is in the range $1 \dots K$.
Vertex number is in the range $1 \dots \text{num_of_vertices}$.
All scores will be written in x.xxx format.

3.1.4. *<k>_clustering_solution.xgmml* files:

File format is xgmml, described in detail in section 4.2.

3.1.5. *best_clusters.xgmml* file:

File format is xgmml, described in detail in section 4.2.

Example input and output files are available on the course site at virtual. Use them to validate your format. Notice that the cluster number is arbitrarily assigned by cplex, and so will not necessarily match the output examples.

In some cases more than one optimal solution is possible, thus not matching the output examples for those cases. A small test case with only one optimal solution is also provided for you on the course site at virtual, to allow simple validation.

Vertices and edges are to be written to the files in the same order of arrival, excluding self-loops and duplicated vertices and edges.

4. Implementation

4.1. Solving max K-Clustering via CPLEX

IBM ILOG CPLEX Optimization Studio (or simply: cplex) is an optimization software package for integer linear programs (and beyond).

4.1.1. Integrating into existing code

You will be given the files cluster.h and cluster.c. The given module initializes the cplex environment, sets screen as an output for cplex errors and notifications, and sets parameters for cplex. It then calls for a mixed integer program optimization and frees the environment. You will add into this module the following functionalities:

- i. You will translate the problem into an ILP, and use the cplex interface function CPXcopylp to create the corresponding cplex problem instance.
- ii. You will define the variables as binary using the CPXchgctype function.

- iii. Use CPXwriteprob (the call to CPXwriteprob is already written in cluster.c) to output the problem in lp format. The name of the file, passed as a parameter to CPXwriteprob should be <k>.lp according to the current k.
- iv. After the problem has been solved by cplex you will read the solution using CPXsolution to get both the solution's value and the variable assignment.
- v. Return the solution to the calling routine.

You will need to change the cluster.h file as well, to receive the data and return the obtained solution, in the best way you see fit. You may change functions' declaration, add any definition, global/local variables etc'.

4.1.2. Checking assignment value

Due to numerical problems with the exact constitution of real values (even upon binary variables), you might get a value of “almost” 1 instead of an assignment of 1 from the cplex solution assignment array.

Thus if you want to check whether v was assigned 1, you should use the following precompiled command:

```
#define IS_VALUE_1(X) ((1 - X) < 0.00001)
```

For more details on optimization problems and cplex interface see the lecture presentation on virtual. For the online tutorial of cplex, see:

<http://yalma.fime.uanl.mx/cplex11-manual/refcallablelibrary/html/>. Scroll down to 'functions' at the bottom left side window to find all of cplex interface functions mentioned above.

4.1.3. Tips:

- Start by reading the description of the relevant functions in the tutorial.
- Make sure you understand how the parameters to all the functions should be built and how the problem would appear in memory. You have a file on the virtual to help you follow a dry run for the example given here.
- Only then plan what is the best way to line up the variables and constraints in memory, in the form that would allow you easy and simple access, both to create the data for the problem and to read it after.
- Start with small examples and work your way up.
- Errors while running are usually hard to debug through an interface. In case of a crash inside the cplex environment try to make sure the data looks as you intended it to look when you planned the memory allocations and structure. You may also want to try and return to the smaller examples to track the problem. Never the less, these problems are bound to happen and locating them is part of the exercise.
- The function k_cluster outputs an .lp file of the problem. An .lp file is a description for the problem in cplex format, similar to what was shown in class. You can test yourself by opening the file (using a text editor, such as notepad++) and making sure the problem is fed to cplex as you intended it to. You can also open cplex by command line as shown in class and feed the lp file directly to the cplex engine to view the results.
- As always, make sure you free any memory allocated by you. Take special care of clean and easy to understand code, using modularity and commentary – as you can see, someone might need to handle maintenance after you have long finished the work.

4.1.4. Running example: Representation in memory

A full example of a problem representation in cplex memory can be found in the presentation. The following example refers only to the memory representation of the binary variable Z_1^1 .

Let us assume that the integer variable Z_1_1 holds the index that represents Z_1^1 in the problem, and that Z_1_1 values in `matind` and `matval` arrays begin at location x . `matbeg[z_1_1] = x` will indicate the beginning location of z_1_1 in `matind` and `matval`, and `matcnt[z_1_1] = 3` will indicate Z_1^1 participation in three rows:

$$Z_1^1 - X_f^1 \leq 0, \quad Z_1^1 - X_g^1 \leq 0 \quad \text{and} \quad Z_1^1 - X_f^1 - X_g^1 \geq -1.$$

Z_1^1 participates in the objective function, and its value is W_1 ,

A fact represented in the `obj` array as `obj[z_1_1] = 0.3`.

Z_1^1 participates in the three constraints above. Assuming that the constraints' index is `c1`, `c2`, and `c3` respectively:

`matind[x] = c1`

`matval[x] = 1`

`matind[x+1] = c2`

`matval[x+1] = 1`

`matind[x+2] = c3`

`matval[x+2] = 1`

$$1 * Z_1^1 - 1 * X_g^1 \leq 0$$

$$\text{MAX } [(0.3 * Z_1^1) + \dots]$$

For the three constraints the sense array and rhs would indicate their type and right-side-value:

`sense[c1] = 'L'`

`rhs[c1] = 0`

`sense[c2] = 'L'`

`rhs[c2] = 0`

`sense[c3] = 'G'`

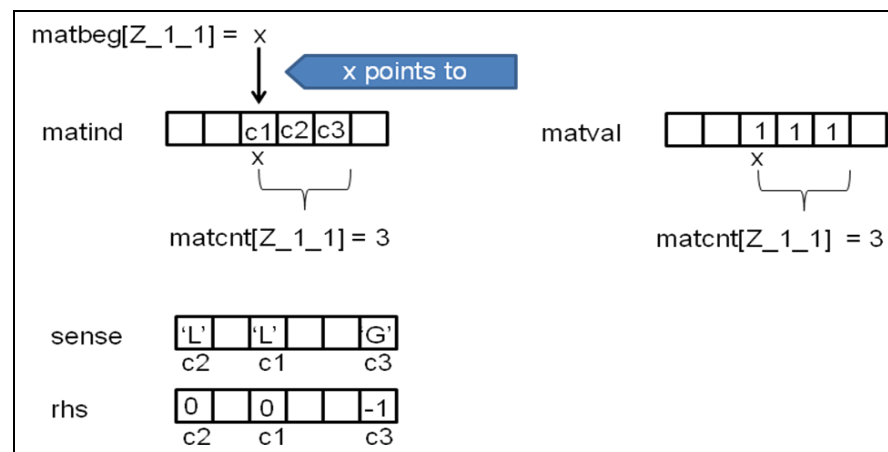
`rhs[c3] = -1`

$$Z_1^1 - X_f^1 \leq 0$$

$$Z_1^1 - X_g^1 \leq 0$$

$$Z_1^1 - X_f^1 - X_g^1 \geq -1$$

In memory:



The output of the module should include:

- File *results* containing the score of each $\langle k \rangle$ -clustering in the range from L to U , the clustered network for the $k=U$ solution and statistics of the solution. See format above in section 3.1.3.


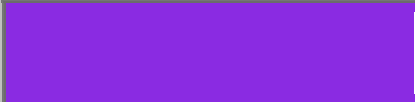




4.2. Producing an XGMML (graphic XML) representation of the K-Clustering results.

In this module you will take each K-clustering calculated for the network and translate its representation into XGMML format, then save it in .xgmml file. XGMML (the eXtensible Graph Markup and Modeling Language) is an XML application based on GML which is used for graph description. The design goals for XGMML, as well as for XML, are to emphasize simplicity, generality, and usability over the Internet.

4.2.1. XGMML output

The output of this module is xgmml files, each containing a specific `<k>_clustering_solution`. The division into clusters would be graphically presented using vertices color: The clusters will be sorted by size, and for each vertex a color is chosen according to its cluster size. For example all vertices in the biggest cluster would be colored in #00FFFF, all vertices in the second biggest cluster will be colored in #0000FF and so on. From the 10th sized cluster and below the color is the same and is #C0C0C0. If few clusters share the same size, order those clusters arbitrarily in their shared range, and assign a different color to each.

Table 1: colors to be assigned to vertices according to their cluster size

#1 cluster	#00FFFF	Aqua	
#2 cluster	#0000FF	Blue	
#3 cluster	#8A2BE2	Blueviolet	
#4 cluster	#A52A2A	Brown	
#5 cluster	#7FFF00	Chartreuse	
#6 cluster	#006400	Darkgreen	
#7 cluster	#FFD700	Gold	
#8 cluster	#FF69B4	Hotpink	
#9 cluster	#FF4500	Orangered	

#10 cluster and above	#C0C0C0	Silver	
-----------------------	---------	--------	--

The output of the module should include:

- $(U - L + 1)$ files with the names $\langle k \rangle_clustering_solution.xgmml$ respectively.
- For $\langle U \rangle$ -clustering, the file *best_clusters.xgmml* holding only the 5 biggest clusters, in terms of number of vertices residing in them.

4.2.2. Creating the XGMML file

The creation of an XGMML document is identical to that of an XML, and all available objects, materials and examples for XML can be used for XGMML. In your implementation you will use the well documented library of libxml2. To this end you should add this line to your module:

```
#include <libxml/tree.h>
```

As shown in class this enables your program to use the c interface of libxml.

Online msdn tutorial:

<http://xmlsoft.org/html/index.html>

4.2.2.1. The .xgmml File format:

Using the XML objects you will create an XML file of the following structure:

- graph
 - node 1
 - graphics
 - node 2
 - graphics
 - ...
 - node n
 - graphics
 - edge 1
 - graphics
 - edge 2
 - graphics
 - ...
 - edge n
 - graphics

After preparing the xml with the xml elements of the library libxml2, save the file with an .xgmml extension.

Refer to appendix A for detailed structure and attributes, and see also the example in section 4.2.3. The xgmml file must correlate exactly to the format given, i.e. maintain the same order of attributes and labels format. Your xgmml file should pass the diff – w check, for the “only one optimal solution” small test case provided in virtual.

4.2.2.2. Creating the next XGMML file

Notice that the content of each xgmml file differs in fact only in the label of the graph, indicating the $\langle k \rangle$ used, and the colors of the vertices, which reflect the clustering. You are to exploit that fact and the use of xml object, so that a basic preparation of the

files would take place only once, for the first k , and for each next $\langle k \rangle_clustering_solution$ file only 4 operations will take place:

1. Changing the property "label" for the graph element
2. Decide which cluster gets which color, according to their sizes.
3. Changing the colors of the vertices.
4. Saving the altered xml document object under the new name.

Changing the colors of the vertices is exactly like handling a linked list: you start from the first element child of graph, that is the first edge, and you continue along the edges elements by asking for the next element sibling.

The changing of color will be done by setting the property "fill" again:

```
xmlSetProp(pGraphics, BAD_CAST "fill", BAD_CAST new_color);
```

The cost of this implementation is holding objects in memory for a longer time, such as the pointer to the xml document object, the benefit however is to avoid going over all the elements and creating them again, in particular the edge elements, which can sum up to $|V|^2$.

4.2.2.3. Creating the best_clusters XGMML file

The best clusters file is based on the k -clustering solution that was produced by $\langle U \rangle$ -clustering.

Only vertices residing in the 5 biggest clusters, in terms of number of vertices, of this solution appear in the best_clusters file, as well as the edges between this set of vertices (notice that an edge can be either inside a big cluster or between two big clusters). Vertices colors are determined in the same fashion as before: vertices residing in the biggest cluster among the 5 are colored with #00FFFF, vertices residing in 2nd biggest cluster are colored with #0000FF and so on. If U is under 5, all of the U clusters will be in the best_clusters.xgmml. If you cannot reduce to 5 biggest clusters (for example, the 5th and 6th cluster have the same size) reduce the extra clusters according to the cluster's weight (sum of the edges residing in it), preferring heavier clusters over lighter ones. If the clusters measure exactly the same in both criteria, reduce them arbitrarily. Either way, the best_clusters file will not have more than 5 clusters in it.

As the best clusters file differs in vertices and edges from the rest of the files, it can be either created new, or from the previous xml document object, by removing unnecessary nodes and edges. This is left for your discretion.

4.2.2.4. Tips:

- Xml files can be easily viewed in most browsers, by typing the path in the address line. They can be easily manipulated using notepad++ (see more details on notepad++ in lecture 2 presentation).
- A basic example for the use of libxml can be found online <http://xmlsoft.org/examples/tree2.c>
More examples are in the second project lecture.

4.2.3. Running example - Visualization through xgmml

Running example showing two vertices and two edges in a 2-clustering. Refer to appendix B for visualization options of the file.

```
<?xml version='1.0'?>
<graph xmlns:cy='http://www.cytoscape.org' label='2_clustering_solution'
  xmlns:dc='http://purl.org/dc/elements/1.1/'
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:xlink='http://www.w3.org/1999/xlink'
  xmlns='http://www.cs.rpi.edu/XGMML' directed='0'>
  <node label="a" id="1">
    <graphics type="ELLIPSE" fill="# FF69B4" width="5"
    cy:nodeLabel="a" cy:borderLineType="solid"/>
  </node>
  <node label="b" id="2">
    <graphics type="ELLIPSE" fill="#00FFFF" width="5"
    cy:nodeLabel="b" cy:borderLineType="solid"/>
  </node>
  .
  .
  .
  <edge label="a (interaction detected) b" source="1" target="2">
    <graphics cy:edgeLabel="weight = 0.980"/>
  </edge>
  <edge label="c (interaction detected) b" source="3" target="2">
    <graphics cy:edgeLabel="weight = 0.200"/>
  </edge>
  .
  .
  .
</graph>
```

To this structure you may add any element/attribute you find useful. However, after any alteration, make sure your output can be read and visualize by the visualization tool as described in Appendix B.

4.3. (Bonus, 5 points) Using Cytoscape and BINGO to visualize and check the biological relevance of the clusters

On the course site at virtual you will find yeast.zip. It contains one *nodes* file and three *edges* files (the *nodes* file is common to all the *edges* files). One of the *edges* files corresponds to a real protein-protein interactions (PPI) network, while the other two are just random networks with the same node degrees. You will test the K-clustering method on these networks by running the K-clustering program with the same range of K for all 3 *edges*. To your manual submission you should add the form in yeast.zip with the answers to the following questions:

1. For each of the 3 *edges* files in yeast.zip: specify the score you received with 4-clustering (run with L=4 and U=4, use .001 as the precision).
2. The true yeast PPI network is the one with the highest 4-clustering score among the three networks (you should see 5-10 points difference in the score). Specify which network is the true PPI network (i.e. write the name of the *edges* file with highest 4-clustering value.)
3. For the true PPI network alone: run again the edges and nodes files with 10-clustering (run with L=10 and U=10, use .001 as the precision). Take the output file of the second module, *best_clusters.xgmml*, containing the 5 largest clusters. Use the BiNGO plugin in Cytoscape to compute the functions associated with each cluster (see Appendix B and C). For each cluster, specify the most significant associated function with its corrected p-value (if none exists, specify that no function was found).

See readme.txt in yeast.zip for more details of the network and time estimation.

5. Project Material

- cluster.zip: containing cluster.h and cluster.c to integrate with module 1, both files may be altered by you.
- makefile skeleton
- Input files and output files containing running examples of small scale networks. Refer to the readme.txt of each example for details of the example network.
- test.zip: input files containing a test network. The output files of this test case are to be submitted in the **results** folder. Refer to the readme.txt for details of the test network and the K range for the submission.
- yeast.zip: data for bonus part analyzing yeast protein networks and a sheet to fill and hand as the manual submission part of the bonus.

6. Error Handling

Your code should handle all possible errors that may occur (e.g. wrong/missing arguments, infeasible range for clustering, memory allocation problems, safe programming). Files' names can be either relative or absolute and can be invalid (the file/path might not exist or could not be opened). Issue a warning whenever a problem occurs, and follow the guideline to continue with the run as long as it is possible. In case you must terminate the program free all allocated memory and issue an appropriate message before terminating.

7. Submission guidelines

Please check the course webpage for updates and Q&A.
Any question should be posted to the course forum.

7.1. Files

The submitted assignment must be in the specified format. Otherwise, it will not pass the automatic checks.

- Create a directory `~/soft-proj-12b/ex3`.
- This directory should contain the `partners.txt` file and the following subdirectories:
 - code** - Containing all code files (`.h` and `.c`) for your project and the makefile.
 - results** – Containing all the output files (`<k>_clustering_solution.xgmml`, `best_clusters.xgmml` and `results`) for the test.zip folder.
 - bonus** – If you have done the bonus part include here the output files (`<k>_clustering_solution.xgmml`, `best_clusters.xgmml` and `results`) for the network you estimated as real with the run of `k=10`.
- The file **partners.txt** should contain the following information:
 - Full Name: your-full-name
 - Id No 1: your-id
 - User Name 1: your-user-name
 - Id No 2: partner-id
 - User Name 2: partner-user-name
 - Assignment No: 3Every partner should have a different `partner.txt` file, containing his details as partner no 1.
- Although the submission is in pairs, every student must have all the exercise files under his home directory as described above. The exercise files of both partners must be identical.
- Set permissions using
 - `chmod 755 ~`
 - `chmod -R 755 ~/soft-proj-12b`

7.2. Hard copy submission

The submission should include printouts of the code files (all `.c` `.h` and makefile), and the name, user-name, and id-number – of both partners. One hard copy should be made for each pair. For the bonus part the submission should also include the sheet in `yeast.zip`, filled with the requested information of this part.

Submit to teacher assistant mail-box: 'dana silverbush', second floor on Schreiber, near the secretaries.

7.3. Coding

Your code should be gracefully parted into files and functions. The design of the program, including interfaces, global variables, functions' declaration (including changing the declarations given in `cluster.c` and `cluster.h`) and partition into modules is entirely up to you, and is graded. You should aim for modularity, reuse of code, clarity, and logical partition. In your implementation, also pay careful attention for use of constant values and use of memory. Do not forget to free any memory you allocated.

As a general rule, if possible you should aim to allocate only necessary memory and free objects (memory and files) as soon as it is possible. Unless instructed specifically (as in section 4.2.2.2), use your better judgment for balancing between clean and manageable code, efficient algorithm, and holding resources.

Code should be commented at critical points in the code and at function declarations.

In order to prevent lines from wrapping in your printouts- please avoid long code.

7.4. Compilation

A skeleton makefile is provided at the website, compiling and linking cluster.* with cplex interface. Notice that at first the linker will fail with **make all**, as there is no main() function declared in the files.

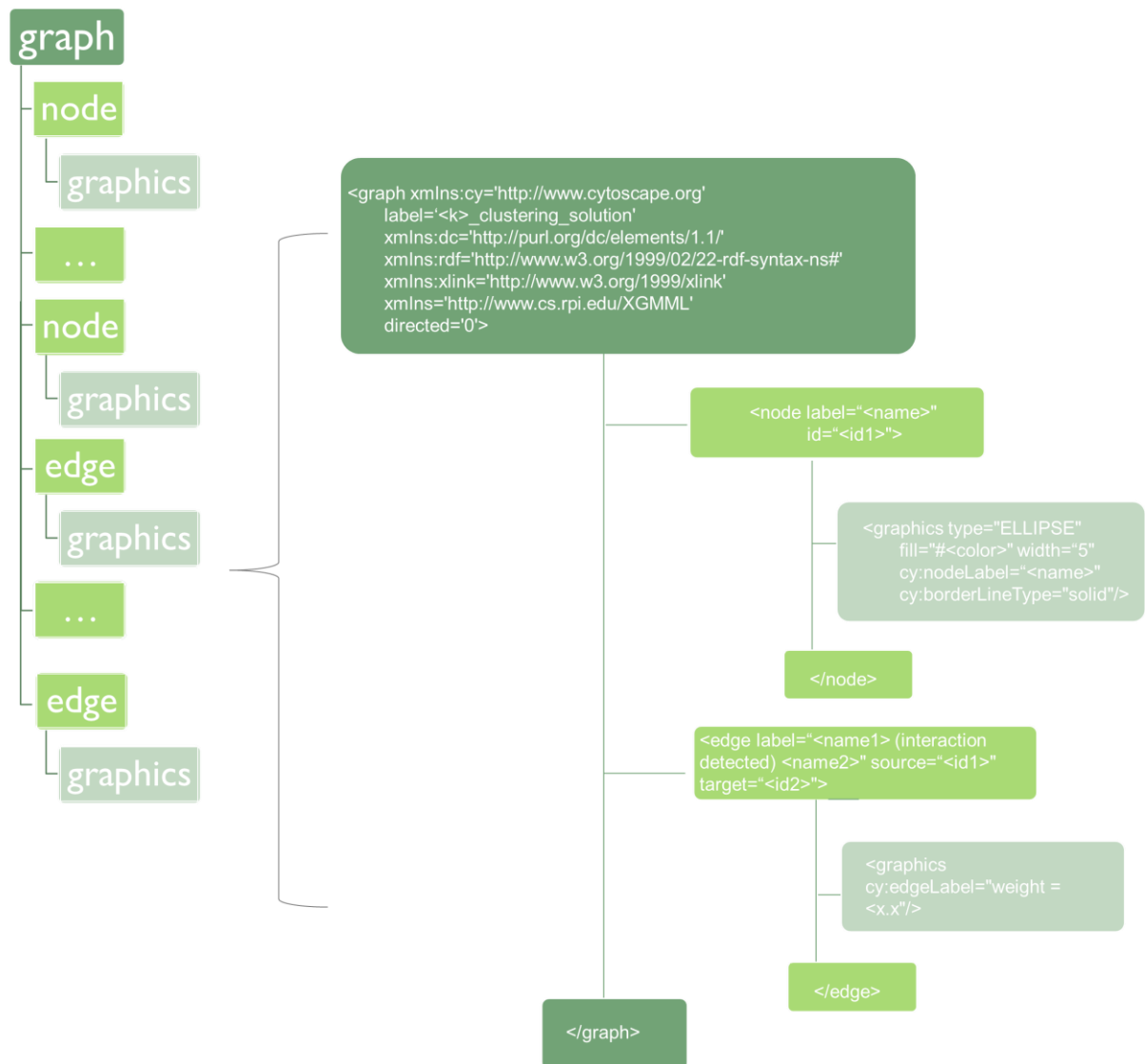
You will add your files to the makefile. Use the flags provided for you to link to libxml. Your project should pass the compilation test with no errors or warnings, which will be performed by running the **make all** command in a UNIX terminal window using your updated and submitted makefile.

Good Luck!

Appendix A - xgmml file format

To the left is the structure with three levels, to the right are the attributes of each level

Appendix figure 1 XGMML structure and attributes



Weight will be written in x.xxx format. Examples files follow that rule.

Specification for each individual `<k>_clustering_solution` file:

1. Graph label is changed to **<k>_clustering_solution**
2. Vertices colors are according to their new clustering

Specification for `best_clusters` file:

1. Graph label is changed to **best_clusters**
2. Only vertices residing in the 5 biggest clusters appear
3. As before, vertices' color determined according to their cluster's size.

Appendix B – Visualize network using cytoscape

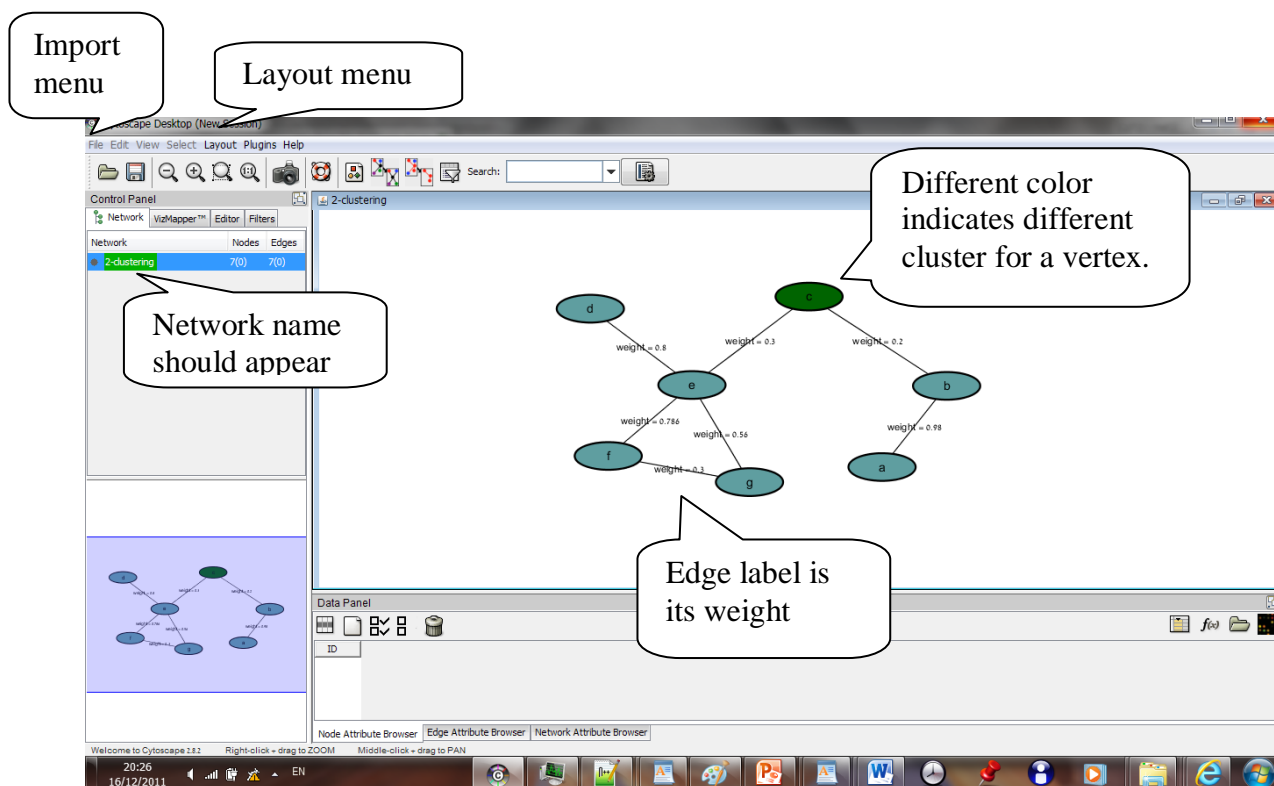
The network visual representation would be through the graphical software Cytoscape. Cytoscape is an open source software platform for visualizing molecular interaction networks. Cytoscape is installed on the university computers (open by writing **cytoscape** in the command line), and you can download and use it on your personal computer for free through the website: <http://www.cytoscape.org/>.

In order to view your clustered network on Cytoscape follow these steps:

- i. Launch cytoscape.
- ii. Click menu File->Import->Network (Multiple File Types)
- iii. Choose the 'local' option
- iv. Select your .xgmml file

Your network might appear as a single vertex. This impression is due to the default layout. You can change the layout through Layout->Cytoscape Layouts, and then choose any layout. 'Circular Layout' or 'Spring Embedded' options are recommended. If you wish to import another file you are advised to first close the current network. You can do it by standing on the network tab and choosing 'Destroy View' followed by 'Destroy Network'. Then follow the instructions again from step ii.

Appendix figure 2 Import a network into cytoscape



Appendix C – Finding the biological functions associated with clusters

The functionality of Cytoscape can be extended by installing additional plugins (under its plugins directory). We use the BiNGO plugin to compute functions associated with gene clusters.

The BiNGO plugin can be freely downloaded from

<http://www.psb.ugent.be/cbd/papers/BiNGO/1>, but in most installations BiNGO was already downloaded when Cytoscape was installed (this is the case for most Schreiber computers). After it has been downloaded you can use the plugins menu in Cytoscape to install it. Click Plugins -> Manage Plugins -> Functional Enrichment and choose BiNGO v.2.44.

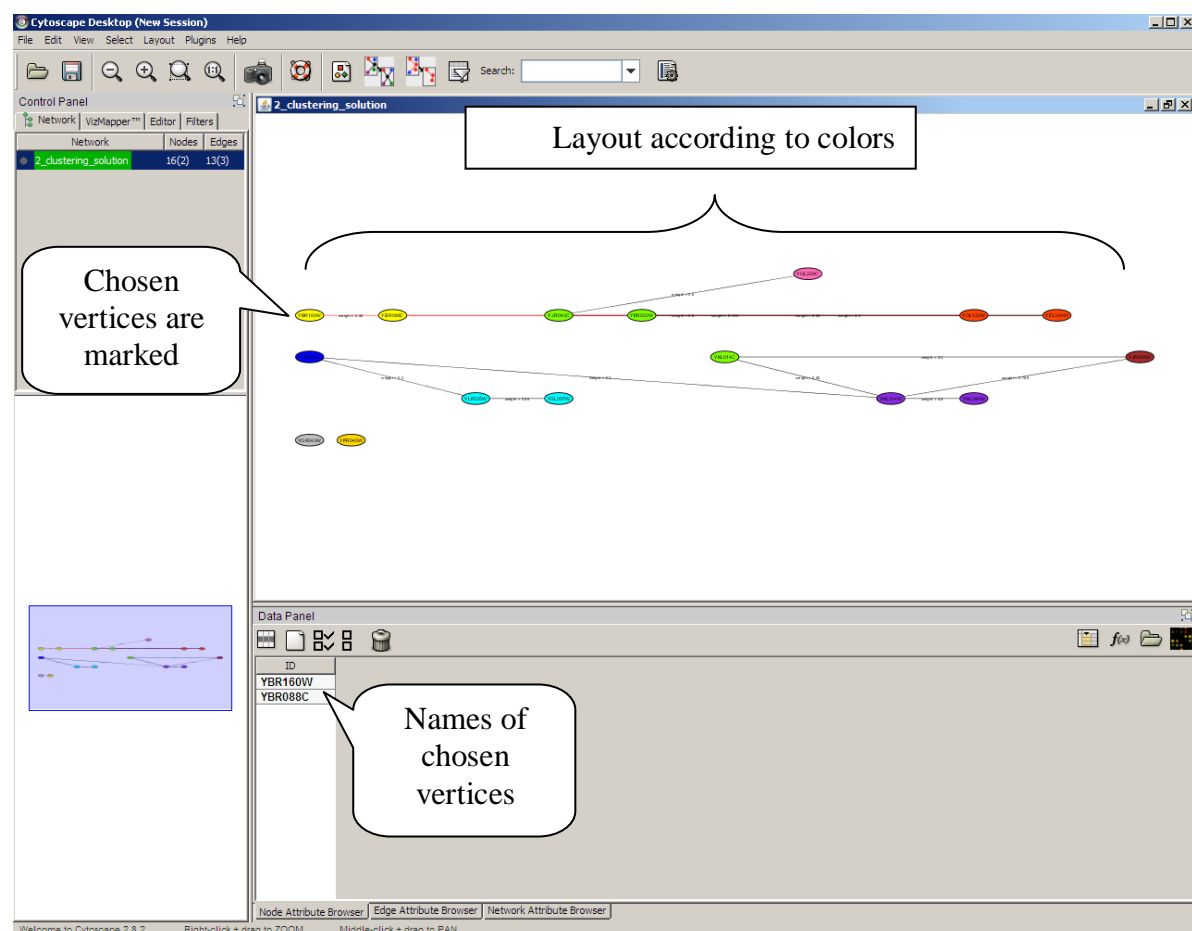
Now open your *best_clusters.xgmml* as described in appendix B, and choose the layout:

Cytoscape Layouts -> Group Attributes Layout -> vizmap:<k>_clustering_solution
NODE_FILL_COLOR.

This layout would separate the network into groups of different color.

For each of these groups, which are in fact the network's 5 top clusters, you would find the common biological function using BiNGO.

Appendix figure 3 Layout the top 5 clusters according to color



Run the BiNGO plugin by choosing from the main menu: Plugin→BiNGO 2.0. A window as in figure 4 will appear.

1. In the cluster name field write any name for the current cluster (figure 4 – BiNGO window).
2. Choose the option Paste Genes from Text (figure 4 – BiNGO window).

3. Choose the group of vertices for the current cluster in cytoscape window (draw a square around the group with your mouse). The chosen vertices would be marked in a

different color . The names of the vertices will appear at the bottom window in **Node Attribute Browser Tab**. (figure 3 – cytoscape window)

Now copy the names from the bottom window, using right click->copy.

5. Paste the names in the BiNGO search window just below the option Paste Genes from Text (All proceeding operations are in figure 4 – BiNGO window)

6. Choose the option **No Visualization**.

7. In the choose organism / annotation: Choose *Saccharomyces cerevisiae*. This is the scientific name of budding yeast.

8. Click the button Start BiNGO.

After the Start BiNGO button is pressed - a new BiNGO output window appears. The first row corresponds to the most significant function found (see Appendix figure 5 for an example). The name of the function appears in the description column. The p-value associated with this function in the one that appears in the column corr p-val (i.e. corrected p-value). Note that if no function was found to be associated with a cluster - its tab will be empty.

Note: You can ignore any warnings by pressing Yes.

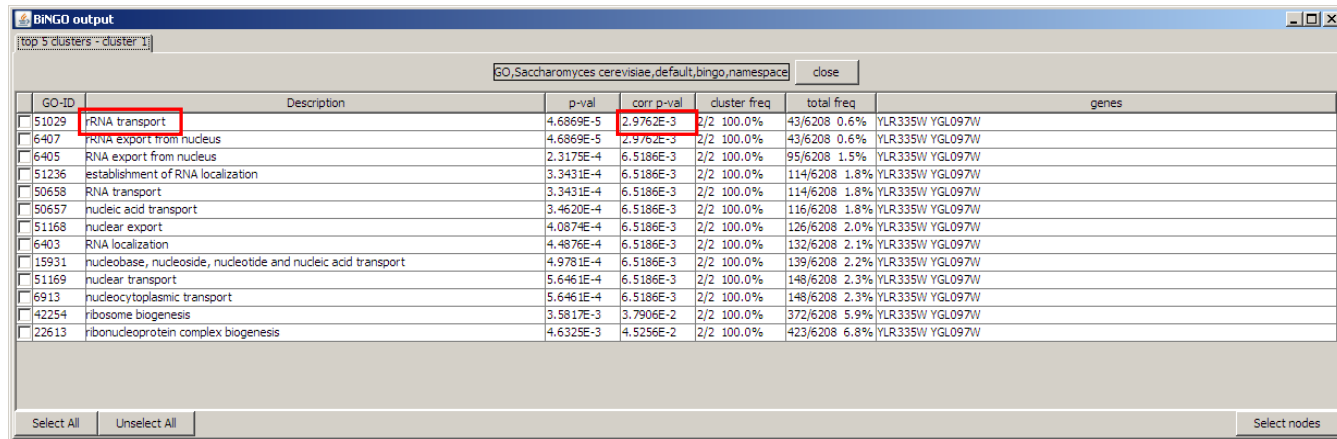
Appendix figure 4: BiNGO search window

Setting the parameters for plugin BiNGO (computing functions for gene clusters).

The image shows the BiNGO Settings window, which is used to configure the BiNGO plugin. The window has a title bar with the text "BiNGO Settings" and standard window controls. The main area is titled "BiNGO settings" and contains several sections of controls:

- Buttons:** "Save settings as default" and "Help" are located at the top right.
- Cluster name:** A text field containing "top 5 clusters - cluster 1".
- Input Methods:** Two checkboxes: "Get Cluster from Network" (unchecked) and "Paste Genes from Text" (checked).
- Gene List:** A text area containing the gene IDs "YLR405W" and "YGL097W".
- Assessment Options:** A section titled "Do you want to assess over- or underrepresentation:" with three checkboxes: "Overrepresentation" (checked), "Underrepresentation" (unchecked), and "No Visualization" (checked).
- Statistical Test:** A dropdown menu set to "Hypergeometric test".
- Multiple Testing Correction:** A dropdown menu set to "Benjamini & Hochberg False Discovery Rate (FDR) correction".
- Significance Level:** A text field set to "0.05".
- Categories to be Visualized:** A dropdown menu set to "Overrepresented categories after correction".
- Reference Set:** A dropdown menu set to "Use whole annotation as reference set".
- Ontology File:** A dropdown menu set to "GO_Biological_Process".
- Namespace:** A dropdown menu set to "biological_process".
- Organism/Annotation:** A dropdown menu set to "Saccharomyces cerevisiae".
- Evidence Codes:** A text field for "Discard the following evidence codes:" which is currently empty.
- Save Data:** A checkbox "Check box for saving Data" (unchecked) and a button "Save BiNGO Data file in:" followed by a text field.
- Start Button:** A large button at the bottom labeled "Start BiNGO".

Appendix figure 5: The output of the plugin BiNGO (computing functions for gene clusters).



GO, Saccharomyces cerevisiae, default, bingo_namespace close

GO-ID	Description	p-val	corr p-val	cluster freq	total freq	genes
51029	tRNA transport	4.6869E-5	2.9762E-3	2/2 100.0%	43/6208 0.6%	YLR335W YGL097W
6407	tRNA export from nucleus	4.6869E-5	2.9762E-3	2/2 100.0%	43/6208 0.6%	YLR335W YGL097W
6405	RNA export from nucleus	2.3175E-4	6.5186E-3	2/2 100.0%	95/6208 1.5%	YLR335W YGL097W
51236	establishment of RNA localization	3.3431E-4	6.5186E-3	2/2 100.0%	114/6208 1.8%	YLR335W YGL097W
50658	RNA transport	3.3431E-4	6.5186E-3	2/2 100.0%	114/6208 1.8%	YLR335W YGL097W
50657	nucleic acid transport	3.4620E-4	6.5186E-3	2/2 100.0%	116/6208 1.8%	YLR335W YGL097W
51168	nuclear export	4.0874E-4	6.5186E-3	2/2 100.0%	126/6208 2.0%	YLR335W YGL097W
6403	RNA localization	4.4876E-4	6.5186E-3	2/2 100.0%	132/6208 2.1%	YLR335W YGL097W
15931	nucleobase, nucleoside, nucleotide and nucleic acid transport	4.9781E-4	6.5186E-3	2/2 100.0%	139/6208 2.2%	YLR335W YGL097W
51169	nuclear transport	5.6461E-4	6.5186E-3	2/2 100.0%	148/6208 2.3%	YLR335W YGL097W
6913	nucleocytoplasmic transport	5.6461E-4	6.5186E-3	2/2 100.0%	148/6208 2.3%	YLR335W YGL097W
42254	ribosome biogenesis	3.5817E-3	3.7906E-2	2/2 100.0%	372/6208 5.9%	YLR335W YGL097W
22613	ribonucleoprotein complex biogenesis	4.6325E-3	4.5256E-2	2/2 100.0%	423/6208 6.8%	YLR335W YGL097W

Select All Unselect All Select nodes

Many functions are associated with the cluster in the picture. The most significant function is "tRNA transport", which has a corrected p-value of 2.9762E-3.

When you will input your own cluster, different names of biological processes than the example will appear under description field. Take the first row, corresponding to the best corrected p-value. This would be the answer for the current cluster in the bonus handout.