Software Project, summer 2013

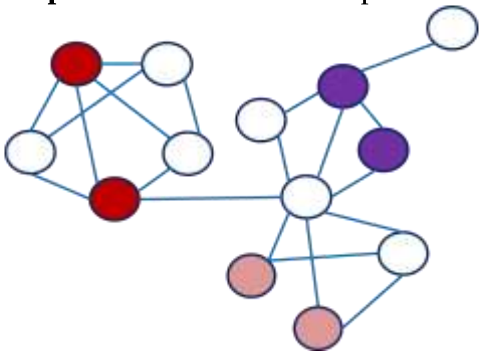
Due by 16.9.2013, 23:59

This is your final software project; it builds on the infrastructure developed in ex. 3, with the necessary alterations to the interface and body. In your final project you will cluster a biological network using the cluster editing formulation and detect functionality in the clusters according to your findings.

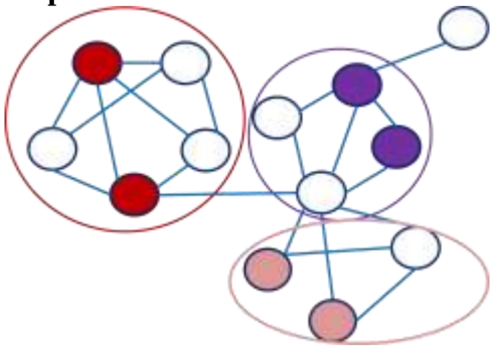
1. Introduction

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:

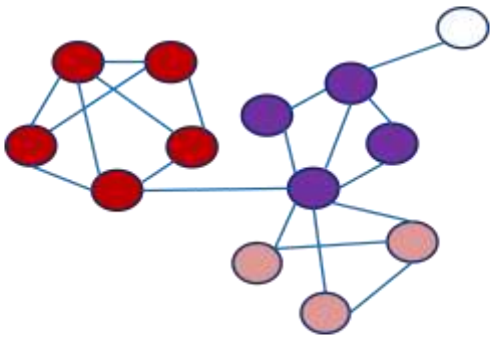
Step 1. Mark the network's proteins whose function is known:



Step 2. Cluster the network to maximize its similarity to a graph of cliques:



Step 3. For uncharacterized proteins, infer their functions according to their cluster members



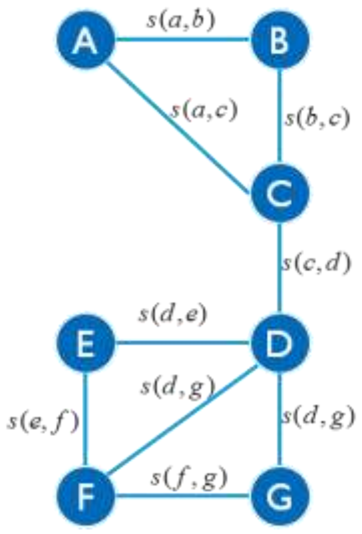
2. The Cluster Editing Problem

A *cluster graph* is a graph consisting of a vertex-disjoint union of cliques. The cluster editing problem for weighted graphs is defined as follows: Given a network *G=(V,E,s)*, with costs s(u,v) over all vertex pairs, find a set of edge modifications of minimum cost which results in a cluster graph.
Note that to create a cluster graph we may remove an existing edge *e* with the cost *s(e)* or add a non-existing edge ("ghost") g with a cost C (which will be given to the program as a parameter).

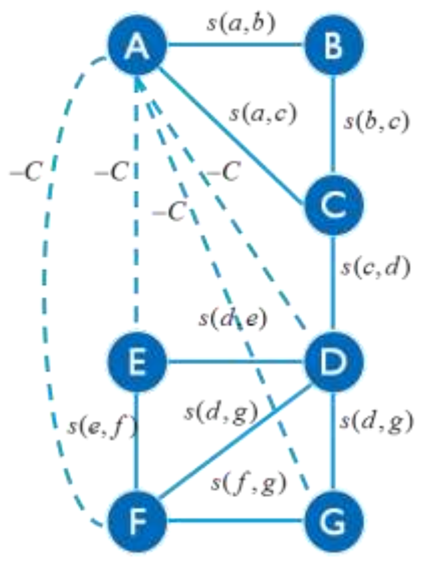
2.1. Example

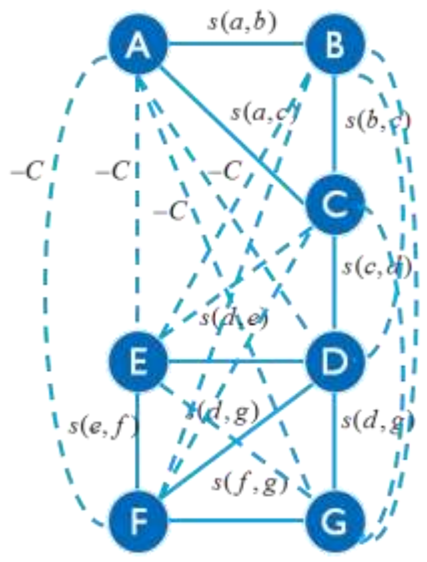
Given a graph with nodes: A..G

Edges: {A,B},{C,B},{C,A},{D,C},{E,D},{F,E},{F,D},{D,G},{F,G} (costs are denoted by $s(u,v)$)

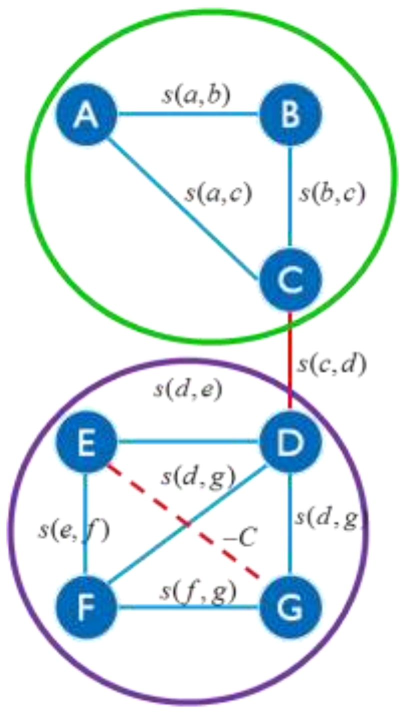


and the ghost edges:
{A,D},{A,E},{A,F},{A,G},{B,D},{B,E},{B,F},{B,G},{C,E},{C,F},{C,G},{E,G}
(costs are all equal to C, s(u,v) is assigned **minus** C).


Vertex A's ghosts


All ghosts

A possible modification of the graph is:



and its cost is $s(c,d)+C$.

2.2. Translating the Cluster Editing Problem to an Integer Linear Program

In your program you shall read a network and cluster it using an integer linear program formulation of the cluster editing problem. You will 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,… a linear program would look as follows:

maximize(minimize) cx

subject to $\left.\begin{array}{l} ax \le b \\ \cdots \\ \cdots \end{array}\right\}$ 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 [Grotschel and Wakabayashi. 1989; journal of Mathematical Programming, 45:52-96] and its details follow.

2.2.1 Binary Variables:

i. X_{ij} - an indication of whether edge (i,j) is in the solution. (i,j) may be a real edge or a ghost edge, so there are $\binom{V}{2}$ variables of this type.

- if $X_{ij} = 1$ then (i,j) is part of the solution. It may be a real edge which was not deleted, or a ghost edge that was added.

- if $X_{ij} = 0$ then (i,j) is not part of the solution. It may be a real edge which was deleted, or a ghost edge that was not added.

2.2.2 Objective:

Minimize the cost of modifications

$$\text{minimize} \quad \sum_{(i,j)\in E} s(i,j) - \sum_{1\le i<j\le|V|} s(i,j)\cdot X_{ij}$$

- Let us examine the contribution of edge (i,j) to the objective function:

	$X_{ij} = 1$ (edge is part of the solution)	$X_{ij} = 0$ (edge is not part of the solution)
$(i,j)\in E$ ((i,j) is a real edge)	s(i,j) – s(i,j) = 0	s(i,j) – 0 > 0
$(i,j)\notin E$ ((i,j) is a ghost edge)	-s(i,j) > 0 Remember that for each ghost edge s(i,j) = -C	0

2.2.3 Constraints:

i. For all $1 \le i < j < k \le |V|$:

$$+X_{ij} + X_{jk} - X_{ik} \le 1$$

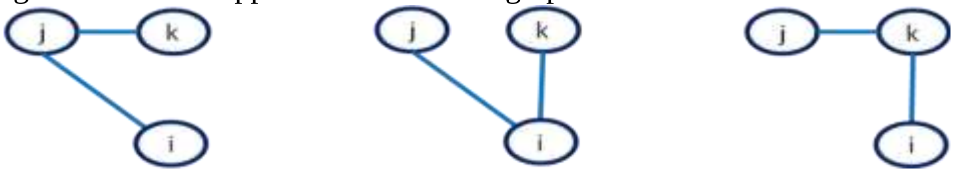
ii. For all $1 \le i < j < k \le |V|$:

$$+X_{ij} - X_{jk} + X_{ik} \le 1$$

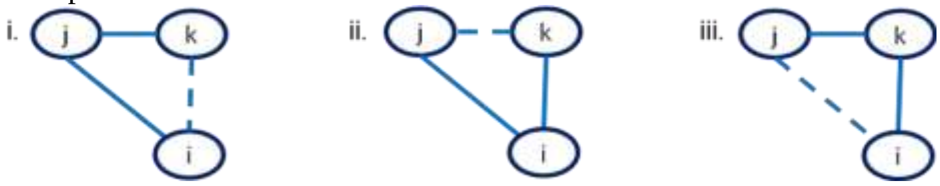
iii. For all $1 \le i < j < k \le |V|$:

$$-X_{ij} + X_{jk} + X_{ik} \le 1$$

- Taking 3 vertices: if two of their connecting edges exist in the solution graph, then the third edge should exists in the solution as well. That is, none of the following situations can appear in the solution graph:



Each constraint prevents one of the following situations respectively, by making sure that if the two solid edges are part of the solution, the dashed edge must be part of the solution as well:



Let us look at constraint i for example. If (i,j) and (j,k) are part of the solution then $X_{ij} = 1, X_{jk} = 1$ and the constraint would form $2 - X_{ik} \le 1$. To follow this constraint, X_{ik} must be assigned 1 as well, meaning it is part of the solution.

For more details see the presentation in class.

3. High level description of the project

The project consists of 4 parts:

- Solving the cluster editing problem by interfacing the CPLEX library.
- 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. Note that the maximum grade for a project is 100.

3.1. General Requirements

The main program of the project will get 3 arguments from the user:
i. Input folder (relative path + slash)
ii. Output folder (relative path + slash)
iii. C (minus C will be assigned as weight for all ghost edges)

Notice that the Linux operating system expects forward slash '/'. C must be a number between 0 (inclusive) and 1 (inclusive), or the program will print an error and exit.

3.1.1. Input

The program shall open the file *network* (see file format below) in *input folder*, and read the network from it. Maximum line length in a file is 500 characters.

The network is based on the infrastructure of Ex3. The graph is represented as an adjacency list, i.e. vertices are implemented as an array where every vertex contains its neighbors. Every vertex is represented by a structure containing the following fields:

a. The vertex name. The vertex name **has** to be unique among all vertices.
b. The vertex ID number. The ID number is an integer number between 0 and n-1, where n is number of vertices. The id number must be unique.
c. A linked-list (doubly or singly) of neighbors connected to the vertex. An edge is kept in both its vertices' arrays.
d. Any additional field you desire.

Every edge holds:

a. A non-negative weight between 0 (exclusive) and 1 (inclusive), formatted to x.xxx.
   For example, the command "add_edge 3 4 0.387621" will result in adding an edge to vertex 3's list and to vertex 4's list of weight 0.388. The formatted weight (0.388) is also the weight to be used for the ILP.
b. Any additional field you desire

Network requirements:

a. If a second vertex arrives with a name that already exists in the network, an error message will be issued to the screen and the new vertex will not be added to the network.
b. If a self-loop or duplicated edge arrives, an error 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.

You may reach O(#vertices) for adding a new vertex and O(min(r1,r2)), where r1, r2 are the degrees of the first and second vertices of the edge, for adding a new edge.

You may and should alter necessary code sections from exe 3, notice that the altered exercise is also part of the project, and will be graded as well.

3.1.2. Output

The network referenced in the output is the real network, without the ghost edges. Thus, when outputting the network, when calculating average weight of edge within and between clusters and when calculating cluster's score and diameter – only real edges are to be considered.

results:

- The score of the optimal cluster editing solution.
- For the clustering solution:
  - 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, organized according to cluster's identification (from 1 to number of clusters):
        - The cluster’s score (the sum of weights over the real 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, using only real edges)

The diameter of a cluster with isolated components (at least one vertex in the cluster is unreachable from another vertex) is infinite. The output for it should be "inf".

clustering_solution.xgmml:

- The clustering solutions will be stored in *clustering_solution.xgmml* file in xgmml format, readable by Cytoscape.

best_clusters.xgmml:

- The 5 biggest clusters will be stored in *best_clusters* file in xgmml format, readable by Cytoscape. The 5 biggest clusters are the clusters with identification 1 to 5 (see explanation of assigning identifications in section 4.1.1.iv).

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 file:

3.1.3. *network* file

The file will contain a list of commands of two types:

```
add_vertex <name>
add_edge <first_vertex_id>  <second_ vertex_id > <weight>
```

Output files:

3.1.4. *results* file:

```
Cluster Editing Optimal Solution
score: <score>
number of clusters: <k>
cost for ghost: <C>
<\n>
The clustered network:
<#vertices> vertices:
<vertex_name> <cluster_identification>
<vertex_name> <cluster_identification>
<#edges> edges:
<first_vertex_name>-<second_vertex_name> <weight>
<first_vertex_name>-<second_vertex_name> <weight>
```

```
<\n>
Clustering statistics for the <k> clusters:
Average weight of an edge within clusters: <weight>
Average weight of an edge between clusters: <weight>
Cluster <cluster_identification> score: <cluster_score> diameter: <longest
shortest path>
Cluster <cluster_identification> score: <cluster_score> diameter: <longest
shortest path>
…
```

All scores and weights will be written in x.xxx format.
score stands for minimal cluster editing cost. *k* stands for number of clusters in that optimal solution.

For the automatic checking, you must print the edges by the following order:

```
1.   For each vertex v from v.id=0 to v.id=n-1
     1.1.   Let "list_edges" be the array of v adjacent edges
                 i.  Print list_edges in the order of the edges' arrival. Do not print
                     an edge if it was already printed for a previous vertex.
```

Thus, an edge (v1,v2) shall be printed only once: during v1 pass if v1.id < v2.id or during v2 pass if v2.id < v1.id.

If the network does not contain any vertices, the program will issue an appropriate message and will not create any output files.
If the network does not contain any edges, the edges section (including "edges:") will not be printed. The rest of the output will be issued.
Remember: <#edges> means the number of edges. So, "<#edges> edges" would be interpreted as "5 edges", for example.

3.1.5. *clustering_solution.xgmml* files:

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

3.1.6. *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 on moodle. Use them to validate your format.
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, to allow simple validation using diff.

IMPORTANT:
Validate your exact format using the command diff –B –w (The parameters instruct to ignores spaces and new lines).

4. Implementation

4.1. Solving Cluster Editing 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_editing.h and cluster_editing.c. The given module initializes the cplex enviorement, 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. 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.

iv. Assign each cluster an identification. The cluster identification is an integer number between 1 and number of clusters, and is determined by the cluster's size, from biggest cluster to smallest, in terms of the number of vertices residing in it. If sizes are equal use cluster's score (taking into account only real edges) as a secondary measure, and random as third and last. Identification 1 will be assigned to the biggest cluster, 2 for the second biggest and so on.

v. Assign each vertex the cluster identification to which it belongs.

vi. Return the solution to the calling routine. The solution is the original graph, with the correct cluster identification for each vertex.

vii. You are responsible also for the code given to you, if you need to, secure it and make sure all memory was released.

You will need to change the cluster_editing.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 moodle. 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.

- 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 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.
- Notice! The ghost edges are part of this module and this module alone. None of the other modules should be affected by those edges.

The output of the module is:
- File *results* - See format above in section 3.1.2.

4.2. Producing an XGMML (graphic XML) representation of the clustering results.

In this module you will take the clustered network created by the cluster editing solution 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 XML 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 division into clusters would be graphically presented using vertices color: The cluster with identification 1 (the biggest cluster) will be assigned the color #0000FF, meaning all the vertices in the biggest cluster will be colored #0000FF. All vertices in the second biggest cluster will be colored in #8A2BE2 and so on. From the 10th sized cluster and below the color is the same and is #C0C0C0.

Table 1: *colors to be assigned to vertices according to their cluster identification*

#1 cluster	#0000FF	Blue	🟦
#2 cluster	#8A2BE2	Blueviolet	🟪
#3 cluster	#006400	Darkgreen	🟩

#4 cluster	#A52A2A	Brown	🟫
#5 cluster	#FF4500	Orangered	🟧
#6 cluster	#FF69B4	Hotpink	🟪
#7 cluster	#7FFF00	Chartreuse	🟩
#8 cluster	#00FFFF	Aqua	🟦
#9 cluster	#FFD700	Gold	🟨
#10 cluster and above	#C0C0C0	Silver	⬜

The output of the module should include:

- *clustering_solution.xgmml* containing the original network, i.e. all vertices and all of the real edges, where vertices are colored according to their cluster identification.
- *best_clusters.xgmml* holding only the 5 biggest clusters (identification 1 to 5), the vertices and edges residing in them, and the edges between them.

4.2.2. Creating the XGMML file via libxml

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 can 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.

The makefile provided for you already contains the flags required to link to xml when working on nova. To download and install libxml on your personal computer you can use this link http://xmlsoft.org/sources/. Download version 2.7.8 to keep compiled with the version on nova.

See code example in http://moodle.tau.ac.il/mod/resource/view.php?id=341323, slides 44-47.

Online msdn tutorial:
http://xmlsoft.org/html/index.html

4.2.3. Creating the XGMML by outputting content to FILE

Alternatively, you may choose to create the xml file by simply writing the content into it (using file functions such as fprintf), and saving the file with an extension

".xgmml". This option does not require downloading and compiling to libxml, or using any xml interface. It might be easier for those wishing to avoid installation of libxml and working with a new interface. Notice, your output files still have to match the exact format specified. Upload them to cytoscape, and also check them with diff.

4.2.3.1. The .xgmml File format:

The XML file is build upon the following structure:

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

Each node requires an id. This id is the vertex id assigned to the vertices upon arrival. It runs from 0 to #vertices-1.
Refer to appendix A for detailed structure and attributes. The xgmml file must correlate exactly to the format given, i.e. maintain the same order of attributes and labels format. The order of vertices and edges, including the edge's source and edge's target, should fit their order in the *results* file. **Your xgmml file should pass the diff –w –B check, for the ONE OPTIMAL SOLUTION small test case provided on moodle.**

4.2.3.2. Creating the best_clusters XGMML file

Create the best clusters file in a way which will avoid duplicate code as much as possible (it is in fact possible to avoid duplicated code almost entirely here). Remember that you are using libxml you can use the already prepared xml document, alter, remove, or add nodes and attributes, and save again.

Only vertices residing in the 5 biggest clusters will 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 `#0000FF`, vertices residing in 2nd biggest cluster are colored with `#8A2BE2` and so on. If the entire network was clustered to less than 5 clusters, all of its 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 and score), reduce them arbitrarily. Either way, the best_clusters file will not have more than 5 clusters in it.

4.2.3.3. 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.
- To free the XML document use:
  ```
  xmlFreeDoc(file);
  xmlCleanupParser();
  xmlMemoryDump();
  ```

4.2.4. Running example - Visualization through xgmml

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

```
<?xml version='1.0'?>
<graph label='clustering_solution'
          directed='0'
          xmlns:cy='http://www.cytoscape.org'
          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'>
          <node label="a" id="0">
             <graphics type="ELLIPSE" fill="#FF69B4" width="5"
           cy:nodeLabel="a" cy:borderLineType="solid"/>
          </node>
          <node label="b" id="1">
             <graphics type="ELLIPSE" fill="#00FFFF" width="5"
           cy:nodeLabel="b" cy:borderLineType="solid"/>
          </node>
          .
          .
          .
          <edge label="a (pp) b" source="0" target="1">
              <graphics cy:edgeLabel="weight = 0.980"/>
          </edge>
          <edge label="c (pp) b" source="1" target="2">
                    <graphics cy:edgeLabel="weight = 0.200"/>
          </edge>
          .
          .
```

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

On the course site on moodle you will find yeast.zip. It contains three *network* files containing the same vertices, and the same amount of edges. One of the network files corresponds to a real protein-protein interactions (PPI) network, while the other two are just random networks. You will test the cluster editing method on these networks by running the cluster editing program for all 3 *network* files. 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 *network* files in yeast.zip: specify the score you received (use .001 as the precision).
2. The true yeast PPI network is the one with the best score among the three networks (you should see ~15 points difference in the score). Specify which network is the true

PPI network (i.e. write the name of the *network* file with highest cluster editing score).
3. For the true PPI network alone: 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_editing.h and cluster_editing.c to integrate with module 1, both files may be altered by you.
    - makefile skeleton

  The first thing you should do after reading this document is open this folder, add function main to the project and compile using the makefile. It will be much easier to solve makefile problems at the beginning than after you have wrote some more code.
- Input files and output files containing running examples of small scale networks.
  If you are not sure about any format issues, use those to make sure you understood.
- 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.
- 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, 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).

Basic list of validity checks is provided for you:

a. Correct number of command line arguments is received
b. Path of input and output folders exists, and inside the input folder lies a file "network"
c. Each command in "network" is recognized.
d. Command length is no more than 500 characters.
e. Correct number of parameters for a command is received
f. Weight is a string composed of digits, and exactly none or one '.' (i.e, 0, 0.1, 1 are all valid, 2a and 1.3.3 are not valid)
g. The weight of a real edge is a positive number between 0 (exclusive) and 1 (inclusive).
h. The weight of a ghost is a positive number between 0 (inclusive) and 1 (inclusive)).

i. A new vertex has a unique name
j. A new edge is not a self loop or a duplicated edge
k. A new edge is connecting two existing vertices

In addition, throughout your program do not forget to check:

l. The return value of a function. You may excuse yourself from checking the return value of printf, fprints, sprint, perror and xmlNewProp.
m. Any additional check, to avoid any kind of segmentation faults, memory leaks, and misleading messages. Catch and handle properly any fault, even if it happened inside a library you are using.

In any case of error the program will output a message starting with “Error:” and followed by an appropriate informative message. The program will disregard the fault command and continue to run.
Terminate the program only if no other course of action exists: An input/output file cannot be found or opened, the return value of a standard function (as malloc) is different than expected, cplex failed, or C argument is invalid. In such case 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-**13b**/ex4.
- 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 *(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 *(clustering_solution.xgmml*, *best_clusters.xgmml* and *results*) for the network you estimated as real.
- 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: 4

Every 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-13b

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_editing.c and cluster_editing.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.
A suggestion for partition to modules: main, validity_checks, graph, cluster_editing, calculate_statistics, output. This is only a basic suggestion and you may partition differently.
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.
Especially, you should aim to allocate only necessary memory and free objects (memory and files) as soon as it is possible.
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

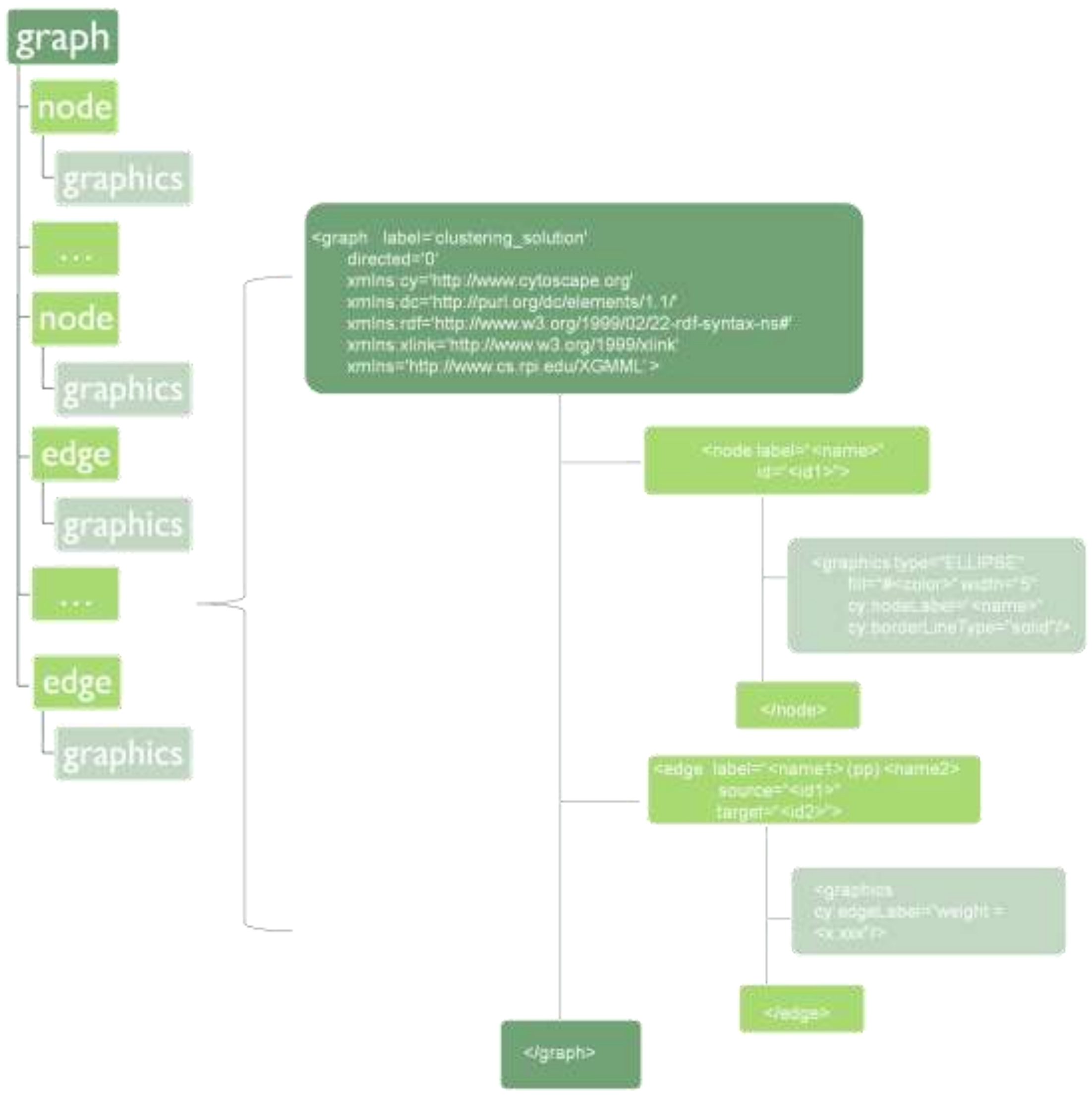
A skeleton makefile is provided at the website, compiling and linking cluster_editing.* with cplex interface. Notice that at first the linker will fail with **make all**, as there is no main() function declared in the files.
You will add your files to the makefile. Use the flags provided for you to link 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*



```
<graph   label='clustering_solution'
      directed='0'
      xmlns:cy='http://www.cytoscape.org'
      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' >

<node label="<name>"
      id="<id1>">

<graphics type="ELLIPSE"
      fill="#<color>" width="5"
      cy:nodeLabel="<name>"
      cy:borderLineType="solid"/>

</node>

<edge  label="<name1> (pp) <name2>
      source="<id1>"
      target="<id2>">

<graphics
cy:edgeLabel="weight =
<x.xxx"/>

</edge>

</graph>
```

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

Specification for clustering_solution file:
1. Graph label is changed to **clustering_solution**
2. Vertices ids are from 0 to number of vertices
3. Vertices and edges are to be inserted to the file in the same order as in the *results* file. Edge's source and edge's target correspond to the same fashion the edges was printed in *results* file (i.e. v1.id<v2.id => source = v1.id, target = v2.id).

Specification for best_clusters file:
1. Graph label is **best_clusters**
2. Only vertices residing in the 5 biggest clusters and between them appear
3. As before, vertices' color determined according to their cluster's identification.

4. Vertices ids are according to their arrival in the full network (i.e. it corresponds to their id in clustering_solution.xgmml)
5. Vertices and edges are to be inserted to the file in the same order as in the *results* file, excluding edges that are not among the 5 biggest clusters.

Use "diff" over the ONE OPTIMAL SOLUTION to verify your format.

Appendix B – Visualize network using cytoscape

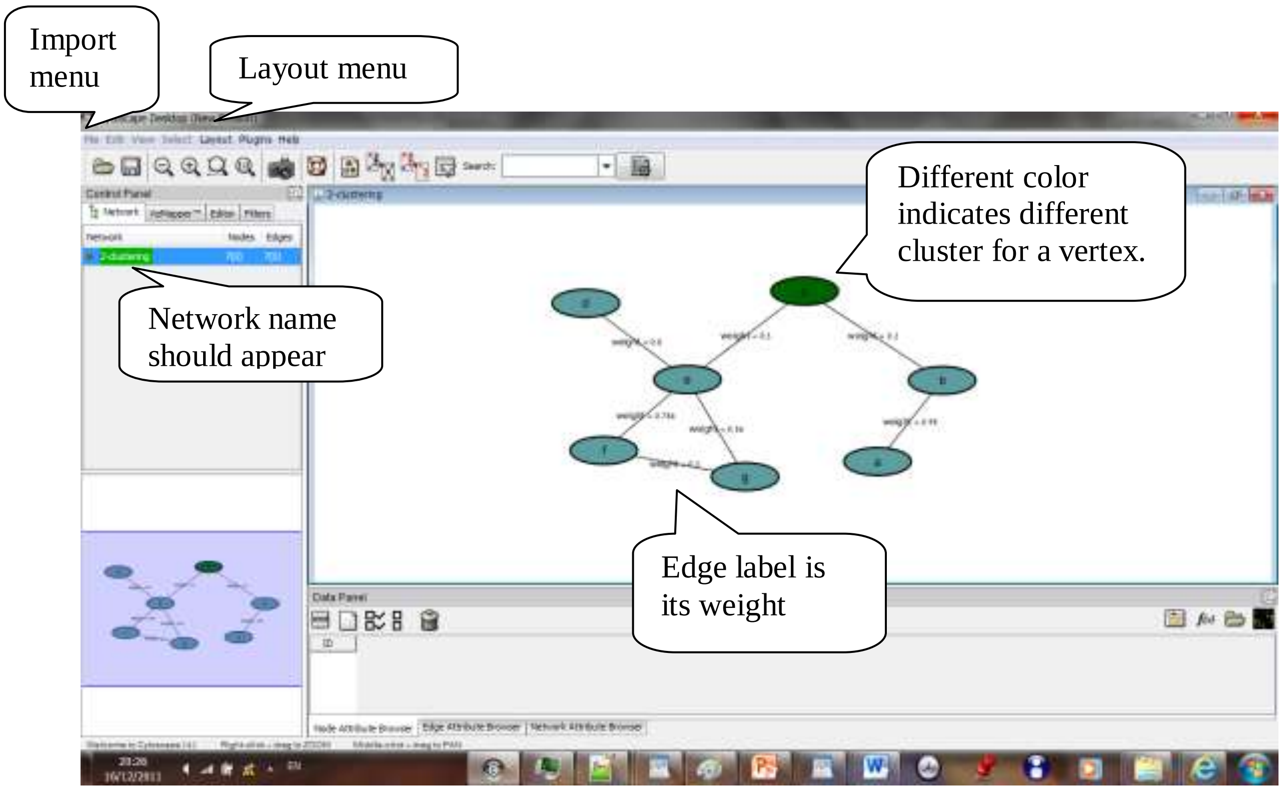
The network visual representation would be through the graphical software Cytoscape. Cytoscape is an open source software platform for visualizing molecular interaction networks. Cytoscape is installed on the university computers (open by writing **cytoscape** in the command line), and you can download 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. We recommend: 'Layout->Cytoscape Layouts->group Attributes Layout->*_NODE_FILL_COLOR'
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'. Another (safer) option is to just open a new window of cytoscape. Then follow the instructions again form step ii.

***Appendix figure 2** Import a network into cytoscape*


Import menu
Layout menu
Different color indicates different cluster for a vertex.
Network name should appear
Edge label is its weight
File Edit View Select Layout Plugins Help
Control Panel
Network
Nodes
Edges
2-clustering
Data Panel

Appendix C – Finding the biological functions associated with clusters

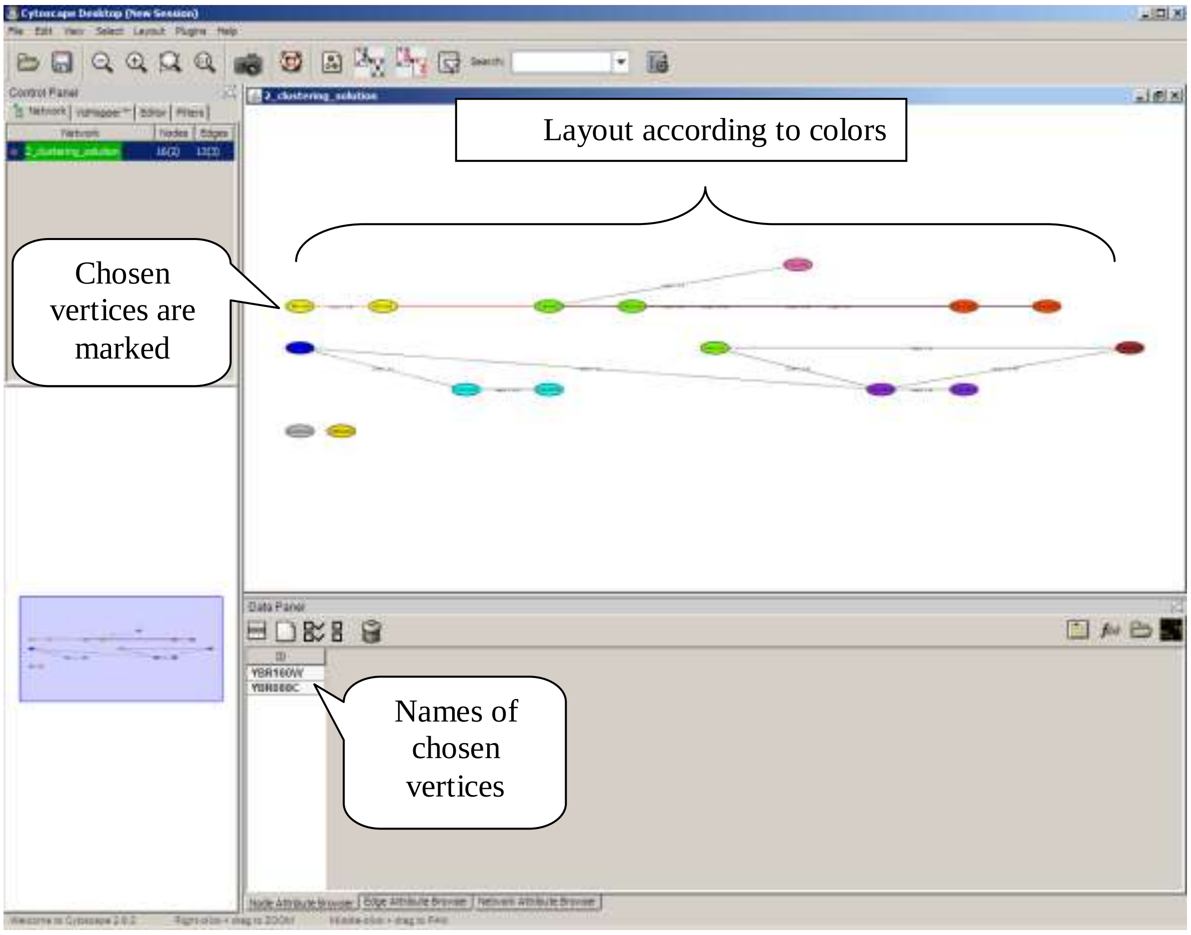
The functionality of Cytoscape can be extended by installing additional plugins (under its plugins directory).We use the BiNGO plugin to compute functions associated with gene clusters.
The BiNGO plugin can be freely downloaded from http://www.psb.ugent.be/cbd/papers/BiNGO/1, but in most installations 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 -> 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 accoring to color*


Cytoscape Desktop (New Session)
Control Panel
2_clustering_solution
Layout according to colors
Chosen vertices are marked
Data Panel
YBR160W
YBR088C
Names of chosen vertices

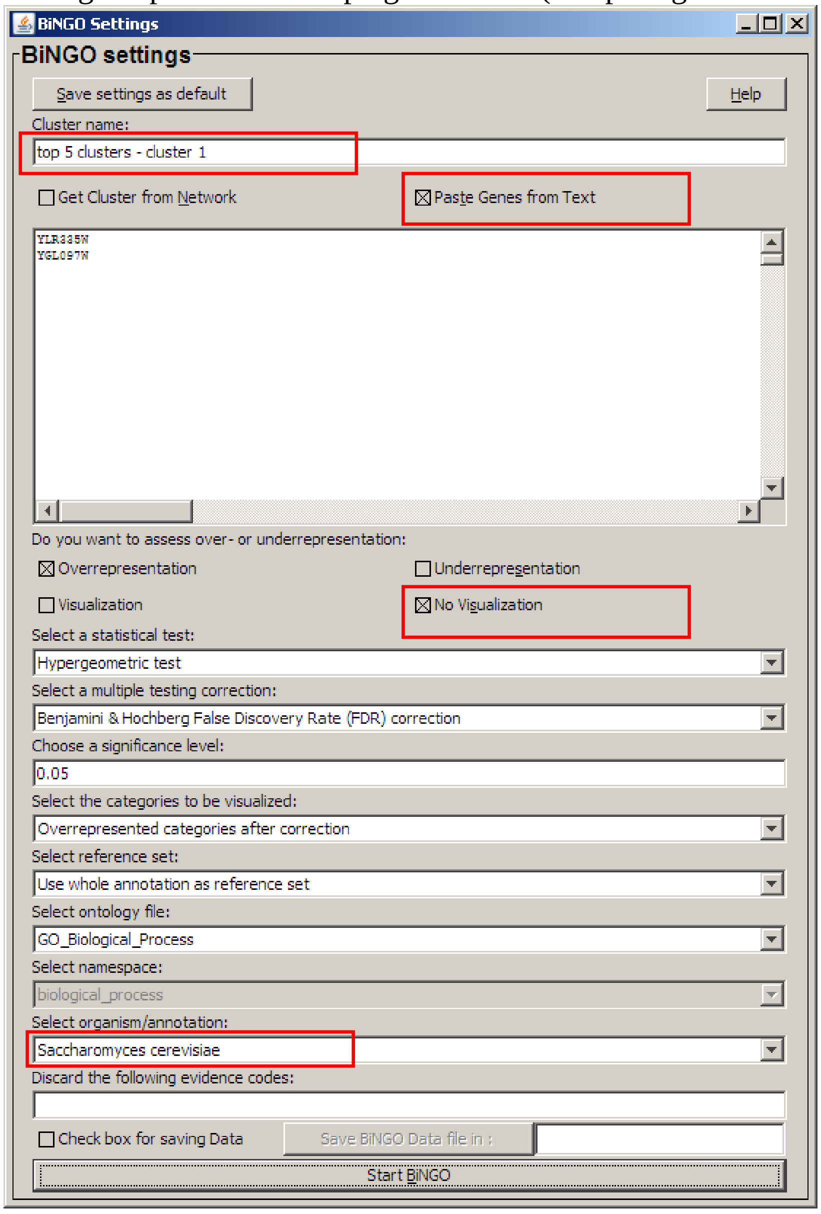
Run the BiNGO plugin by choosing from the main menu: Plugin→BiNGO 2.0. A window as in figure 4 will appear.
1. In the cluster name field write any name for the current cluster (figure 4 – BiNGO window).
2. Choose the option Paste Genes from Text (figure 4 – BiNGO window).

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).



BiNGO Settings

BiNGO settings

Cluster name: top 5 clusters - cluster 1

☐ Get Cluster from Network

☒ Paste Genes from Text

```
YLR335W
YGL097W
```

Do you want to assess over- or underrepresentation:

☒ Overrepresentation

☐ Underrepresentation

☐ Visualization

☒ No Visualization

Select a statistical test: Hypergeometric test

Select a multiple testing correction: Benjamini & Hochberg False Discovery Rate (FDR) correction

Choose a significance level: 0.05

Select the categories to be visualized: Overrepresented categories after correction

Select reference set: Use whole annotation as reference set

Select ontology file: GO_Biological_Process

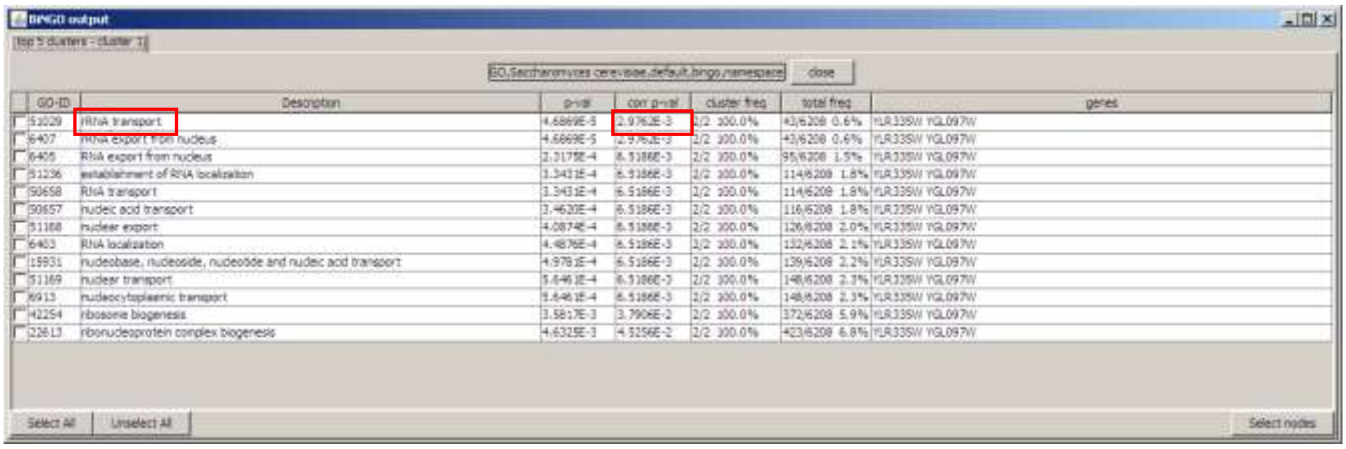
Select namespace: biological_process

Select organism/annotation: Saccharomyces cerevisiae

Discard the following evidence codes:

☐ Check box for saving Data

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


BiNGO output
top 5 clusters - cluster 1
GO.Saccharomyces cerevisiae.default.bingo.namespace
close
Select All
Unselect All
Select nodes

	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	rRNA export from nucleus	4.6869E-5	2.9762E-3	2/2 100.0%	43/6208 0.6%	YLR335W YGL097W
☐	6405	RNA export from nucleus	2.2175E-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

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.