

MetNet Tools (Cytoscape Plug-ins)

Kyongryun Lee

Julie Dickerson

Eve Wurlete

Iowa State University

Contents

1. Dot Layout	
1.1 Simple Dot Layout -----	3
1.2 Rank-Cluster Dot Layout -----	4
1.3 Layout XML File -----	4
2. Subgraph Creation	
2.1 Subgraph Creator -----	7
2.2 Choosig the Subgraph Creation Method -----	8
2.2.1 Read from File -----	9
2.3 Creating a Subgraph by Pathway Name -----	10
2.4 Creating a Subgraph by P Neighborhood -----	10
2.5 Other Subgraph Creation Options -----	11
2.6 Finding and Viewing Cycles -----	12
2.7 Finding and Viewing Paths -----	14
3. CytoScript	
3.1 Introduction -----	16
3.2 System Design -----	16
3.3 Example -----	17
3.4 Animations -----	17
3.5 Mapping Rule Specification -----	18
3.6 Graphical Demonstration -----	19
3.7 Animation GUI -----	22
4. Installation -----	27
Acknowledgment -----	28

1. Dot Layout

For a graph view, a graph layout algorithm must compute positions of the node and edge figures. Many graph layout algorithms are already implemented in cytoscape. Dot Layout is added to cytoscape. There are two types of layout that can be computed by Dot. One is a Simple Dot Layout, and the other is Rank-Cluster Dot Layout.

1.1 Simple Dot Layout

One of types in dot layout is a generic layout called the Simple Dot Layout. It can be selected from Layout Menu. Figure 1.1 shows a graph layout done using the simple dot layout.

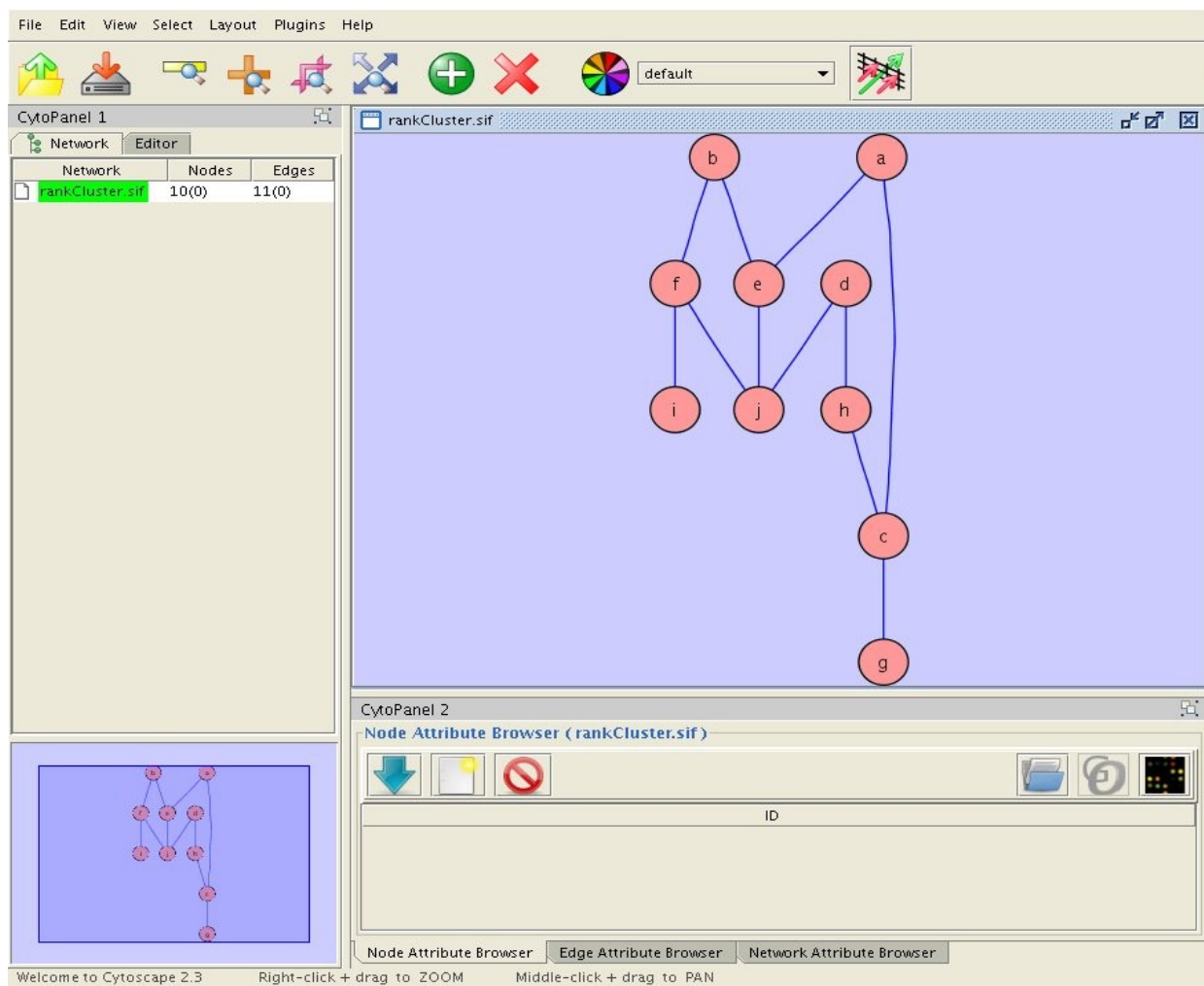


Figure 1.1 Example of the simple dot layout using a small graph.

1.2 Rank-Cluster Dot Layout

It can be also used to compute a more customized layout. Based on their values of certain node properties, the node figures can be placed on horizontal ranks or into clusters. From Layout Menu, select the Dot Rank-Cluster Layout, then an open file dialog pops up to show in which a layout XML file must be selected (see Figure 1.2.1 below). Figure 1.2.2 below shows the same graph

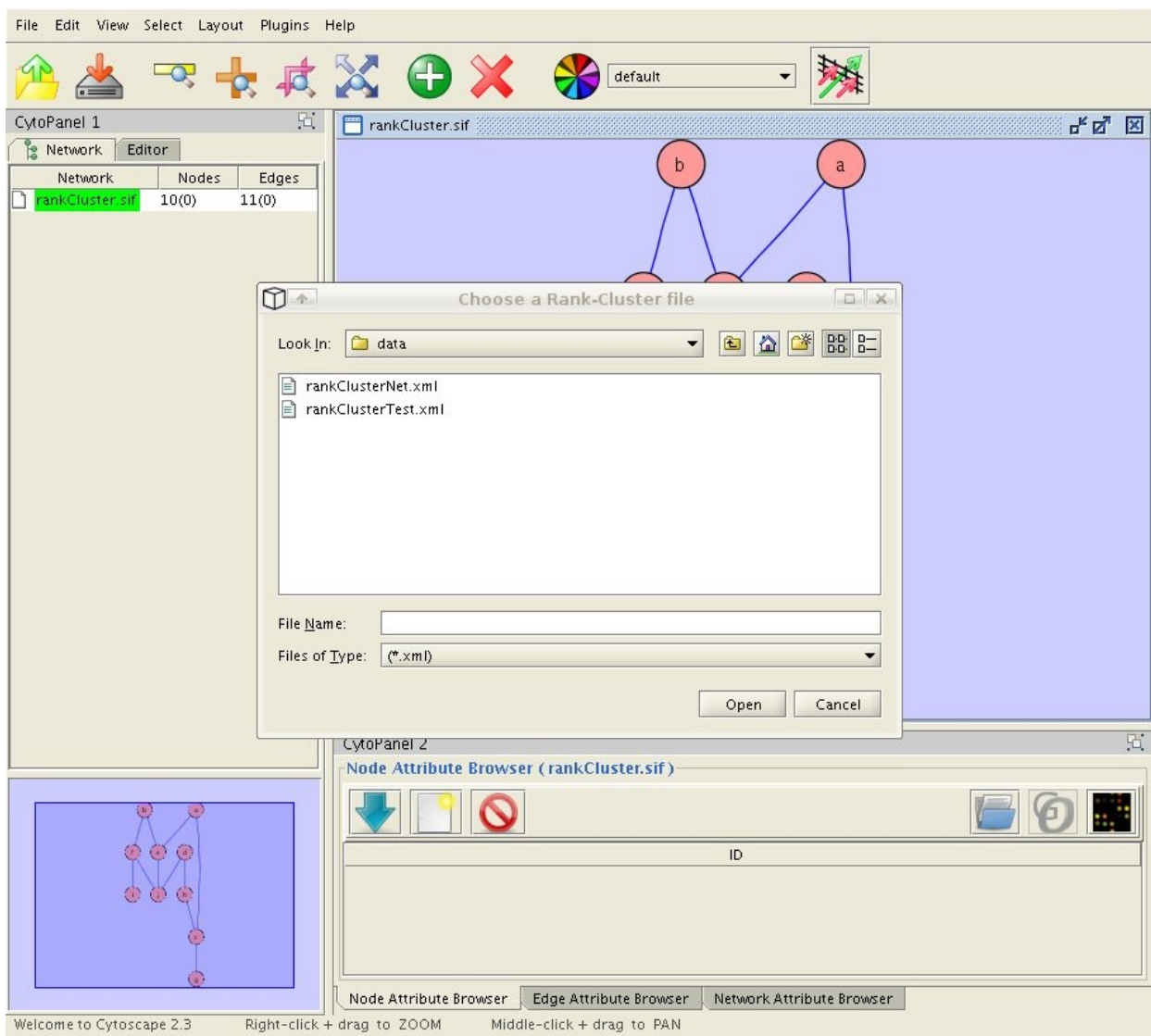


Figure 1.2.1 Dialog Box to choose a layout XML file

as in Figure 1.1 using the rank-cluster layout.

1.3 Layout XML File

The ranks and clusters of the dot rank-cluster layout are specified in a layout XML file. The XML file selected in Figure 1.2 (a) is shown below:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE layout [
<!ELEMENT layout (rank*, cluster*)>
<!ELEMENT rank (atom*, composite*)>
<!ATTLIST rank
label CDATA #REQUIRED>
<!ELEMENT cluster (atom*, composite*, cluster*)>
<!ATTLIST cluster
label CDATA #REQUIRED>
```

```

<!ELEMENT atom EMPTY>
<!ATTLIST atom
property CDATA #REQUIRED
value CDATA #REQUIRED>
<!ELEMENT composite ((atom | composite), connective, (atom | composite))>
<!ELEMENT connective EMPTY>
<!ATTLIST connective
type (and | or) #REQUIRED>
]>

<layout>

<rank label="rank1">
  <atom property="nodeType" value="type1"/>
</rank>

<rank label="rank2">
  <atom property="nodeType" value="type2"/>
</rank>

<rank label="rank3">
  <atom property="nodeType" value="type3"/>
</rank>

<cluster label="AandB">
  <composite>
    <atom property="label" value="a"/>
    <connective type="or"/>
    <atom property="label" value="b"/>
  </composite>
</cluster>

<cluster label="EandF">
  <composite>
    <atom property="label" value="e"/>
    <connective type="or"/>
    <atom property="label" value="f"/>
  </composite>
  <cluster label="OnlyG">
    <atom property="label" value="g"/>
  </cluster>
</cluster>

</layout>

```

The `<rank>` tag is used to specify the nodes that should be placed in a certain horizontal rank. Each rank has a label, specified by the `label` attribute, which is shown on the left-hand side of the graph view. Nodes are selected using the same type of XML as in the property-to-visual-attribute mappings (see section 3.5).

Similarly, the `<cluster>` tag is used to specify nodes to place in a cluster. Each cluster is surrounded by a rectangle and has a label, specified by the `label` attribute, which is shown on the upper-left of the cluster. The `<cluster>` tag uses the same node selection mechanism as the `<rank>` tag. `<cluster>` tags can be nested inside each other, creating nested clusters as shown in clusters EandF and

OnlyG above.

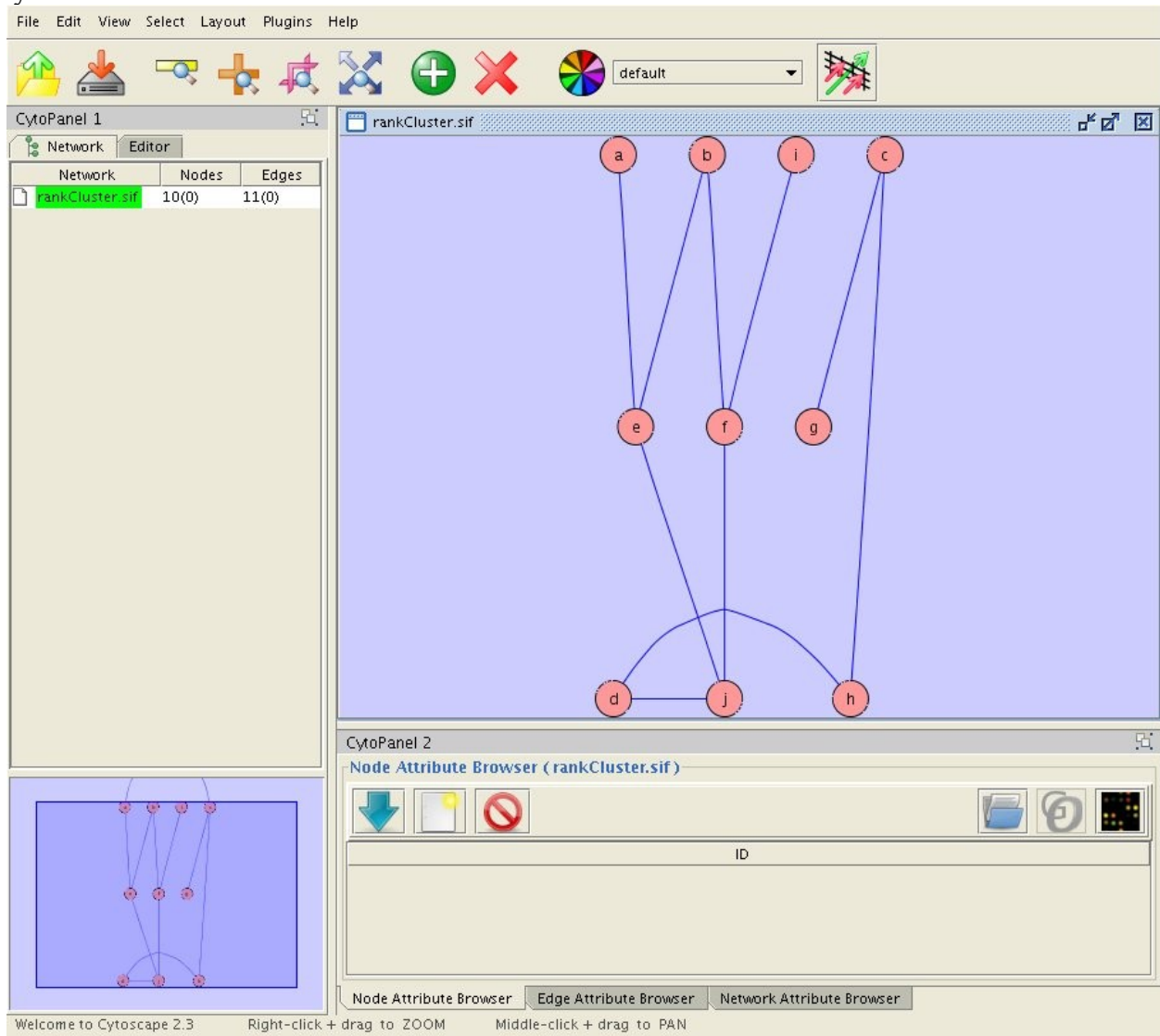


Figure 1.2.2 Rank-cluster dot layout.

2. Subgraph Creation

The Subgraph Creator creates a new subgraph of an loaded graph. This is to help us efficiently analyze and view the graph, creating a subgraph with selected nodes and edges. We can see the subgraph creation on plugins menu, and it follows 4 steps to create.

2.1 Subgraph Creator

Open the Subgraph Creator by selecting Create Subgraph from plugins menu. The first screen of the Subgraph Creator is shown in Figure 2.1. There are two options. One is the all nodes and edges in memory, and the other is the graph in active graph window ONLY. The first option is for that the new subgraph could contain any node and edge in the graph archive. The second is for that the new subgraph can only contain nodes and edges from the top open graph.

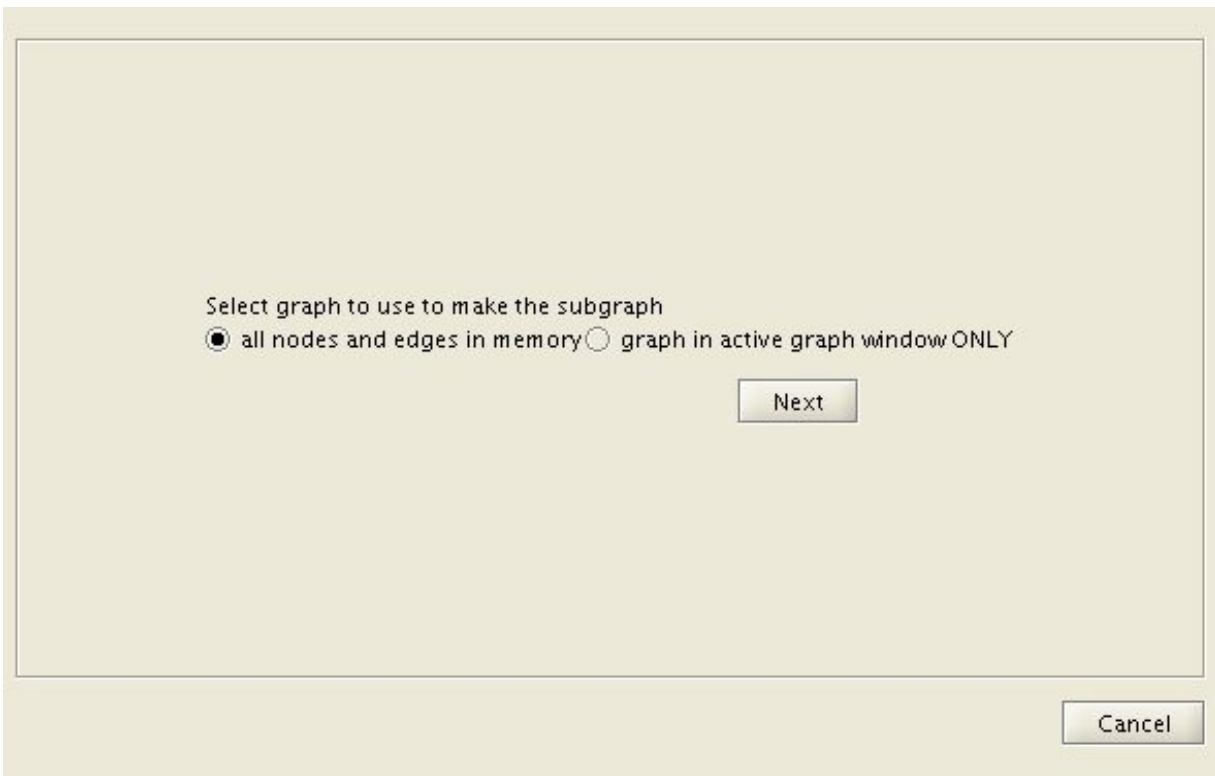


Figure 2.1 The first screen of Subgraph Creator.

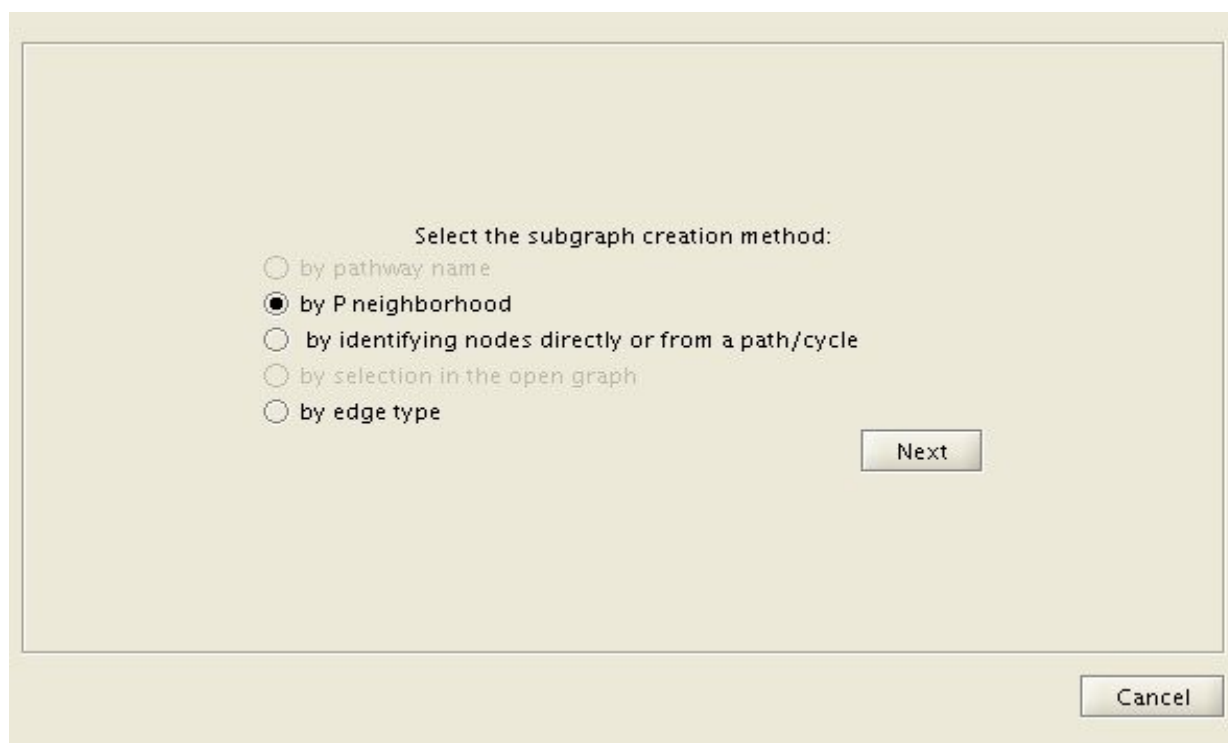


Figure 2.2.1. Main Subgraph Creator screen. This screen lets you indicate how you choose the nodes and edges in the subgraph. Some of these options may be unavailable, depending on the files present in the graph archive.

2.2 Choosing the Subgraph Creation Method

The subgraph creation method is indicated on a frame of the Subgraph Creator (See Figure 2.2.1 above). This shows available method lists; some options may be unavailable, depending on the files present in the graph archive. Let us select one of the available method lists and click the Next button. For instance, we select this option, by P neighborhood.

The subgraph creation method can be applied by going through the Subgraph Creator multiple times. The screens for each of the individual subgraph creation methods (e.g. Figure 2.2.2) have a set of three buttons in a frame at the bottom: Create subgraph now, Create subgraph then choose another method, and Choose another method without making a subgraph now. The Create subgraph now button creates the subgraph based on the previous selection and closes the Subgraph Creator while the other two buttons take you back to the main Subgraph Creator screen. The Choose another method without making a subgraph now button cancels the current action, while the Create subgraph then choose another method button creates the subgraph but instead of showing it, holds it in memory and returns you to the main Subgraph Creator screen. In this way a subgraph can be made by any combination of options.

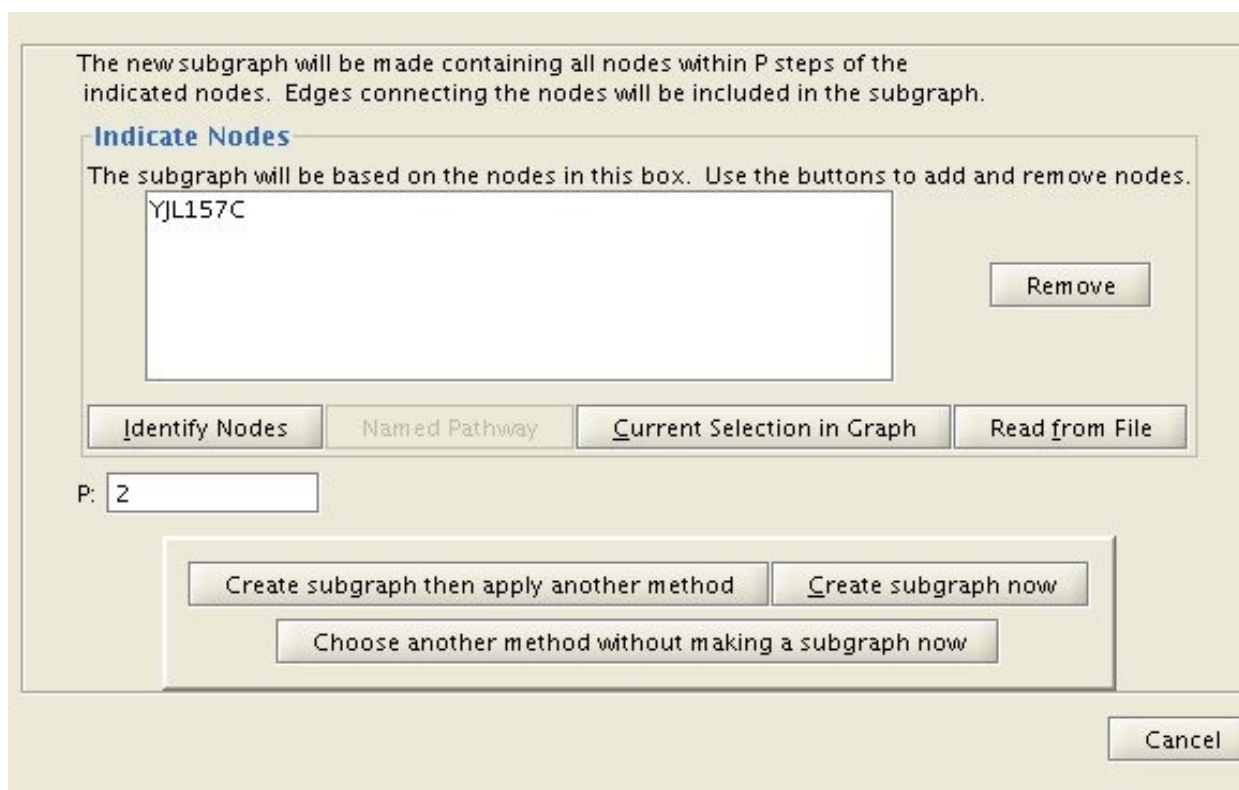


Figure 2.2.2. Subgraph Creator by p neighborhood subgraph screen. The screen is set up to create a subgraph of the node within two steps of the *YJL157C* node.

For instance, you may have a large graph archive that has nodes from multiple pathways. You may wish to view all of the nodes in a particular pathway that are within 5 steps of a particular node. You can do this by first creating a subgraph by pathway name, then clicking the *Create subgraph* then choose another method button, then creating a subgraph by P neighborhood.

2.2.1 Read from File

To create subgraph, we are able to select nodes from the file. In figure 2.2.2, the button “Read from File” works to read selected node from the file. For the file, it is able to read all kinds of attributes. File format is as follows.

```
ID
YER179W
YGL035C
YIL143C
YPL248C
YOR120W
YBR018C
YOL051W
YBR019C
```

At the first line, it must be an attribute name. From the second line, there are node names with the

attribute name.

2.3 Creating a Subgraph by Pathway Name

For by pathway name, all of the pathways will be listed in the large box; select the pathway(s) whose nodes and edges you want included in the subgraph. As many pathways as desired can be selected.

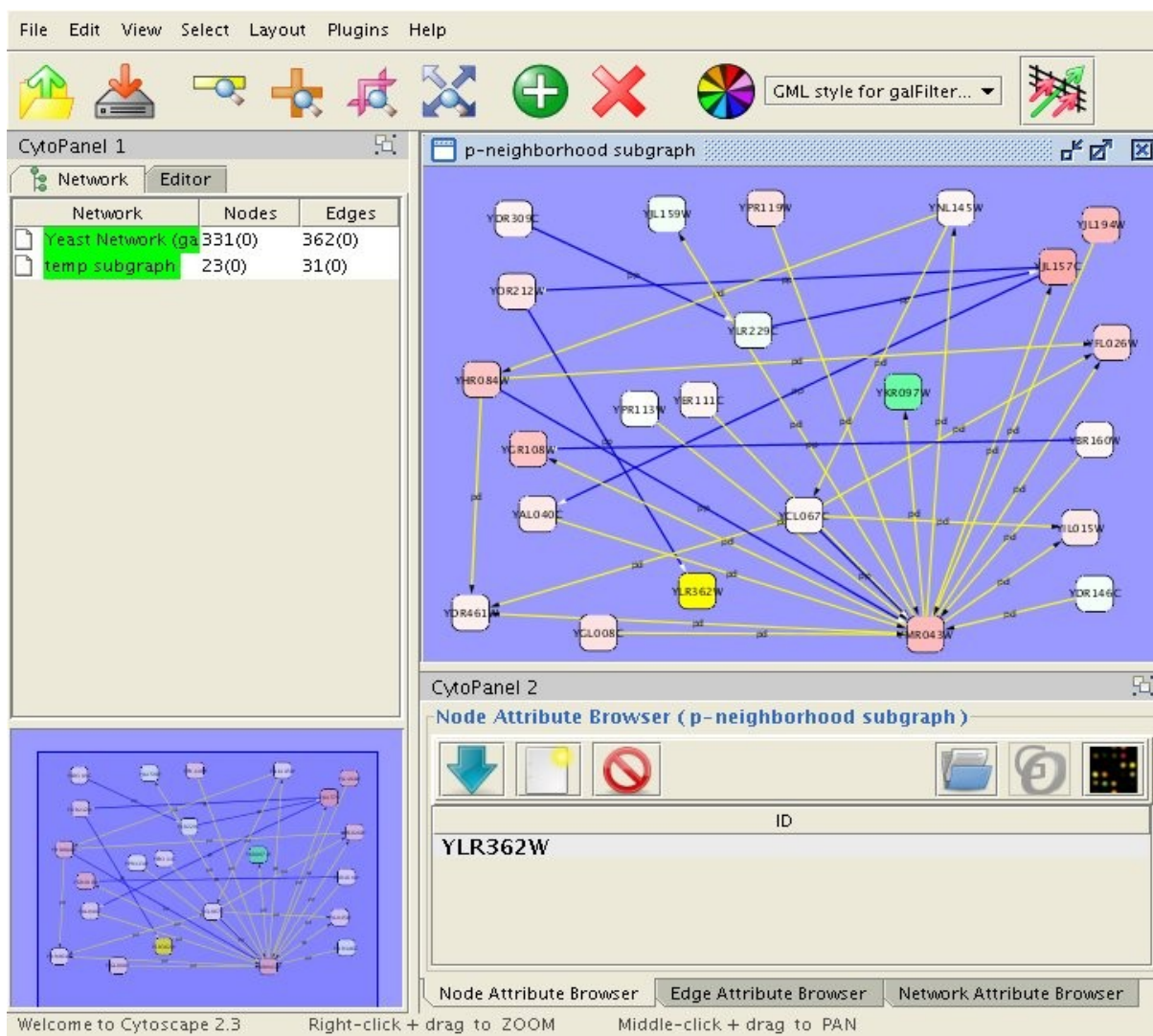


Figure 2.4 Subgraph of nodes within 2 steps of the node *YJL157C*

2.4 Creating a Subgraph by P Neighborhood

A p neighborhood is the set of nodes p steps from the node(s) of interest. Nodes connected by an edge are one step apart. The screen used to make p neighborhood subgraphs is shown in Figure 2.2.2

above. Figure 2.4 shows the p neighborhood of one node, but p neighborhoods can be made for many nodes. The nodes that will be used for the p neighborhood subgraph are shown in the white list on the p neighborhood subgraph screen. Nodes are added to this list by clicking the Identify Nodes (each node selected using the Identify Node dialog), Named Pathway, Current Selection in Graph (the node(s) selected in the top graph are added to the list), or Read from File (the file should be a simple text file listing the unique ID of each node on a separate line) buttons. After adding the desired node(s) to the list, type the p value in the P box.

2.5 Other Subgraph Creation Options

The by identifying nodes directly or from a path/cycle option provides an interface for several subgraph creation options. All of these options let you specify the nodes that you want to include in the subgraph; edges connecting these nodes will also be included. If you know which nodes you want to include, you can use the Identify Node dialog. Alternatively, a particular cycle or path can be selected using the path or cycle finder in the usual manner. Figure 2.5.1 shows the creator of by identifying nodes directly or from a path/cycle.

Moreover, we can create a subgraph by selecting the option, by edge type. Figure 2.5.2 shows subgraph creator by edge type.

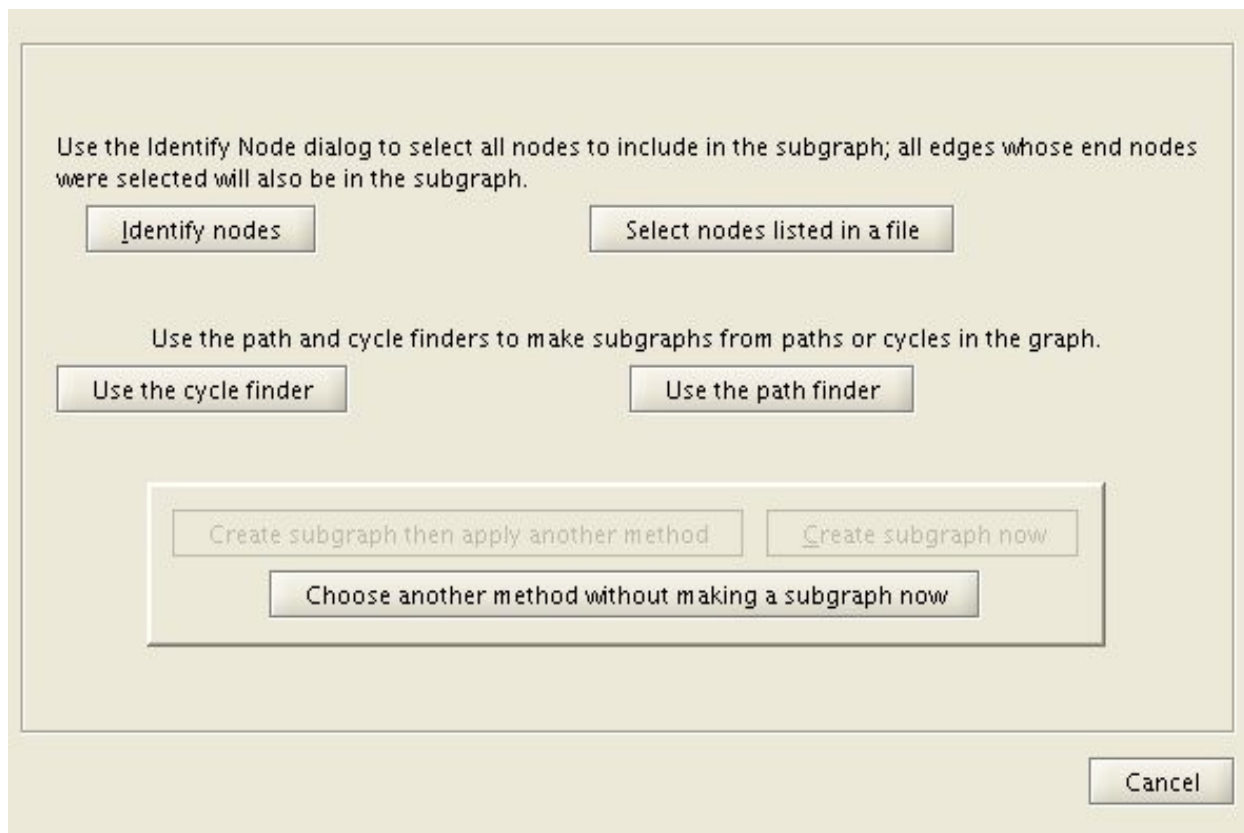


Figure 2.5.1 Subgraph Creator by identifying nodes directly or from a path/cycle

2.6 Finding and Viewing Cycles

A path in a digraph is an alternating sequence of nodes and edges, $v_1e_1v_2e_2v_3 \dots v_{k-1}e_{k-1}v_k$, such that $\text{tail}(e_i) = v_i$, $\text{head}(e_i) = v_{i+1}$, and all nodes are distinct. A path can be viewed as starting at node v_1 and following edges through the graph until node v_k is reached. If $v_1 = v_k$, the path is called a cycle.

The 'Find Cycles' is used to find cycles in CytoScape. The dialog is opened by selecting Find cycles from plugins menu. The dialog will find all cycles in the graph by clicking the "Find all cycles" button, or you can restrict it to searching for cycles of a particular length or containing particular nodes by clicking the "Find Cycles" button. To restrict the cycles to particular nodes, click either the Identify node or Select node buttons. The Max cycle length box is used to set the maximum cycle size (in number of edges).

The 'Find Cycle' dialog also contains two menus namely 'File' with Menu item 'Save Cycles' and 'Filter' with two menu items namely 'Filter Cycles' and 'Remove Filter'. By clicking the 'Save cycles', the found cycles can be copied to a data file for future reference. The Screen shot for this operation is showed in the Figure 2.6.2. Figure 2.6.1 shows that there are four Cycles found in an example network (galfiltered.gml) and out of them one is highlighted in Yellow (nodes) and red (edges) colors.

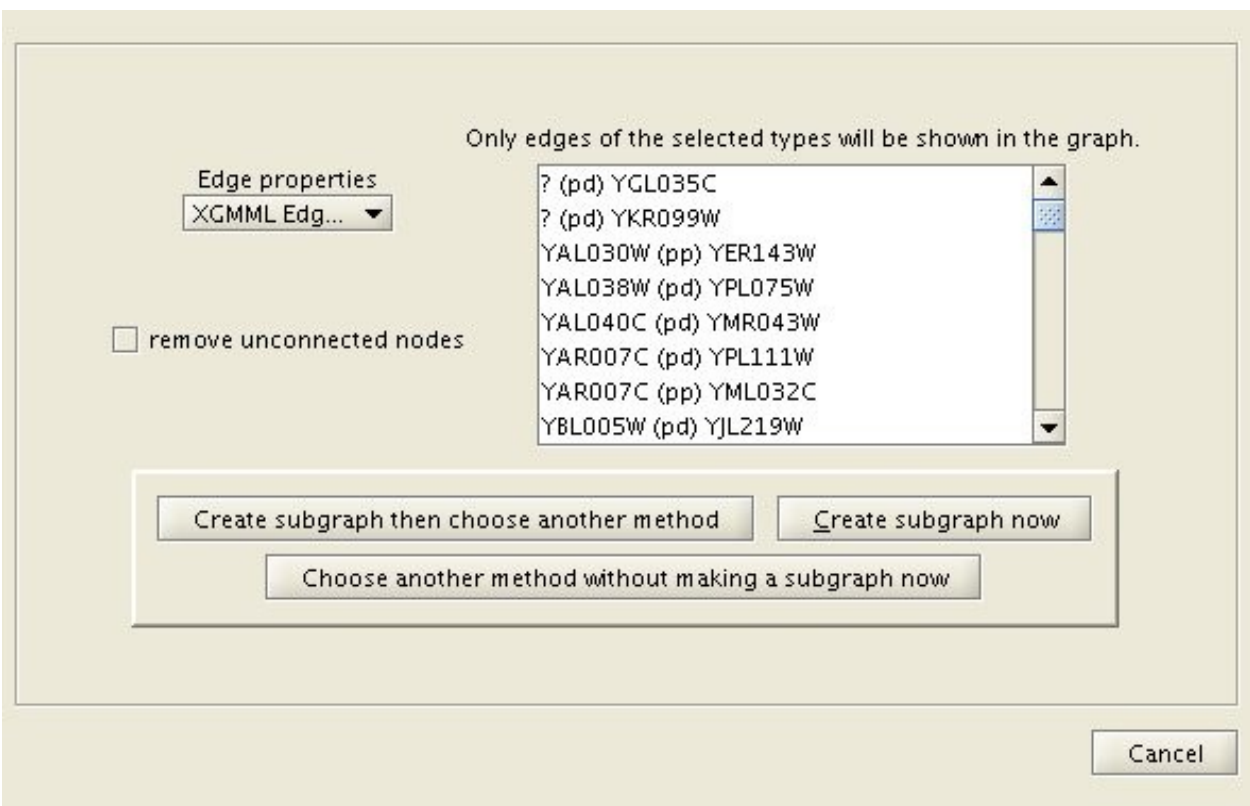


Figure 2.5.2 Subgraph Creator by edge type

particular nodes, click either the Identify node or Select node buttons. The Max cycle length box is used to set the maximum cycle size (in number of edges). Click the Find cycles button to find cycles of the length and containing the nodes you specified, or click the Find All

Cycles button to find all cycles in the graph.

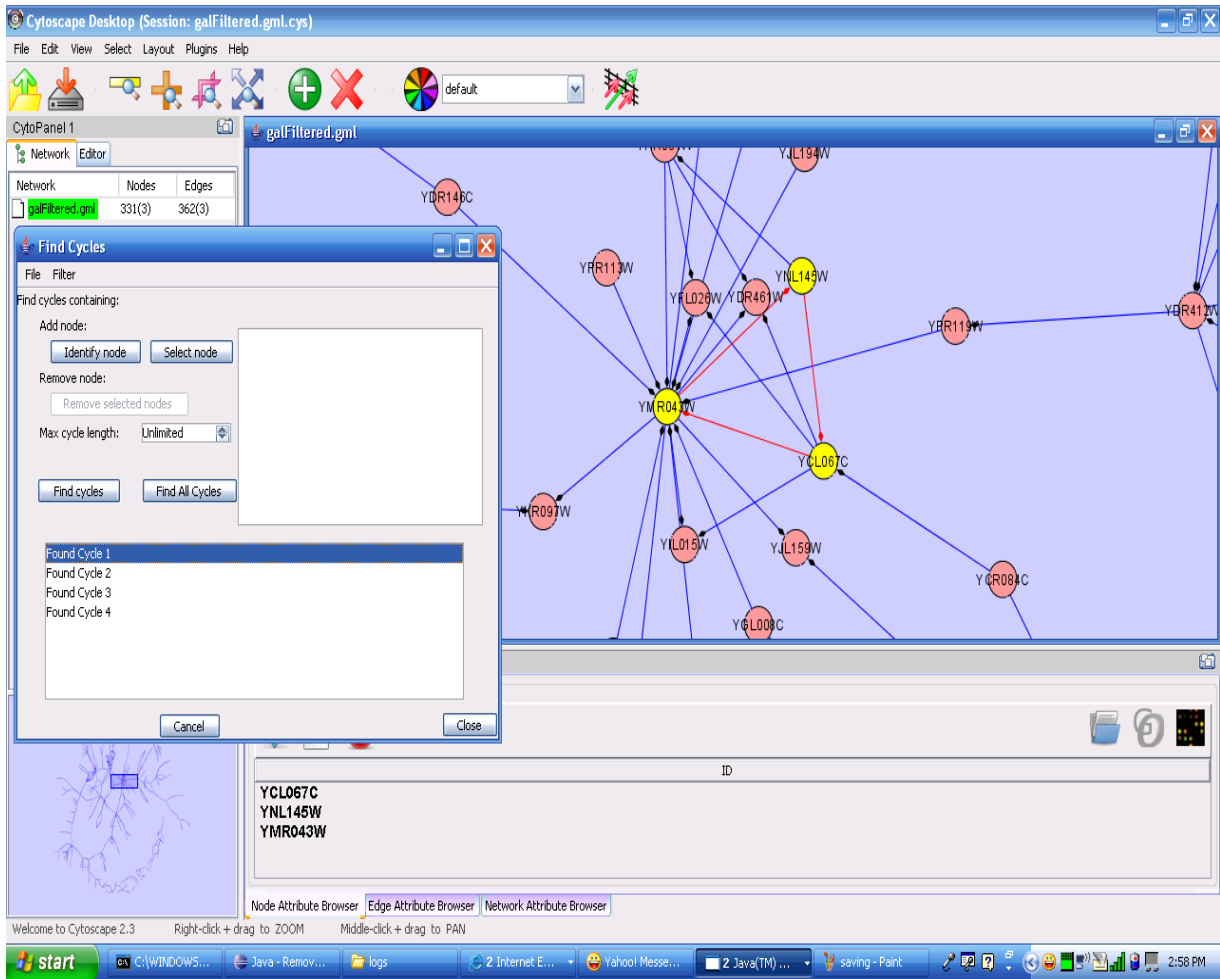


Figure 2.6.1. The Find Cycle dialog. The cycle which consists of nodes with yellow color is highlighted in red color.

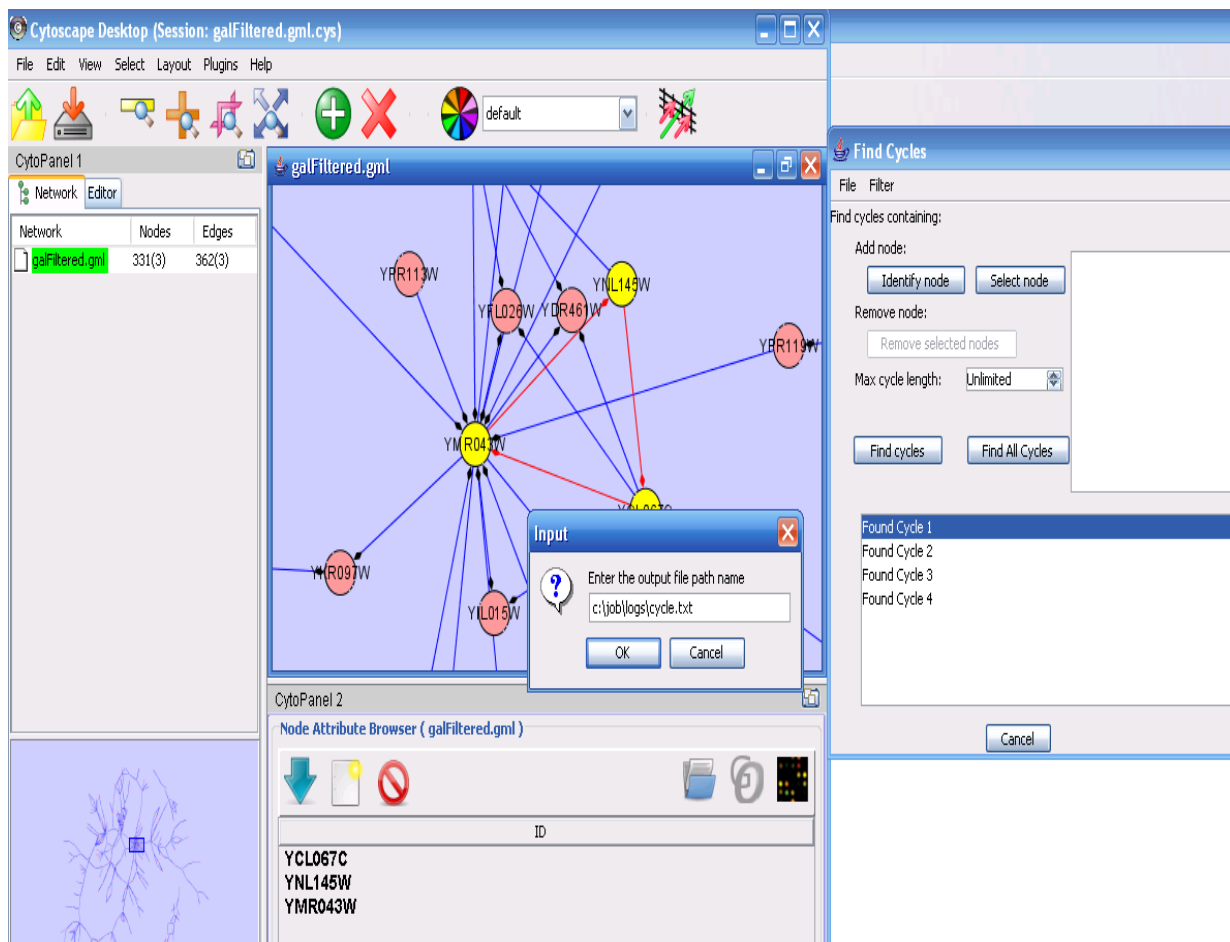


Figure 2.6.2 The Find Cycle dialog. The cycle which consists of nodes with yellow color is highlighted in red color and we are able to save the cycle.

2.7 Finding and Viewing Paths

A path is the collection of nodes and edges that lead from one node to another node. The `Find Paths` searches for all paths of the specified length between two nodes. Open the `Find Paths` dialog selecting `Find Paths` from plugins menu

Set the starting and ending node for the path using either the `Identify Node` dialog (by clicking the appropriate `Identify Node` button) or by selecting the nodes directly in the graph (select the node then click the appropriate `Select Node` button). The maximum path length is set to *unlimited* by default; change this to the maximum path length you want (each edge is counted as one unit of length) and click the `Find Paths` button to perform the search. All paths found will be put in the list box; click on the path name to view it in the graph. Figure 2.7 shows the result of this function.

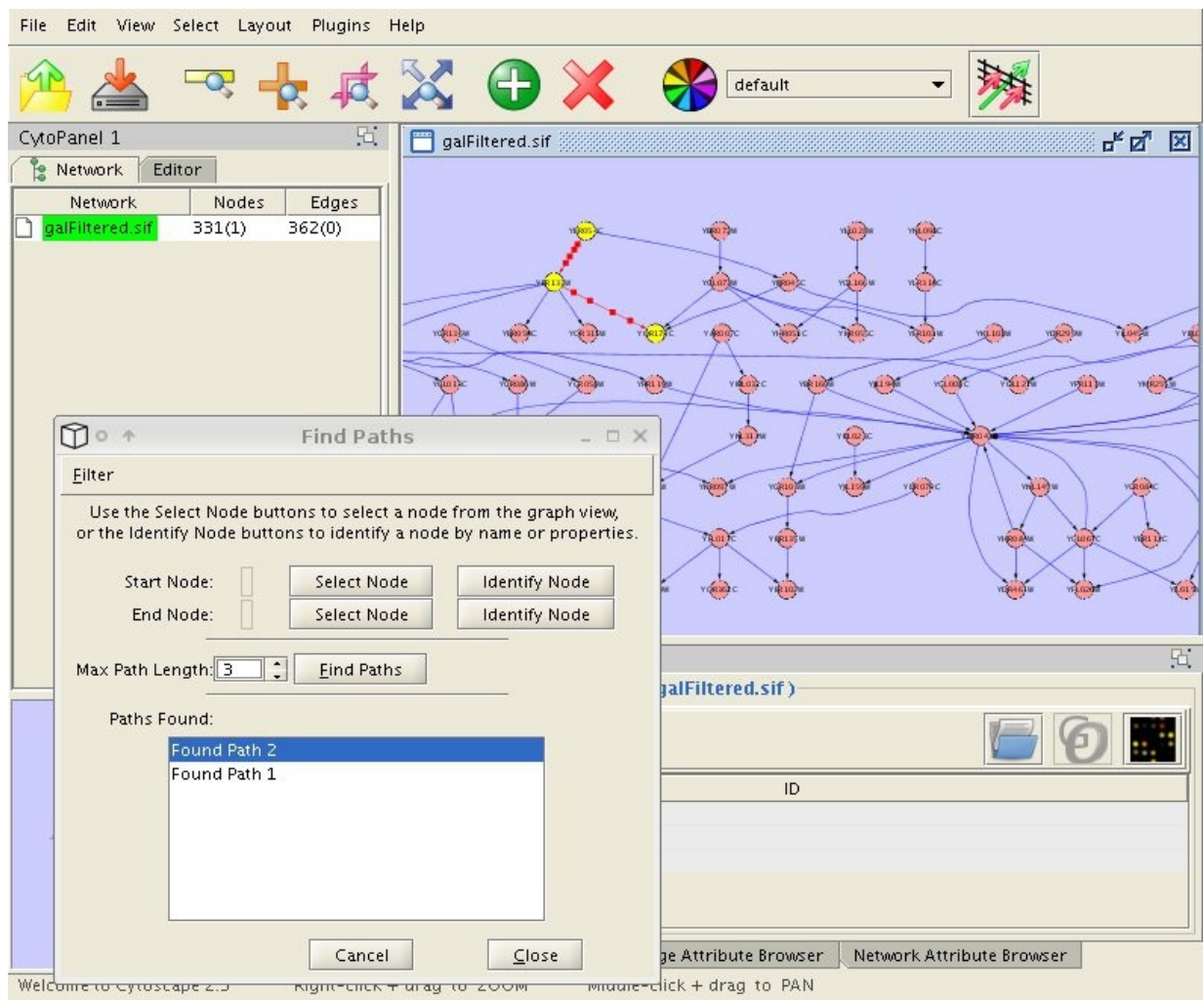


Figure 2.7. Find Paths dialog. The dialog found all paths from node *YDR229W* to *YDR461W* (in yellow). One path was found and highlighted in red color.

3. CytoScript : Running Scripts in CytoScape

3.1 Introduction

CytoScript is a program to execute scripts in languages such as R, Scheme, Python, and Perl in Cytoscape. The goal of CytoScript is to extend Cytoscape through scripts, with the target application being the analysis of network or experimental data. The results of the analysis may then be represented in the Cytoscape network view. The scope of this document is limited to running R in Cytoscape.

R is a scripting language for statistical computing and graphics. It provides various statistical methods such as clustering, classical statistical tests, linear or nonlinear modeling and various graphical techniques. Moreover, the Bioconductor project provides many R packages (essentially plugins for R) designed for data analysis in bioinformatics. Thus, researchers often use R to analyze biological experimental data and networks.

Cytoscape is a bioinformatics software to visualize networks and to visually integrate these networks with gene expression profiles and other experimental data. Many researchers are developing useful programs to analyze data and networks in Cytoscape through plugins like CytoScript.

3.2 Design

Figure 2.1 shows the structure of a system to run R in cytoscape. The step of the structure is as follows. First, we use the GUI provided by CytoScript to load a file written in R. Second, CytoScript has uses javax.script to execute scripts written in any language with a javax.script engine implementation. The R engine is provided by the edu.iastate.metnet.Rpackage that relies may eventually support JRI, SOAP, and Rserve backends, though only JRI is supported currently. JRI is a simple Java to R interface written by Simon Urbanek that allows calling R from Java. Finally, the R script is executed by JRI.

The R script uses the rJava and rmetnet packages. rJava is the inverse of JRI in that it provides a link from R to Java. It allows the script to show the results of analysis in Cytoscape. The rmetnet package provides convenience functionality for interacting linking R with the MetNet platform. Currently its sole feature is an implementation of the abstract graph class (provided by the Bioconductor graph package) using a CyNetwork object from Cytoscape.

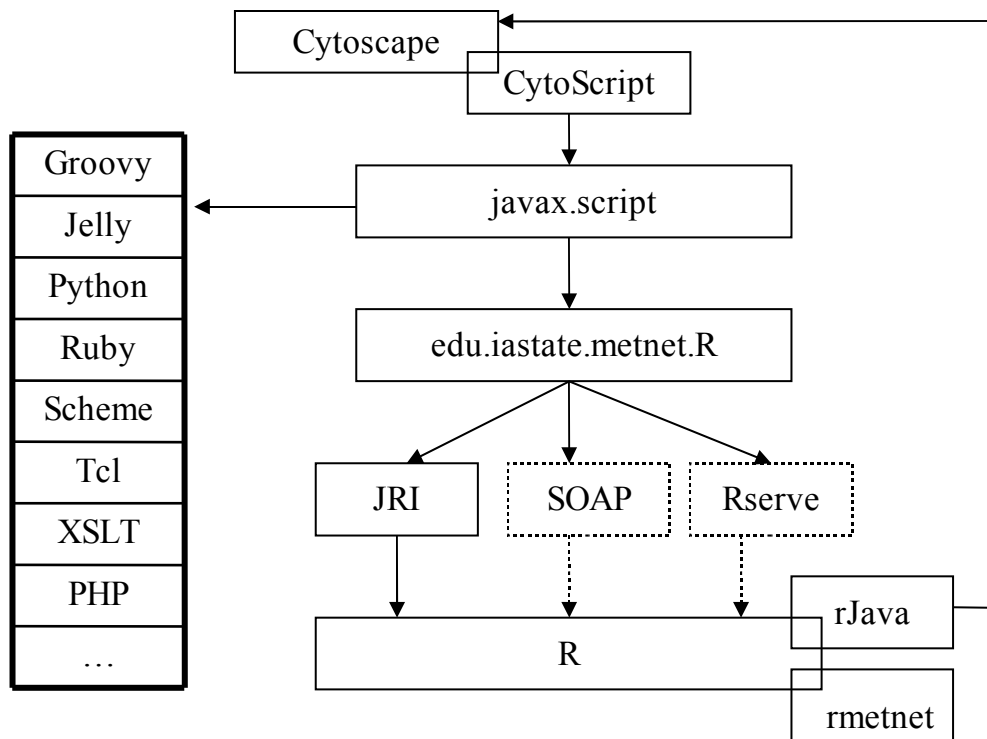


Figure 3.2 System Design

3.3 Example

As an example, let us find strongly connected components (SCCs) for a given network, using R.

1. Call Cytoscape to get the current network, using rJava.

```
cy_network <- .jcall("cytoscape/Cytoscape", "Lcytoscape/CyNetwork;",
"getCurrentNetwork")
```

2. Create our graph object.

```
network <- new("graphCytoscape", cy_network)
```

3. Find SCCs in the graph.

```
comps <- strongComp(network)
```

3.4 Animations

The Animator plugin provides animation support to Cytoscape, so scripts run via Cytoscript may use animations to show results. Animations may affect anything controlled with the VizMapper tool, which includes color, shape, font face, arrow type, line type, size of font, node height and width, label and label color. Each frame of the animation consists of a Cytoscape VisualStyle object with an associated duration.

It is not trivial to construct VisualStyle objects, so the Animatorplugin takes a language-oriented approach and provides a simple expression-based mechanism for describing the mappings that constitute a VisualStyle. This language of so called “Mapping Rules” are described in the next section. For instance, let us make it animate to fill the red color of some nodes.

Example

1. We make animation object from the program in R, using rJava.

```
animation <- .jnew("edu/iastate/cytoscape/animation/SimpleAnimation").
```

2. Next we create a frame for each strongly connected component.

2-1. We use mapping rules to specify that the nodes in the component are to be colored red.

```
rules <- paste("node canonicalName =", comp, "-> fill color = 255,0,0", col = ";")
```

2-2. Make a frame object which is the basic mapping rule (VisualStyle) storage unit in the animation framework. Here 0.5 is the duration of the frame.

```
frame <- .jnew("edu/iastate/cytoscape/animation/SimpleFrame", rules, 0.5)
```

2-3. Add the frame to the animation.

```
.jcall(animation,, "addFrame",  
        .jcast(frame, "edu/iastate/cytoscape/animation/Frame"))
```

3. Play the animation.

```
.jcall("Animator",, "playAnimation",  
        .jcast(animation, "edu/iastate/cytoscape/animation/Animation"))
```

Following above steps, we can make animations easily from R using mapping rules.

3.5 Mapping Rule Specification

The mapping rule syntax is inspired by that of the FCModeler mapping rules. The general syntax is given below:

```
type attribute-test -> appearance
```

The `type` indicates the type of element that is being mapped and may be either node or edge. The `attribute-test` follows the syntax:

```
attribute-name operator attribute-value
```

where the `attribute-name` is the name of an attribute in Cytoscape’s CyAttributes object, for the

given element type, operator is one character in the set {=, <, >} that tests the named attribute against the attribute-value. Finally, the appearance relates a visual property-name to a property-value by the syntax:

```
property-name = property-value
```

property-name could be “fill color”, “border color”, “font face”, “font size”, “height”, “width”, “shape”, “lable”, “label color”, “tooltip”, “line type” for node and “color”, “line type”, “source arrow”, “target arrow”, “lable”, “tooltip”, “font face”, “font size” for edge.

As a simple example, if one wished to fill all of the nodes with a value for “expression” greater than 3 with blue:

```
node expression > 3 -> fill color = 0,0,255 //R,B,G for property-value
```

Multiple mapping rules may be separated by the semicolon.

3.6 Graphical Demonstration

The following figures demonstrate the generation of an animation by an R script via CytoScript.

Demonstration • Used script language : R

- Goal : find strongly connected components (SCCs)
- Input : Yeast Network
- Output : colored strongly connected components with red.
(Others are colored by white)

We begin with the network shown in Figure 3.6.1. In Plugins, we select CytoScript, and load the file written in R (see figure 3.6.2). Figure 3.6.3 shows strongly connected components with red colors. There were two sets of SCCs found in the YeastNetwork. To see each individually, we can create an animation. To play animation, we write an R script that generates the animation from the analysis results. It loads the frames into a new animation and displays the animation GUI. In the play animation dialog, we can set the speed by adjusting the delay bar and the choose the desired cycle mode using radio buttons. Please see Figure 3.6.4.

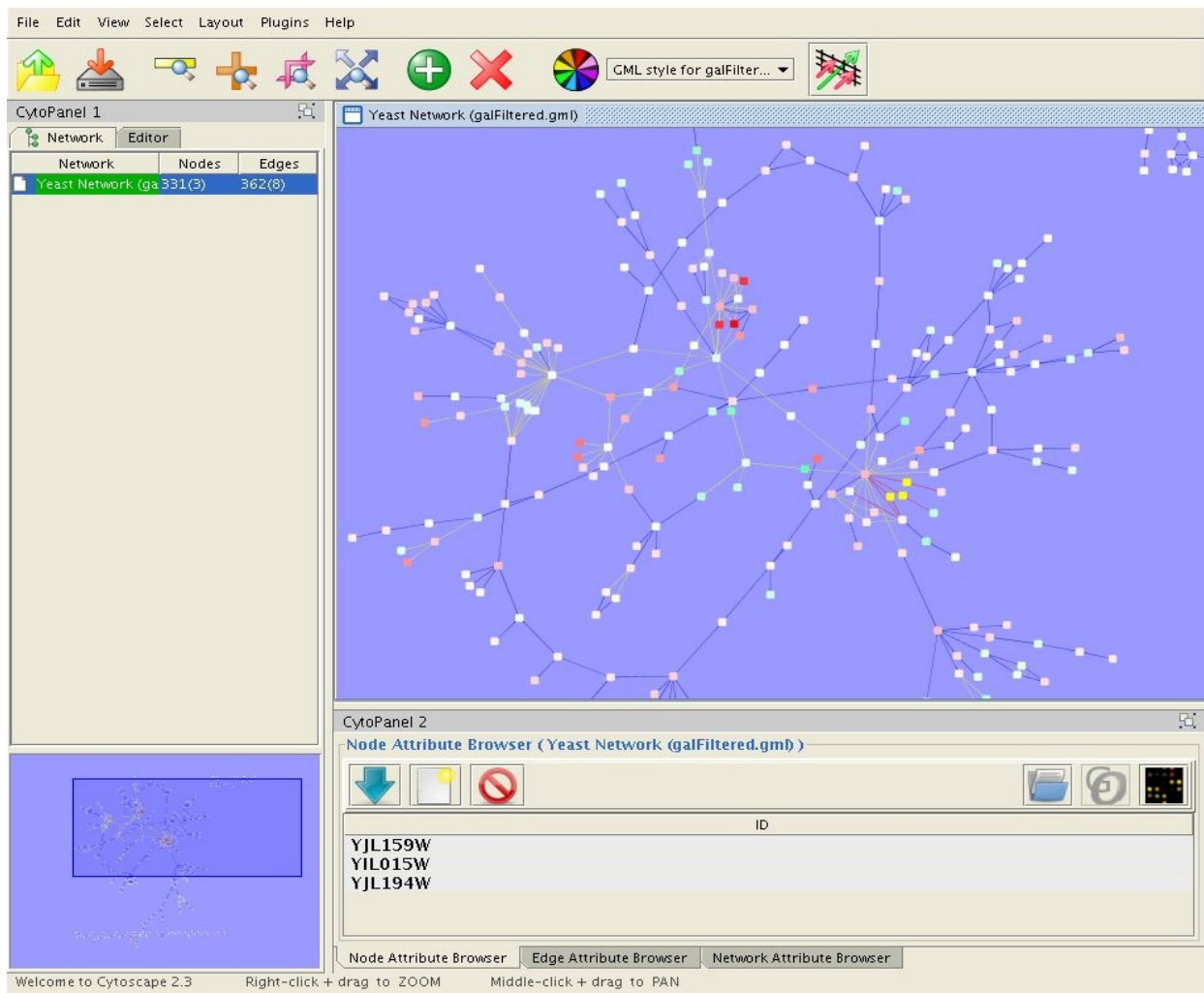


Figure 3.6.1 Current Network

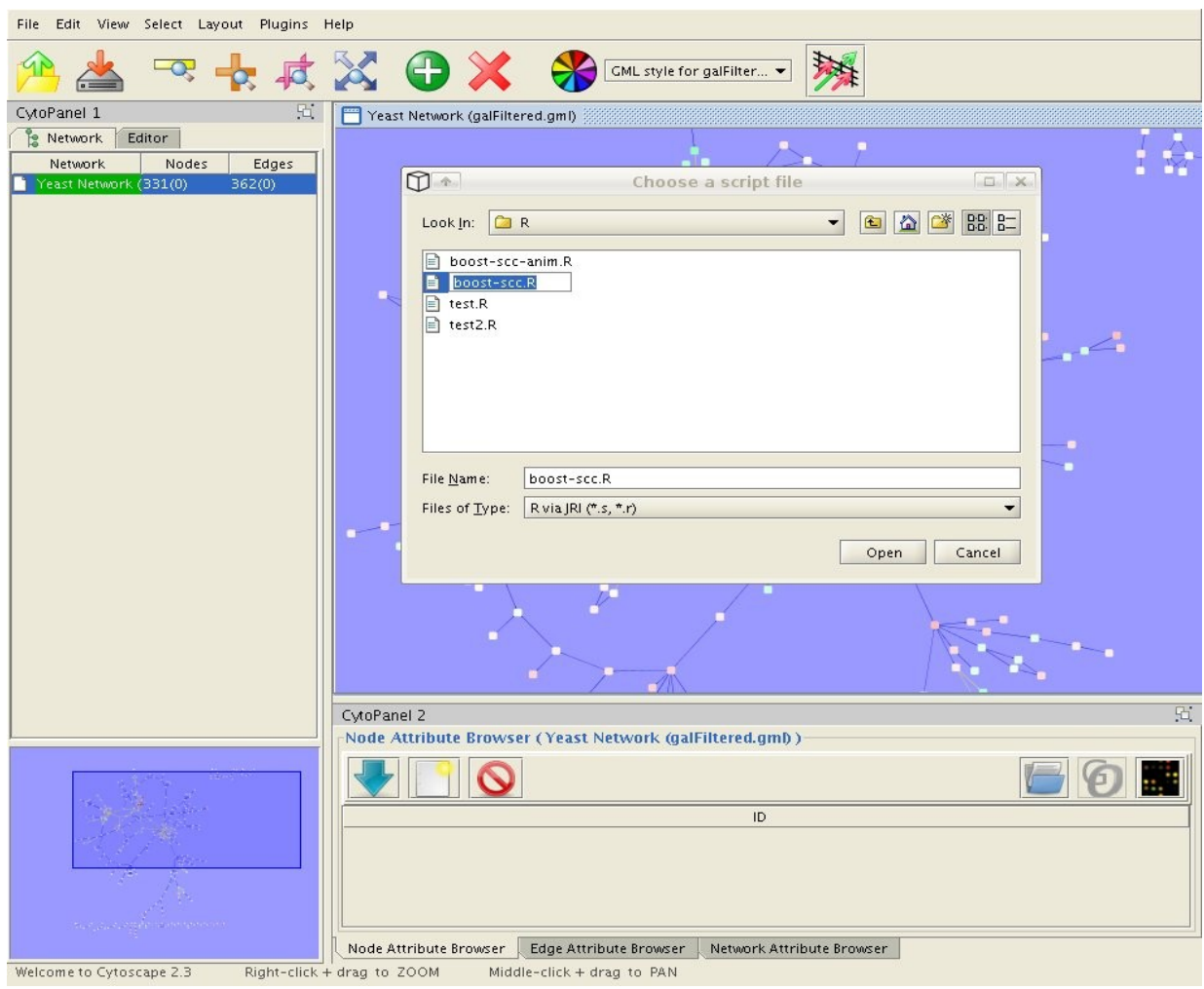


Figure 3.6.2 Loading a file

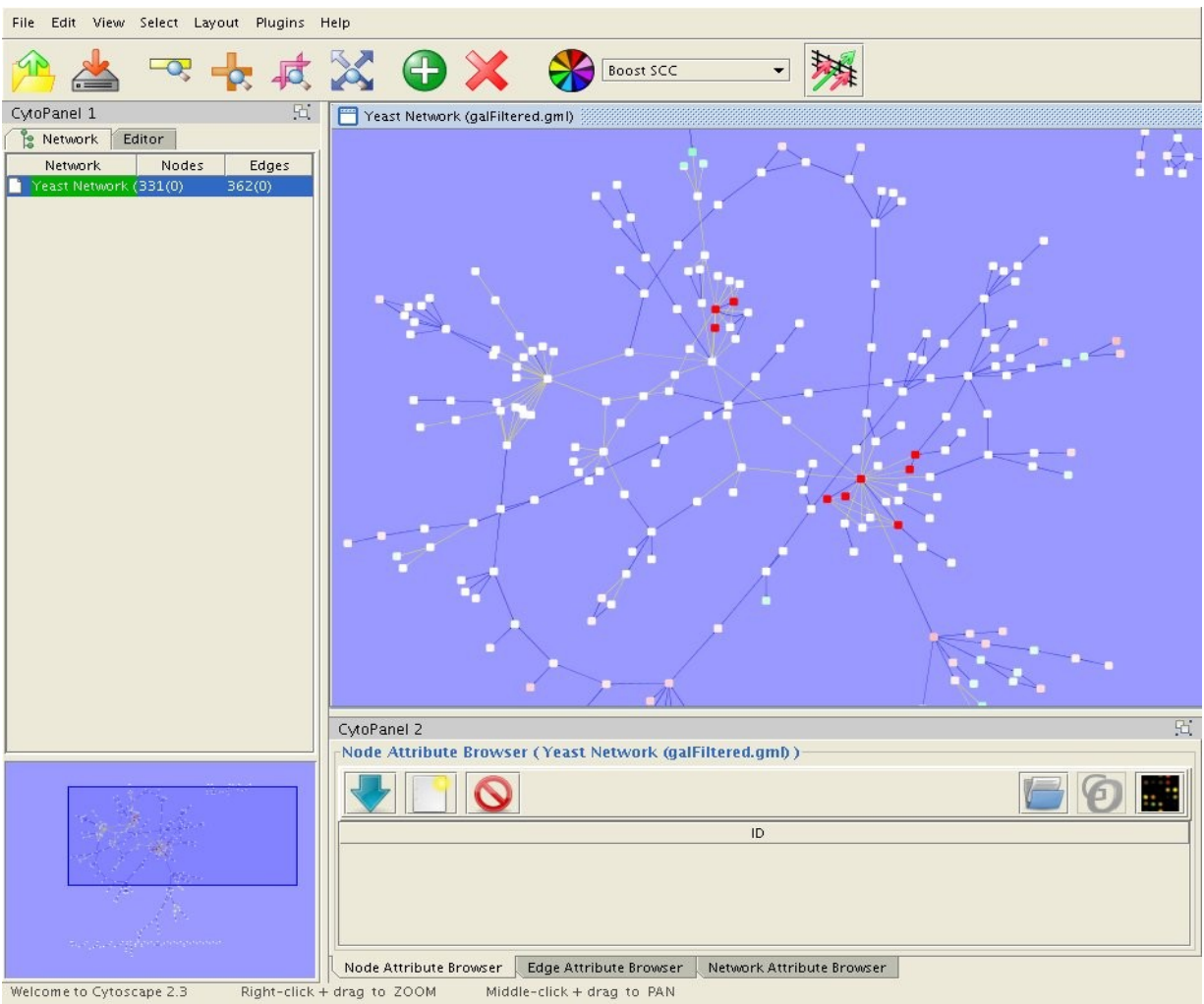


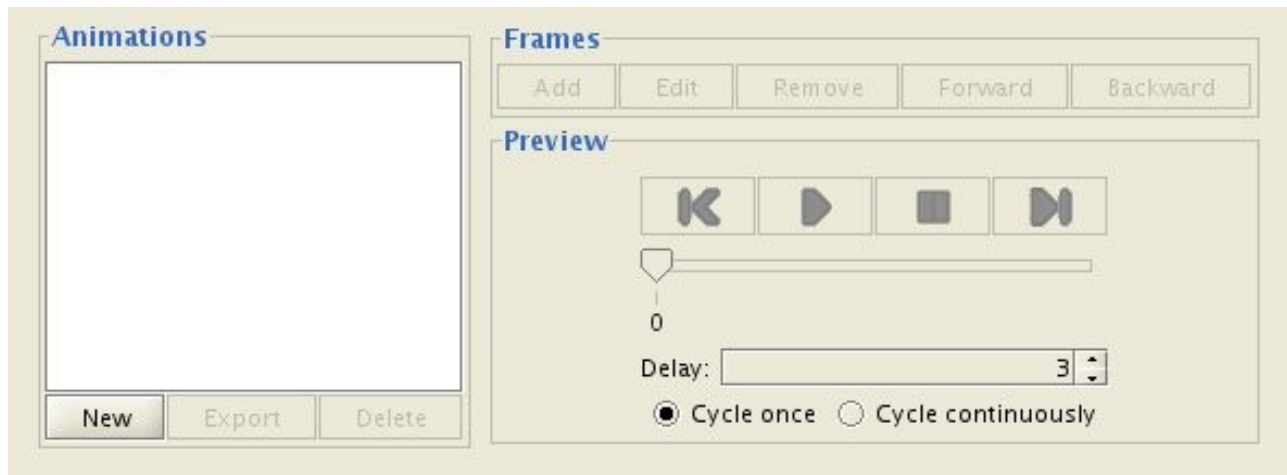
Figure 3.6.3 SCCs in the network

3.7. Animation GUI

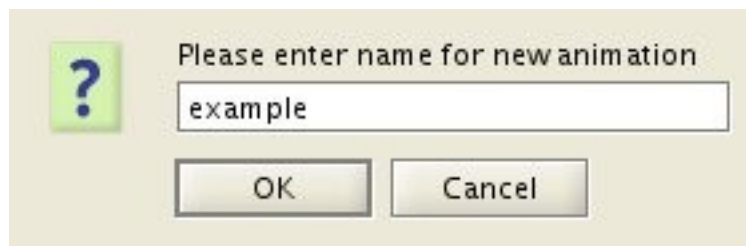
The Animator plugin also provides animation GUI to Cytoscape, so we may run animation of a given network in our favorite. We are able to select the properties of visualization through VizMapper on the GUI. A following example is shown in detail.

1. Import a network. For this example, we use a sample network, “GalFiltered.sif”

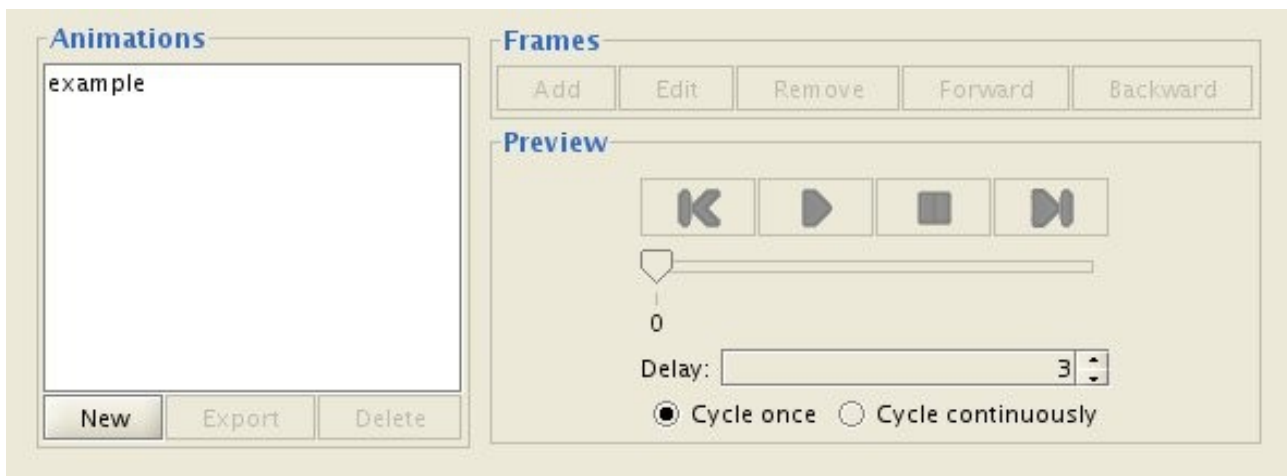
2. Choose “Create Animation” on menu of Plugins. Animation GUI would be shown up as follows.



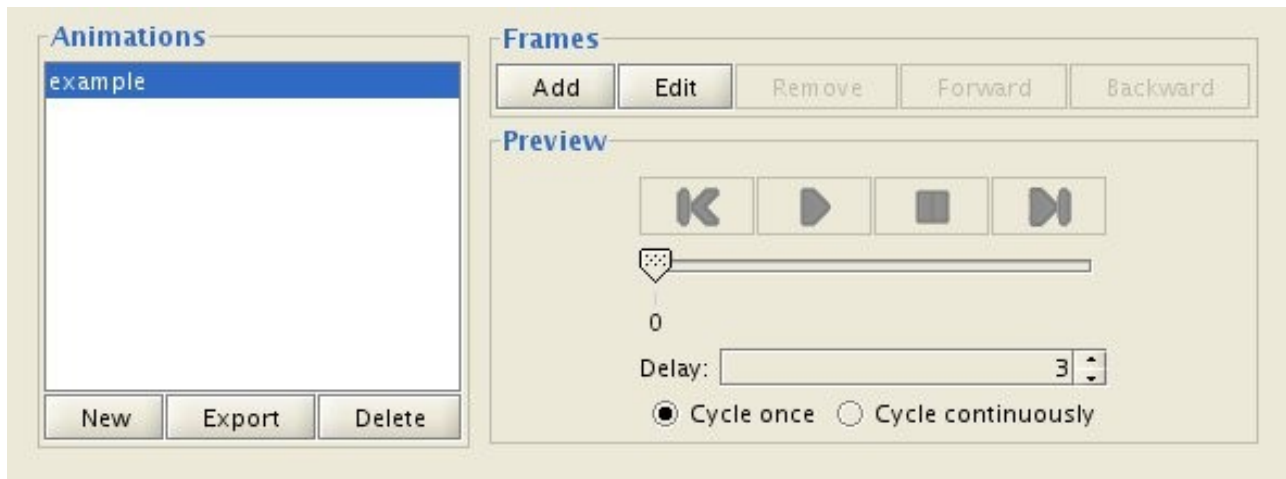
3. Click “New” button. This button is to create the name of animation.



4. Click “OK” button. Then the animation name would be registered on the GUI.

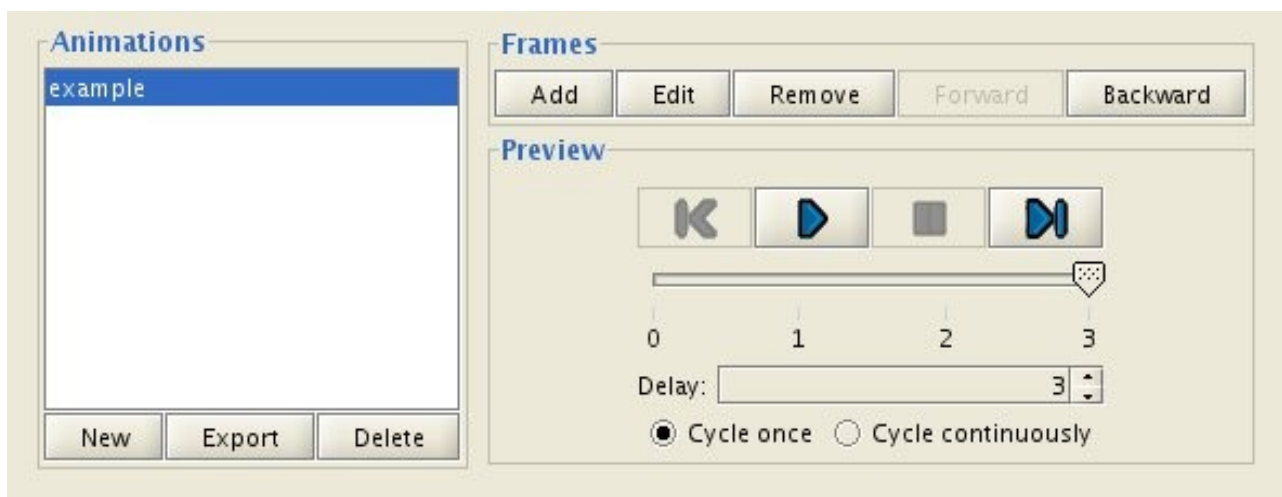


5. We need to make the frame of animations to run. When we click animation name, example, the buttons, ”Add” and “Edit”, are active. “Add” button is to make the frame and “Edit” button is to set up the visual properties on the frame.



6. Click “Add” button. Then one frame would be created for the animation. On this frame, we set up the visual property. Click “Edit” button then VizMapper would show up. We edit visual properties through this VizMapper such node color, node shape, etc. For example, we set the node color with the value of attributes, “1”, to red. If we want to make more frames, click “Add” button and click “Edit” button and set up the visual properties. In this example, we created three frames and set the node color with the value of attributes, “1”, to red, with the value of attributes, “2” to yellow and with the value of attributes, “3” to blue.

7. Then, we can see final GUI to run as follows.



8. For more explanation, on the Preview, we can see the frame bar. Through this frame bar, we can see visual style of animation on each frame. On the Frame, “Backward” button moves the frame

on the backward and “Forward” button moves the frame on the forward.

9. Let us run our animation. To do this, click “Play” button.

10. We can see the animations we created. Each frame is shown as follows.

The screenshot displays the Cytoscape 2.4.0 software interface. At the top, there is a menu bar with 'File', 'Edit', 'View', 'Select', 'Layout', 'Plugins', and 'Help'. Below the menu is a toolbar with various icons for file operations and navigation. A search bar is located to the right of the toolbar, with a dropdown menu currently set to 'node'. The main workspace is divided into two panels: 'CytoPanel 1' and 'CytoPanel 2'. 'CytoPanel 1' contains a 'Network' tab and an 'Editor' tab. A table below the tabs shows the network details:

Network	Nodes	Edges
galFiltered.sif	331(0)	362(0)

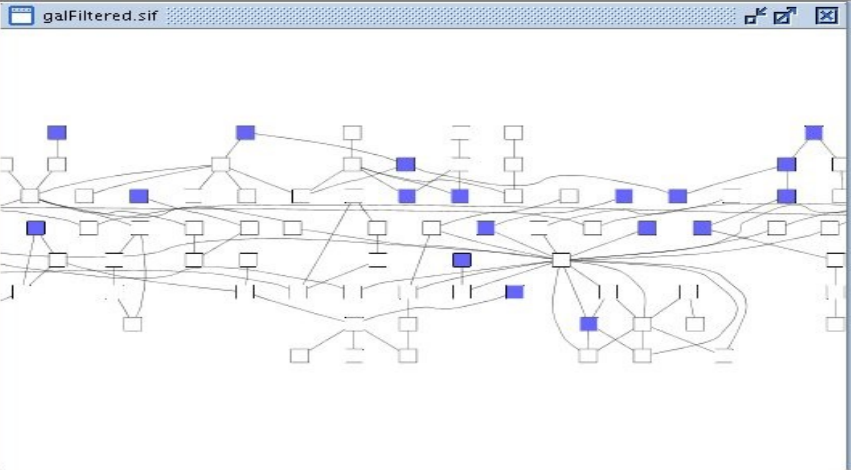
The main visualization area shows a complex network graph with numerous nodes and edges. Some nodes are highlighted in red, while others are white. The graph is dense and interconnected. 'CytoPanel 2' is currently empty, showing only an 'ID' field. At the bottom of the interface, there are three tabs: 'Node Attribute Browser', 'Edge Attribute Browser', and 'Network Attribute Browser'. The status bar at the very bottom reads: 'Welcome to Cytoscape 2.4.0 Right-click + drag to ZOOM Middle-click + drag to PAN'.

File Edit View Select Layout Plugins Help

node Search:

CytoPanel 1

Network	Nodes	Edges
galFiltered.sif	331(0)	362(0)



CytoPanel 2

ID

Node Attribute Browser Edge Attribute Browser Network Attribute Browser

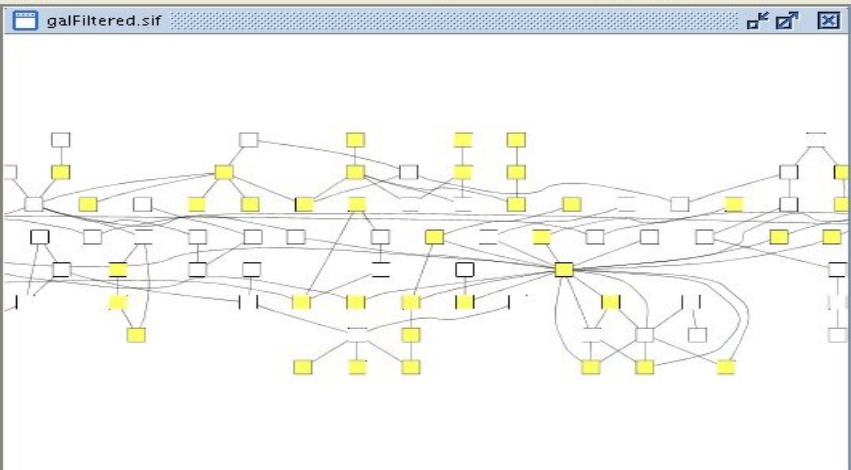
Welcome to Cytoscape 2.4.0 Right-click + drag to ZOOM Middle-click + drag to PAN

File Edit View Select Layout Plugins Help

node Search:

CytoPanel 1

Network	Nodes	Edges
galFiltered.sif	331(0)	362(0)



CytoPanel 2

ID

Node Attribute Browser Edge Attribute Browser Network Attribute Browser

Welcome to Cytoscape 2.4.0 Right-click + drag to ZOOM Middle-click + drag to PAN

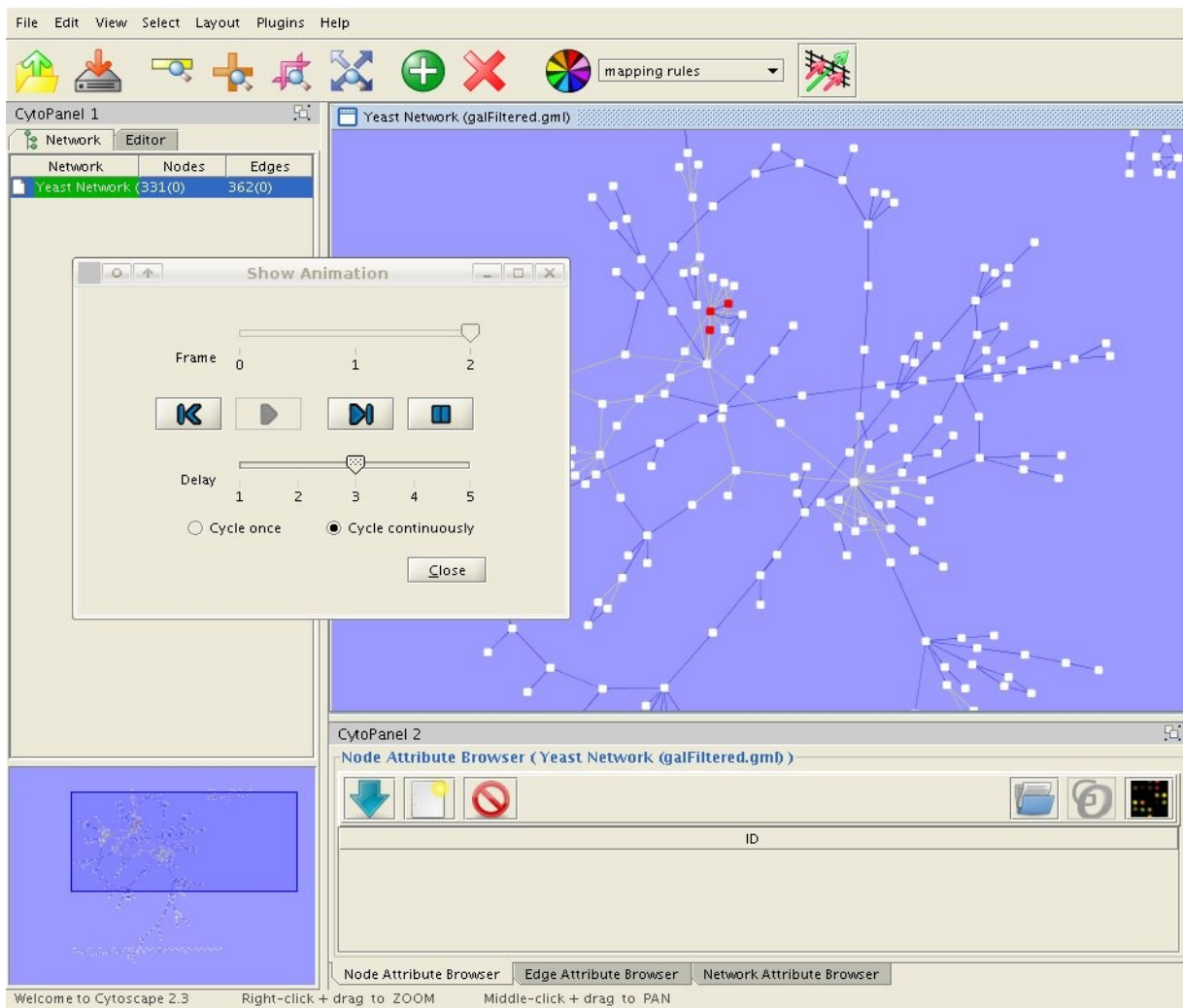
4. Installation

Dependencies :

The *javax.script*-compliant R engine provided by *edu.iastate.metnet.R* (part of CytoScript) requires R (of course) and the JRI library. JRI is included with the rJava R package, which is also required for running the example R snippets in this document.

Environment Variables :

Users of all operating systems need to set `R_HOME` to the root directory of your R installation (ie `/usr/lib/R`). The JRI library needs to be on your `java.library.path`. To achieve this, set the following



environment variable (depending on your operating system) to the location of JRI.

Figure 3. 6.4 Interface of animation

Linux: `LD_LIBRARY_PATH`

Windows: PATH

Mac: DYLD_LIBRARY_PATH

Plugin Installation :

Put Dot.jar, FCMPugin.jar, CytoScript.jar and Animator.jar in the Cytoscape Plugins folder.

Acknowledgment

MetNet Tools are one of MetNet projects supported by the National Science Foundation Arabidopsis 2010 grants DBI-0209809 and DBI-0520267. The work is based on FCModeler which is one of MetNet projects. Most contents of Subgraph Creation in this paper is taken over FCModeler Manual.