Dr Tomasz Kurowski
t.j.kurowski@cranfield.ac.uk

(Optional) Cytoscape App Development

Introduction

In this practical we will develop a very simple Cytoscape plugin for network data analysis. Cytoscape plugins (also known as *Apps*) are written in Java and take advantage of the Cytoscape API. Documentation is available at:

https://cytoscape.org/javadoc/current_release/

The Cytoscape API is basically a set of objects and methods that can be used by developers to interact with the data structures of Cytoscape to run analysis programs on the loaded networks.

General note: during this practical we will see several Cytoscape classes and methods, please have a look at the Cytoscape API Java documentation to get detailed information on them.

Getting started

This practical will use Netbeans as an IDE to create the App. We will use Maven to simplify managing dependencies in the project. As Cytoscape Apps are based on the OSGi framework, the best starting point would be to start with an "OSGi Bundle" project template.

We will start with a simple App which will add a menu option to Cytoscape which will display a "Hello World!"-style message when pressed. Create a new project by selecting **Java with Maven > OSGi Bundle**. You can name it something like "FirstPlugin". The Group Id is arbitrary (our example will use "bix").

Notice that an **Activator** class has been created for you. We will replace it with a Cytoscape-specific extension. Delete the **Activator.java** file and replace it with a **CyActivator** class. Edit the **CyActivator.java** file as follows:

```
package bix.firstplugin;

import org.cytoscape.service.util.AbstractCyActivator;
import org.osgi.framework.BundleContext;

public class CyActivator extends AbstractCyActivator {
     public CyActivator() {
          super();
     }

     @Override
     public void start(BundleContext bc) {
          // TODO
     }
}
```

In order for Maven to find and download the Cytoscape dependencies that we need, we have to list them in the **pom.xml** file. Add the values listed below to **pom.xml**, in the **<dependencies>** tag, alongside **org.osgi** (you can also copy the below from a text file on Canvas):

```
<dependency>
  <groupId>org.cytoscape</groupId>
  <artifactId>application-api</artifactId>
  <version>3.9.0</version>
</dependency>
<dependency>
  <groupId>org.cytoscape</groupId>
  <artifactId>model-api</artifactId>
  <version>3.9.0</version>
</dependency>
<dependency>
  <groupId>org.cytoscape</groupId>
  <artifactId>service-api</artifactId>
  <version>3.9.0</version>
</dependency>
<dependency>
  <groupId>org.cytoscape</groupId>
  <artifactId>viewmodel-api</artifactId>
  <version>3.9.0</version>
</dependency>
<dependency>
  <groupId>org.cytoscape</groupId>
  <artifactId>vizmap-api</artifactId>
  <version>3.9.0</version>
</dependency>
<dependency>
  <groupId>org.cytoscape</groupId>
  <artifactId>presentation-api</artifactId>
  <version>3.9.0</version>
</dependency>
<dependency>
  <groupId>org.cytoscape</groupId>
  <artifactId>work-api</artifactId>
  <version>3.9.0</version>
</dependency>
```

We should also provide Maven with the locations of Cytoscape repositories. Open the **pom.xml** file and add the following inside the **<project>** tag:

```
<repositories>
  <repository>
    <id>cytoscape_snapshots</id>
    <snapshots><enabled>true</enabled></snapshots>
    <releases><enabled>false</enabled></releases>
    <name>Cytoscape Snapshots</name>
    <url>https://nrnb-
  nexus.ucsd.edu/repository/cytoscape_snapshots/</url>
  </repository> <repository>
  <id>cytoscape_releases</id>
  <snapshots><enabled>false</enabled></snapshots>
  <releases><enabled>true</enabled></releases>
    <name>Cytoscape Releases</name>
    <url>https://nrnb-nexus.ucsd.edu/repository/cytoscape_releases/</url>
  </repository>
</repositories>
```

Another thing that needs changing is the entry point of our bundle, since we replaced the **Activator** class with **CyActivator**. Find the **<Bundle-Activator>** property and change its value to use **CyActivator**.

Optionally, you can also change the contents of the **<name>** tag. By default, it will be something like "**FirstPlugin OSGi Bundle**". This will be the name of your App shown by Cytoscape.

We have set up the most basic possible App, which could be installed but does nothing. You can repeat the above steps when you create another App. Now to add some functionality!

Hello world

Now create a new class called **MenuAction** within the same package. We will use it to set the position of our plugin in the Cytoscape interface and make it perform some action. Edit **MenuAction.java** as follows:

```
package bix.firstplugin;

import java.awt.event.ActionEvent;
import javax.swing.JOptionPane;
import org.cytoscape.application.swing.AbstractCyAction;
import org.cytoscape.application.swing.CySwingApplication;

public class MenuAction extends AbstractCyAction
   { CyApplicationManager cyAppManager;

   MenuAction(CyApplicationManager cyAppManager) {
        super("My first app");
        setPreferredMenu("Apps");
        this.cyAppManager = cyAppManager;
   }
```

```
    @Override
    public void actionPerformed(ActionEvent e) {
      String msg = "Congratulations, this is your first app!";
      JOptionPane.showMessageDialog(null, msg);
    }
}
```

MenuAction sets our plugin title to "**My first app**" and adds it to the "**Apps**" menu of Cytoscape. Details on the constructor of **AbstractCyAction** can be found in the API documentation, for now you can consider this as a template for your plugin actions.

Finally, **actionPerformed** provides the code that is executed whenever the plugin is invoked by clicking on it from Cytoscape. In this case a dialog box is opened to display a celebratory message.

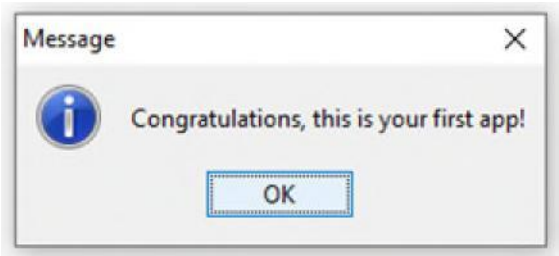
With our action ready, we can register it in the **start** method of our **CyActivator**. Replace the TODO placeholder with:

```
CyApplicationManager cyAppManager = getService(bc,
                                CyApplicationManager.class);
MenuAction menuAction = new MenuAction(cyAppManager);
        registerService(bc, menuAction, CyAction.class,
        new Properties());
```

This should register and activate the MenuAction class.

At this point we can compile the project and create a .jar archive, which should be generated in the **target** directory (not **dist** like in our previous practicals). Now we can start Cytoscape and install the plugin. This can be done by clicking on the **App Manager** that is located under the **Apps** menu of Cytoscape, and then on "Install from file..." which will open a dialog to choose the FirstPlugin.jar file that you have just compiled.

Once this is done, navigate to the Apps menu and if all went fine we should find there a plugin called "My first app". By clicking on it we invoke its execution and a dialog box should appear.



As the message says, congratulations on implementing our first Cytoscape plugin!

Dr Tomasz Kurowski
t.j.kurowski@cranfield.ac.uk

Our first (second) app

We are now ready to implement a (hopefully) more useful app. Our app will provide some basic information on a loaded network such as the number of nodes of the network, the number of edges, the average number of edges per node and the identifier of the most highly connected node.

Moreover, we will allow a user to select a node and we will provide the in-degree and out-degree (i.e. number of the incoming and outgoing edges of the node).

The first thing to do is to create a panel with all the elements the graphical interface will need. You can do this by adding a new JPanel (with all the necessary labels and elements) to our project which can be called, for example, PluginGUI. An example of the final panel is depicted in Figure 1.

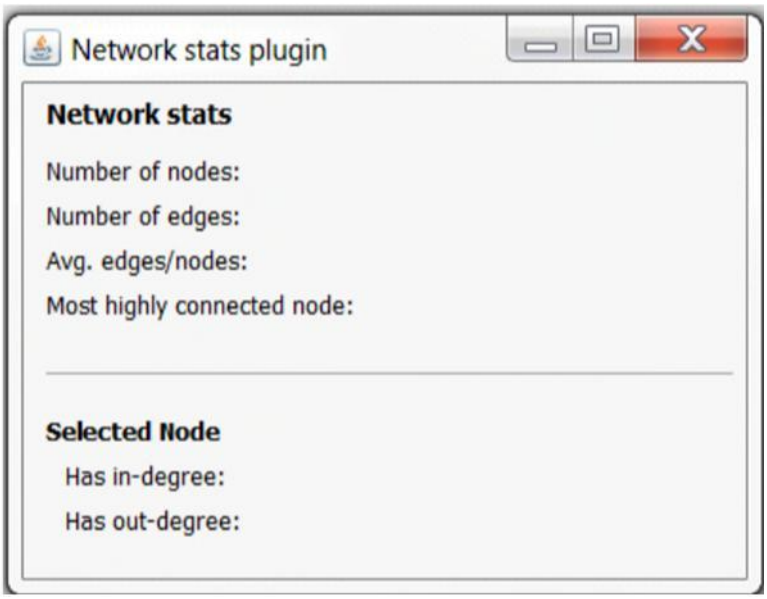


Figure 1: The plugin panel.

It can be useful to add a myGUI panel within the plugin action class (some code is omitted):

```
public class MenuAction extends AbstractCyAction {
    PluginGUI myGUI;
    [...]
}
```

Once the panel has been created it needs to be inserted into a JFrame which has to be displayed whenever the plugin is invoked.

This can be done by creating the frame and setting it to visible within the **actionPerformed** method of the plugin. The action performed method should therefore look like this (some code is omitted again):

```
public void actionPerformed(ActionEvent e) {
    JFrame myFrame = new JFrame();
    myGUI = new PluginGUI();
    [...]
    myFrame.add(myGUI);
    myFrame.setVisible(true);
    [...]
}
```

Now we need to add the logic to the interface, that is, the code computing all the interesting bits and pieces we want to know on our network.

There are many ways to make the JPanel see Cytoscape data; what we will do is to pass the structures needed to the PluginGUI. In particular, two sub-classes of the main class `Cytoscape` are of primary importance:

- `CyNetwork` that contains all the information on the loaded Cytoscape network;
- `CyNetworkView` that contains all the graphical information on the network.

Information on these classes is provided in the core documentation API present at:

https://cytoscape.org/javadoc/current_release/org/cytoscape/model/CyNetwork.html
https://cytoscape.org/javadoc/current_release/org/cytoscape/view/model/CyNetworkView.html

For the time being we will just use `CyNetwork` though.

The PluginGUI definition (file **PluginGUI.java**) will have to be updated in the following manner (note that some code is omitted):

```
import org.cytoscape.model.CyEdge;
import org.cytoscape.model.CyNetwork;
import org.cytoscape.model.CyNode;
import org.cytoscape.model.CyTableUtil;
    [...]
    public PluginGUI(CyNetwork net) {
         initComponents();
         [...]
    }
    [...]
```

The **actionPerformed** method of **MenuAction.java** has to be updated accordingly to get the cyNetwork and pass it to PluginGUI.

```
public void actionPerformed(ActionEvent e) {
    [...]
    final CyNetworkView netView = cyAppManager.getCurrentNetworkView();
    final CyNetwork net = netView.getModel();
    myGUI = new PluginGUI(net);
  [... load and set visible the frame that will contain the PluginGUI...]
}
```

As mentioned before, the logic of the plugin will be implemented within the **PluginGUI.java** in this case. The first part requires to count the number of nodes and edges present into the network. This can be done by using two methods in the `CyNetwork` class:

- `CyNetwork.getNodeCount()`
- `CyNetwork.getEdgeCount()`

whose meaning should be evident from their names.

These methods can be used to retrieve the required information and pass them on to the jLabels that you should have added to the PluginGUI.

To keep the code clean we will implement three methods in the **PluginGUI** class working on the **CyNetwork** class. The method computing the number of nodes is reported below:

```
public int getNodesNumber(CyNetwork net){
    if(net != null) {
        return net.getNodeCount();
    } else {
        return -1;
    }
}
```

A similar method has to be implemented for the number of edges. Another convenient method to be implemented is the **processNet**() that will contain all the calls to the other methods doing the actual legwork. It will also afterwards pass the results to the corresponding labels in the Frame. Some error checking would be good too, for example, in the case there is no network loaded, the number of nodes/edges is set to -1 and the interface should check this and write N/A (i.e. not available) instead of just -1.

Please also note that it can be useful to have the cyNetwork object explicitly defined within the **PluginGUI** class.

```
public class PluginGUI extends javax.swing.JPanel {
    CyNetwork net;
    [...]
}
```

The two numbers computed above can be used to work out the average number of edges per node that can be displayed in the corresponding label.

The next step will be to compute the id of the node having the highest degree of connectivity. We will therefore need to loop through all nodes, get their number of incoming and out-going edges and keep track of which node has the highest number of edges connecting it with other nodes.

A list of all nodes in the network can be obtained by calling the method `CyNetwork.getNodeList()` which returns a `List<CyNode>`. Every element within Cytoscape is identified by a number (the Session Unique ID) that can be obtained -- in the case of CyNodes -- with `CyNode.getSUID()`. All the meta information on the network, like node/edge names or labels, their selection status and so on are stored in a `CyTable` that can be accessed by means of the specific element we are interested in getting the properties of. For example, to get the node name of a `CyNode myNode` within a network `myNet` one would call:

```
myNet.getRow(myNode).get(CyNetwork.NAME, String.class)
```

The list of edges connecting a CyNode myNode can be obtained by means of the call:

```
CyNetwork.getNeighborList(myNode, CyEdge.Type.ANY)
```

which returns a `List<CyEdges>`. Please note that `CyEdge.Type.ANY` can be replaced by CyEdge.Type.INCOMING to get the incoming edges or by `CyEdge.Type.OUTGOING` to get the list of outgoing edges.

An example of method to compute the nodeID of the most highly connected node of the network is therefore the following:

```
public String getTopConnectedNode(CyNetwork net) {
    int maxConn=-1;
    String nodeName="";
    int totEdges=-1;
    int nCount=net.getNodeCount();
    List<CyNode> nodesList=net.getNodeList();
    for(int i=0; i<nCount;i++){
   totEdges=net.getNeighborList(nodesList.get(i),CyEdge.Type.ANY).size();
         if(totEdges > maxConn){
           CyNode myNode = net.getNode(nodesList.get(i).getSUID());
            maxConn = totEdges;
             nodeName = net.getRow(myNode).get(CyNetwork.NAME, String.class);
         }
    }
    return nodeName;
}
```

The final step of this plugin is to compute the in-degree and out-degree of a selected node. As anticipated earlier, information on selected nodes is present in the `CyTable` associated with the nodes of the network. The list of selected nodes of a network `net` can be obtained with:

```
List<CyNode> selectedNodes =
CyTableUtil.getNodesInState(net,"selected",true);
```

A possible method to obtain the information on a selected node can be the following (that has to be added to **PluginGUI.java** and whose output result has to be linked to the corresponding labels in the GUI).

```
public String[] getSelectedNodeDegrees(CyNetwork net) {
    //nodeInfo[1] --> NodeName
    //nodeInfo[1] --> in-degree
    //nodeInfo[2] --> out-degree
    String[] nodeInfo={"","",""};
    List<CyNode> selectedNodes =
          CyTableUtil.getNodesInState(net,"selected",true);
    if (selectedNodes.size() == 1) {
        CyNode node = selectedNodes.get(0);
          nodeInfo[0] = net.getRow(node).get(CyNetwork.NAME,
                                             String.class);
       nodeInfo[1]= Integer.toString(net.getNeighborList(node,
                                    CyEdge.Type.INCOMING).size());
      nodeInfo[2] = Integer.toString(net.getNeighborList(node,
                                    CyEdge.Type.OUTGOING).size());
    } else {
        JOptionPane.showMessageDialog(null,
                     "Please select EXACTLY ONE node!");
        nodeInfo[0] = "none";
        nodeInfo[1] = "N/A";
        nodeInfo[2] = "N/A";
    }
    return nodeInfo;
}
```

This ends our programming effort. We can finally build the project as done earlier and install the plugin through the App Manager of Cytoscape.

We can now test the plugin: let us start Cytoscape and click on the plugin name under the **Apps** menu. If we do not load any network then we should obtain a result similar to what displayed in Figure 2, while if we load a network (such as the sample network "sampleNetwork.sif" downloadable from Canvas) and select a node before invoking the plugin, the result should look like Figure 3.

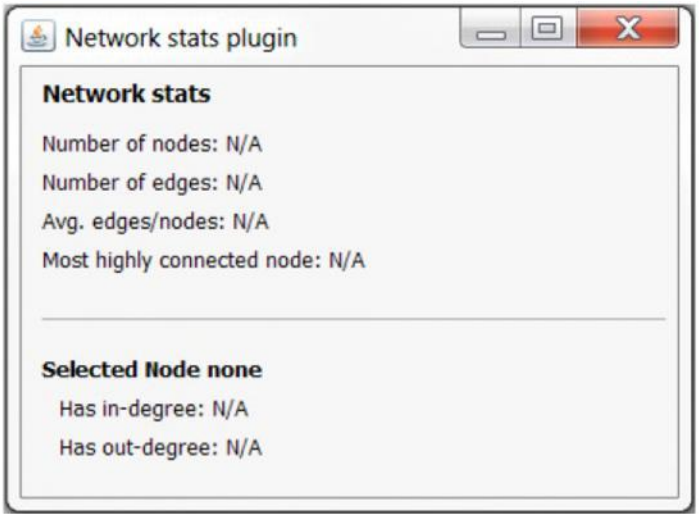


Figure 2: Output when no networks are loaded.

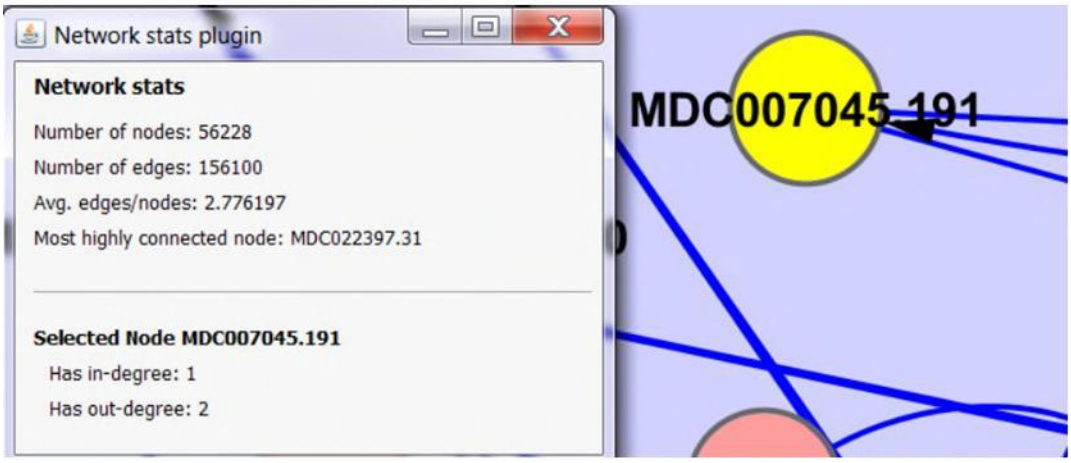


Figure 3: Output when a network is loaded and node MDC007045.191 is selected.

Dr Tomasz Kurowski
t.j.kurowski@cranfield.ac.uk

Extra challenges

The problem

The problem you are going to tackle is the search for the shortest path connecting two nodes in an oriented graph. The user will select two nodes of the network and the plugin will output one of the shortest paths connecting them or will output a message informing that no such path can be found.

The solution

Let us call **N1** and **N2** respectively the two nodes we want to connect with the shortest path. One possible algorithm is to perform a **breadth first search** starting from the N1 (i.e. start from the N1, visit all its children and repeat this recursively paying attention not to visit a node more than once -- that is, **mark visited nodes**) and if node N2 is met in the visit then the shortest path is found. Please note that in this way the shortest path between N1 and N2 is found but that path might not exist or might be longer than the path starting from N2 and reaching N1. This is why the same procedure should be repeated starting from N2 and trying to reach N1.

An example of input and output of the plugin is shown in the figure below:

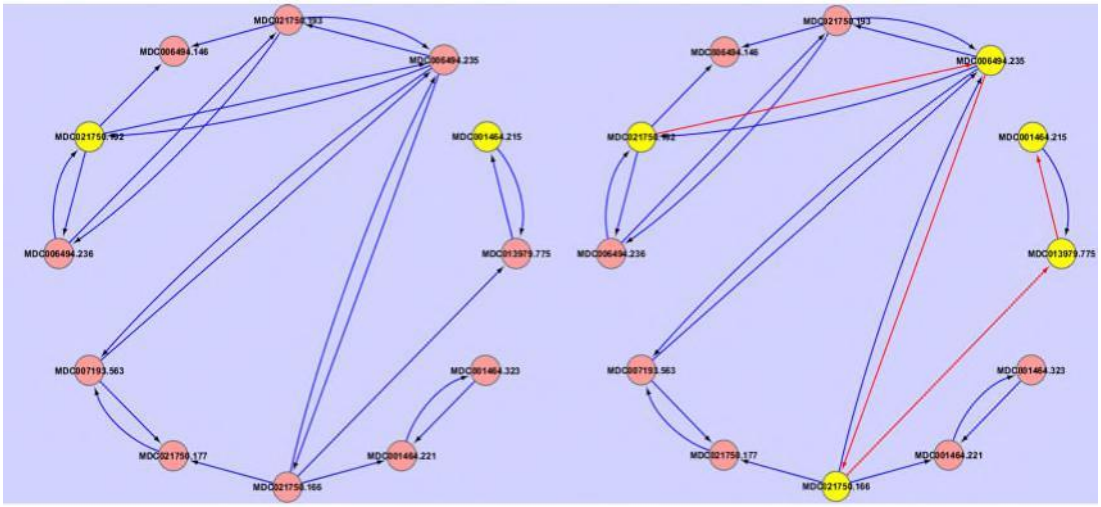


Figure 4: Input (left) and output (right) of the plugin..

The plugin

Below are some methods which you may find useful in your implementation (look them up in the API documentation):

- `CyTableUtil.getNodesInState(net,"selected",true)` : gets the nodes that are currently selected;
- `CyNode.getSUID() : gets the` Session Unique ID of the node;

- CyNetwork.getNeighborList(cNode, CyEdge.Type.ANY): gets the List<CyNode> of all the nodes adjacent to a specified node. It is possible to select in-coming or outgoing edges (with CyEdge.Type.INCOMING or CyEdge.Type.OUTGOING);
- CyEdge.getSource() : gets the CyNode that is the source of the edge (i.e. the node that has that edge as outgoing edge);
- CyEdge.getTarget() : gets the CyNode that is the target of the edge (i.e. the node that has that edge as incoming edge);
- CyNetwork.getConnectingEdgeList(CyNode node1, CyNode node2, CyEdge.Type.DIRECTED) : gets the list of CyEdges connecting node1 to node2. The edge type can be changed;
- CyNetwork.getRow(CyNode/CyEdge).set(CyNetwork.SELECTED, true/false): selects/unselects the node/edge of the network CyNetwork;
- CyNetworkView.updateView() : redraws the graph in the Cytoscape main window.

The idea behind a possible implementation of this plugin is the following:

1. Check if the user has selected exactly two nodes (N1 and N2); if so, go to step 2. Otherwise display an error message and exit;

2. Compute the shortest path between N1 and N2, starting from N1 (which is the current node):

   a. visit all children of the current node and mark them as visited;

   b. for every visited node, add its id and its parent (with the position of the parent in the structure) to a suitable structure (see below);

   c. repeat step a. and b. until node N2 is found (or until no more unvisited children are left), in this case go to step d.

   d. return the element in the structure corresponding to N2.

3. Compute the shortest path between N2 and N1 as at step 2;

4. Travel through the nodes structure starting from N2 up to N1 (by using the parent position information stored) keeping track of the length of the path. Note that while doing this it might be convenient to store the edges met along the path somewhere for later use (i.e. to highlight them and the corresponding nodes).

5. Repeat step 4. starting from N1 up to N2.

6. If path computed at 4. is shorter than path at 5. select nodes and edges present in the path obtained at 4., otherwise select the nodes and edges along the path computed at 5.

If no path connecting N1 and N2 can be found, display an error message.

A possible implementation of the structure described at step 2. b. can be the following:

```
public class pathNode {
    long ID;
    long parentID;
    long parentPosition;

    public pathNode(long nodeID, long parent,
                    int parPosition) {
            ID = nodeID;
            parentID = parent;
            parentPosition = parPosition;
    }
}
```