Social Network Apparatus Specification

August 19, 2010

Gavin Simons

I. Introduction

The primary function of SocialNet is to demonstrate concepts of graph theory, specifically those which pertain to computer science. These concepts focus on the interconnectivity of nodes and the “rich gets richer” theory.

The Social Network Apparatus shows users a variety of graphs to demonstrate social network theory. The types of graphs displayed are generated from pre-existing data sets, randomly generated data, or data generated from Facebook friend relationships. A specific graph is displayed depending on the variables passed into the apparatus by Locus via the option tag, detailed below. Due to the large size of many of the graphs, the graphs are constructed and added to the GUI incrementally. As an Apparatus, SocialNet’s code is separate and stand-alone from Locus’ code. When SocialNet is first started, Locus sends over a series of parameters which will tell SocialNet everything it needs to know in order to start itself up.

This work was supported in part by NSF grant CNS-0938948.

II. Functionality

SocialNet will display a graph while calculating relevant statistical information about it. Currently, SocialNet can only display a graph read in from a text file, a graph generated randomly, and a graph generated from Facebook friends. It should be noted, when making a Facebook graph, SocialNet gets the necessary data by accessing the User’s Facebook page and running a specific Facebook Application. This application is separate from SocialNet and is discussed in the SocialNet Facebook App Specification.

III. User Interface

The GUI consists of a text area where relevant data about the graph is displayed, a frame where the graph itself is displayed, and a series of buttons. When the graph is displayed, the buttons are along the left, and the text area is along the bottom. These buttons allow for displaying/hiding the node labels, for saving the graph, for activating the Help window, and for searching for a specific node. Also, at the bottom of the display, there is a text area where statistical information about the current graph will be displayed. In the area where the graph is displayed, the buttons are along the left side of the display, the text area is on the bottom of the display.

The only part of SocialNet which the user interacts with is the window where the graph is displayed. The user cannot choose which graph is displayed if SocialNet is accessed from a Lab, since the graph is chosen by the Lab in Locus. The only things the user can control in this situation is a series of buttons on the left side of the display.

By default, the user is always in navigation mode. In navigation mode, the user can navigate and manipulate the graph as they see fit. They can zoom in and out of the graph using the scroll wheel, select specific nodes, and move around the graph. When a node is selected, its label becomes visible if it isn’t already and becomes highlighted. Also, the labels of the nodes directly adjacent to the selected node become visible. If another node is selected, the node labels that were rendered visible from the previously selected node are rendered invisible.

The button “Display Labels,” displays the labels does just that, it displays the labels of the nodes of the graph if the User desires. The “Display Labels” button is essentially a toggle. When pressed, it will either have the node labels appear or hide them. The labels themselves will appear text near each node, and will move around with the node if the node is moved by the user.

When searching for a specific node, the user clicks on the “Search” button, and a pop-up frame will appear. In this frame, the user can enter in the name of the node which they want to search for. If the
node exists in the graph, the node is selected, its label will become visible, and the labels of its neighbors will be made visible.

The “Save” button saves the graph as a .png file which the user can access at a later time to see what the graph looked like. Only the graph and text area are put into the .png file. The “Snapshot” button will take an image of the graph and text area, then places it into the Locus notebook display. This will let the user store a series of images of SocialNet graphs in the Locus notebook display, allowing for quick and easy reference.

Like in most other programs, the “Help” button brings up the Help menu. Here, the user can get an explanation on all the functions of the SocialNet Apparatus. The Help menu lacks a search function and just uses a link hierarchy.

IV. XML Specification

The Social Network Apparatus needs to know which graph to display, and this information will be passed from LoCuS (which would read it from the Lab XML file) as a DOM node (“<options>” tag). The DOM node would contain all the relevant information – particularly, the type of graph it’s going to be (existing (<ExistingGraph>), random (<RandomGraph>), Facebook (<FacebookGraph>)), name of the graph file to be read in (if required), chunk-size in which graphs should be read in (an integer, chunkSize = “20”, default of 20) and an integer representing the number of nodes if the graph is to be generated randomly (numNodes = “10”). For RandomGraphs, there is an optional tag for increasing the graph’s connectivity (moreNodes = “10”) which takes in an integer. For a FacebookGraph, only the chunkSize = “10” must be specified, as the user will be prompted for login information by the program. There is also an optional tag for if the text area isn’t to be displayed (showStats = 'false', the default is false). This might come into play if a lab writer wants the user to make a deduction about a graph without the benefit of knowing the calculated statistics.

Example XML:
<options>
   <ExistingGraph showStats="true"/>
   <graph path = "rex.sn"/>
   <graph path = "ExampleGraph.txt"/>
</ExistingGraph>
</options>

Example XML:
<options>
   <RandomGraph chunkSize="10" numNodes="50" showStats="false"/>
</options>

Example XML:
<options>
   <FacebookGraph chunkSize="10" showStats="false"/>
</options>

V. Snapshot Functionality

The Snapshot of a SocialNet consists of the displayed graph and the text area, with the text area located directly under the graph. This way, the user will be able to see and make comparisons between various graphs and their statistics easier. However, if SocialNet was instructed to not show the statistics, the text area will not be in the snapshot.

VI. Implementation Details
After the SocialNet is first initiated, the graph is generated in chunks, which will let the display be updated in steps as particularly large graphs are being constructed. As each “chunk” is generated, it is passed to the display, which then updates. That “chunk” gets added to the display and the user sees the new nodes get added. This process repeats until the entire graph has been rendered in the GUI. This will prevent the user having to wait for an extended period of time where nothing happens in the program while a large graph is being read in.

To implement SocialNet, it is suggested that a `Generator` super class is used. `Generator` will build the various graphs incrementally, and having it be a super class makes it easier for both organization and if you want to add in new types of graphs later on. It is also suggested that the actual graph generation take place in a separate thread. This will help keep the program running smoothly and avoid any “jerky” incremental loading in the GUI.

The basic subclasses of `Generator` include one that reads in graphs from text files, one that generates graphs randomly, and one that makes a graph from a Facebook user’s friends. To make a graph from the user’s Facebook friends, its suggested that a Facebook Application is made which SocialNet can interface with. Facebook changes its code often, making any code designed to log into Facebook and crawl it difficult to keep functional. It is also suggested that it be possible for multiple `ExistingGraphs` to be generated from a single call from Locus. This will make it easier for the user to compare multiple pre-existing graphs side-by-side.

`Generator` should be the “Producer” for `TeachingGraph`, which will hold the graph data structures, “Consume” the data produced in `Generator`, and calculate the statistics of the graphs. It is suggested that Jung¹ be used to calculate the statistics of the graphs. It has a wide array of tools specifically designed for calculating various things about sets of data. Good statistics to be calculated for a graph are the diameter, global clustering coefficient, average local clustering coefficient, number of connected triplets, and number of closed triplets. `TeachingGraph` is then in turn “Observable” by the GUI class, `MainFrame`. It is suggested that Prefuse² be used to visualize the graph data, as it is well suited for handling and displaying large data sets. A more thorough description of the GUI layout can be found above in section three.

When implementing SocialNet, it is necessary to keep in mind that it won’t be talking much with Locus. Locus will pass in the variables SocialNet needs, then SocialNet will have to handle everything else on its own. For more information on how an Apparatus should be made and how they work, see the Apparatus Specification, another manual.

VII. Coupling to other Apparatuses

While other Apparatuses are able to communicate with each other, SocialNet does not need to do so.

Footnotes
1. The Jung library and API can be found at [http://jung.sourceforge.net/](http://jung.sourceforge.net/)
2. The Prefuse library and API can be found at [http://prefuse.org/doc/](http://prefuse.org/doc/)