15-112: Fundamentals of Programming and Computer Science
Project Proposal

Michael Rosenberg
Kosbie, Section J
mmrosenb@andrew.cmu.edu
April 15, 2015

The Problem

Network Science, according to `www.network-science.org`, is "The research of complex networks and systems"[1]. In my personal interpretation, network science uses the fundamental concepts of mathematical graph theory to study applied problems in other fields. Network science contributes to many fields because it caries the surprising power of being able to abstract complex systems into simple mathematical relations; this abstraction allows network science to contribute many significant implications in academic research. Fields such as computer science, statistics, biology, physics, sociology, and economics have all used tools and ideas in network science to study topics that range from the structural hubs of the internet to the clique behavior of social groups in society.

Over the last 30 years, many modules and packages have been developed to assist researchers in the study of networks. Some of these packages are full-fledged executable programs, and some of these packages are libraries that are accessible via a programming language. However, these packages have major accessibility issues. Many of these packages demand a programming background that several researchers in academia don't have and don't have the time to learn. Some of these packages feature clunky interfaces for representing graphs that can be tedious for researchers that are not as familiar with the data structures that represent networks. What is likely the biggest issue is the fact that some of these packages don't have efficient ways to extract metrics on the networks; hence, it is left to the researcher in these instances to write the metric builders themselves, which seems discouraging for researchers who may be interested in studying networks. This all contributes to a huge barrier of entry in network science, which is not a good thing to have in a burgeoning field.

[1]Scholz, Matthias. "Network Science." Network Science. N.p., n.d. Web. 13 Apr. 2015. <http://www.network-science.org/>.

Hence, there is a demand for a network package that is:

- Accessible to non-programmers.
- Easy to use for non-quantitative persons.
- Has efficient metric builders already implemented.

Generally, the problem I am trying to address is "**How do we make a more accessible package for studying networks?**"

The Solution

My solution is to create a Networks Graphical User Interface (GUI). The overall idea of the Networks GUI is to have an area where the user can build the network using tools and labels to clearly define a network how they want it to be, and then to present the user with metrics and information on the network as the user builds it. This solution comes in three parts: A front-end, A back-end, and an application. The front-end is the interface where the user can build the network by accessing certain buttons and certain modes to adjust the nodes and edges of the network. This front end also contains a report of the metrics on the network and a selected subset of nodes in the network if the user chooses to do so. The back-end is the area where I will have implemented certain algorithms to extract certain metrics on the network. The application is a webscraper for Wikipedia that provides the user with a sense of the powerful implications of networks. While I have made some progress on the front-end of the project (see code artifact `termProject.py`), I plan to work mainly on the back-end and the application as early as possible because these parts will take up the bulk of my time. I plan to finish up the front-end after I finish the other two portions of the project.

The Front-End: The Interface

The interface will be established through object-oriented design. My four superclasses are:

- Node, the initial building block of a network.

- Edge, objects that connect nodes.
- Button, which give us particular rules as to what we can edit given which button is selected
- Animation, which contains information about the entirety of the interface.

A node will contain this information:

- Its incoming and outgoing adjacency lists (via its incident edges).
- Its color, shape, size, and location on the screen.
- How to draw itself and whether or not a given mouse click is on the node.
- If the node is selected or not.
- A dictionary of characteristics associated with the given node.

Edges will have two classes. The superclass edge will be an undirected edge, which means that there is no specified direction on the edge (an edge connecting node u to node v is also an edge connecting node v to node u). A directed edge will inherit from this superclass edge, and this edge does specify a direction (a directed edge goes from node u to node v, and is not the same as an edge going from node v to node u). The general plan for the spec of an edge will be that it contains this information:

- The two nodes that it is attached to.
- the color, size, and location on the screen.
- If the edge is selected or not.
- How to draw itself and whether or not a given mouse click is on the edge
- A dictionary of characteristics (including weight, an important property of edges) associated with the given edge.

Buttons will be continuously subclassed given the commonalities in how a button can draw itself. The general information attached to a button will be:

- Its size, shape, color, and location.
- The name associated with that button.
- If the button is selected or not.
- Whether or not a given click of a mouse is in the button.
- A set of rules associated with mouse clicks, drags, and releases and with key presses that will define what will happen on a given event if the button is selected.

The Animation will be built in the framework of the class-based eventBasedAnimation. This animation will contain information such as

- a universal storage Struct whose attributes can be pointed to by other objects in the interface
- A node list, edge list, and button list for containing all other objects in the animation.
- Information pointing to the metrics calculated from the given node list and button list
- A method to draw everything in the list and the metrics in a select portion of the screen
- A set of event-based methods that point to event-based methods in particular buttons given which button is selected.

The main graphics engine used to generate this interface will be Tkinter.

The Back-End: Metric Building and Algorithms

The first thing I will do will be to use my node list and my edge list to produce an adjacency matrix where

$$A_{i,j} = \begin{cases} 1 & \text{if there is an edge connecting node } i \text{ to node } j \\ 0 & \text{otherwise.} \end{cases},$$

Where a given undirected edge would represent both a connection from node i to node j and a connection from node j to node i. I will store this matrix in this format using the package `numpy`. The metrics that I will be using are laid out in the book *Networks: An Introduction* by Mark Newman[2]. I plan to produce at least 8 metrics based off a given network. A few examples of these metrics include:

- Eigenvector centrality for the given nodes in a network, which is a measurement of how central a given node in a network is given the centrality of its neighboring nodes. The way we tend to calculate this centrality is by initially setting the centrality of each node in our graph to $x_i = 1$, and then given a sequence of nodes, calculate each node's centrality via

$$x_i' = \sum_j A_{ij} x_j.$$

- Calculating the clustering coefficient for each individual vertex, a measure of local clustering which is defined to be

$$C_i = \frac{\text{(number of pairs of neighbors of } i \text{ that are connected)}}{\text{(number of pairs of neighbors of } i\text{)}}$$

- Cosine Similarity, which is a measure of how similar in terms two nodes are in a network (named for how similar it is to the equation $cos(\theta) = \frac{\vec{x}\cdot\vec{y}}{|\vec{x}||\vec{y}|}$), which is defined to be

$$\sigma_{ij} = \frac{\sum_k A_{ik} A_{kj}}{\sqrt{\sum_k A_{ik}^2}\sqrt{\sum_k A_{ij}^2}}$$

Many of these metrics require a certain number of advanced algorithms to be properly calculated. For instance, to calculate betweenness centralities for weighted graph, which is a measure of the number of shortest paths from all vertices that pass through a given node, I need to calculate the weights of the paths flowing through each node in a network. Here is some pseudo-code that lays

[2]Newman, Mark. Networks: An Introduction. : Oxford University Press, 2010-03-25. Oxford Scholarship Online. 2010-09-01. Date Accessed 13 Apr. 2015 <http://www.oxfordscholarship.com/view/10.1093/acprof:oso/9780199206650.001.0001/acprof-9780199206650>.

out how this works:

```
def Calc(startNode,graph):
 startNode.d = 0
 startNode.weight = 1
 for node in graph:
  if (node != startNode): node.d = ""
 def furtherDistances(dist):
  #check if any distance d
  counter = 0
  for node in graph:
   if (node.d == dist): counter += 1
  if (counter > 0):
   for node in graph:
    if (node.d == dist):
     for secondNode in node.outAdjacencyList:
      if (secondNode.d == ""):
       secondNode.d = dist+1
       secondNode.weight = node.weight
      elif  (secondNode.d = dist+1):
       secondNode.weight += node.weight
      elif (secondNode.d < dist+1): pass
   return furtherDistances(dist+1)
 furtherDistances(startVertex.d)
```

It is important to note that there is a potential for some of these algorithms to take very long as the network gets bigger. For example, the Ford-Fulkerson algorithm for calculating maximum flow takes average time $O((m+n)m/n)$, where n is the number of vertices and m is the number of edges. This may lend itself to possibly change the way I store adjacency information (either with an adjacency list or temporary queues for algorithms) in order to more efficiently calculate these metrics. I may choose to write an algorithm that detects certain characteristics of a given network and chooses the most efficient data structure for the given network.

The Application: Webscraping Wikipedia

I am planning on implementing a part of my interface to work with a webscraping algorithm for Wikipedia. The algorithm would work like this: Given a starting page (node) on wikipedia and an integer k, build me a network of all paths of length less than or equal to k from this starting page

to pages on wikipedia. Here is some pseudo-code to identify how this would work:

```
def scrape(startPage,k):
 wikiGraph = Graph(nodeList = [], edgeList = [])
 startPageNode = Node(url = startPage)
 wikiGraph.nodeList.append(startPageNode)
 def webCrawl(graph,startPageNode,k,depth = 0):
  if (depth < k):
   for word in startPageNode.url.html.body:
    if (word.tag == "href"): #it's a link
     attachedNode = Node(url = word.url)
     wikiGraph.nodeList.append(attachedNode)
     wikiGraph.edgeList.append(Edge(u = startPageNode, v = attachedNode))
     webCrawl(graph,attachedNode,k,depth+1)
 webCrawl(wikiGraph,startPageNode,k)
 return wikiGraph
```

I will likely make design decisions while building this webscraper to make this process more efficient (given how quickly this network will grow). Given the webscraping framework, I plan to implement this by retrieving HTML via `urllib2` and parsing the HTML via `BeautifulSoup`.

The Modules/Technologies

Besides the modules discussed in class (eventBasedAnimation, Tkinter, random, math, and structClass), I will be using

- `numpy` for matrix-based calculations.
- `urllib2` For getting html from the world wide web.
- `BeautifulSoup` for parsing html taken by `urllib2`.

Literature on Network Science

I do understand the fact that, during this process, there may be several CAs who are not as familiar with network science as I am. Hence, I felt it would be good to attach some suggested reading for the CA who may want to know more about the field. I have opted to not cite the following sources because citing Wikipedia is arguably inappropriate.

- http://en.wikipedia.org/wiki/Network_science is a great "starting node" (sorry, I had to make the pun) for understanding certain network models and the use of networks in various fields. Here is a link to a more descriptive sub-field in network science (http://en.wikipedia.org/wiki/Homophily), and here is a link to amore quantitative theory in network science (http://en.wikipedia.org/wiki/Reciprocity_(network_science)).

- For a very thorough and academic introduction to network science, I often recommend *Networks: An Introduction* by M.E.J. Newman. Newman is one of the leading figures in modern network science, and this book is filled with some of the most up-to-date topics in the field. For a networks book on a more applied topic, I often recommend *Social and Economic Networks* by Matt Jackson. Jackson is one of the leading figures in the burgeoning sub-field of network economics, and he has held major positions at Stanford University and Northwestern University.

- This paper on networks (http://advances.sciencemag.org/content/1/1/e1400005) has recently made a splash in the press

  (http://www.wired.com/2015/02/academic-hiring-uphill-battle/).