ECS253 - Homework 2

Name: Camille Scott
Email: camille.scott.w@gmail.com

May 3, 2016

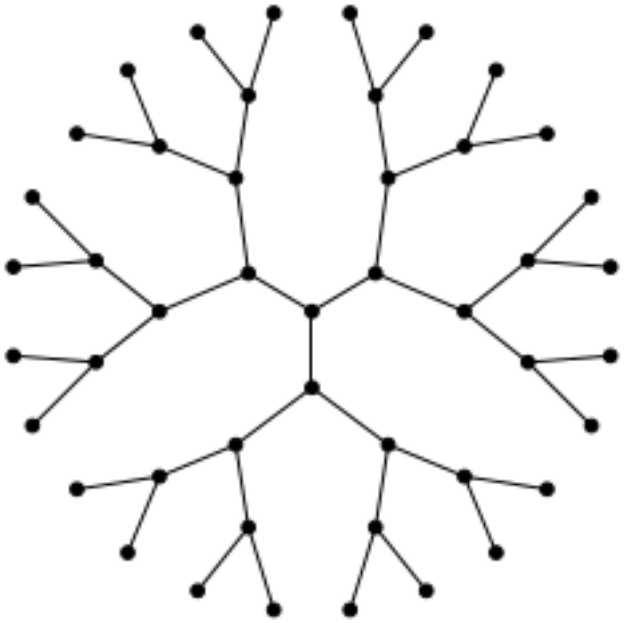
This is the common problem set for Homework 2 from the spring quarter Network Theory class at UC Davis taught by Prof. Raissa D'Souza. The original assignment is at http://mae.engr.ucdavis.edu/dsouza/Classes/253-S16/hw2.pdf. Source code for this notebook is on github at https://github.com/camillescott/ucd-ecs253.

```
In [93]: %pylab inline
         import numpy as np
         import networkx as nx
         import seaborn as sns
         sns.set_style('ticks')
         sns.set_context('poster')
```

```
Populating the interactive namespace from numpy and matplotlib

WARNING: pylab import has clobbered these variables: ['draw', 'record']
`%matplotlib` prevents importing * from pylab and numpy
```

1 The Cayley Tree

A Cayley tree is a symmetric regular tree emanating from a central node of degree k. Every node in the network has degree k, until we reach the nodes at the maximum depth d that have degree one and are called the leaves of the network. The figure below shows a Cayley tree with $k = 3$ with depth $d = 4$.



For a Cayley degree of degree k and depth d calculate:

1.1 Vertices at Exactly Distance 1

Let's start by considering a Cayley Tree at $d = 1$ (which is actually just the star graph). The central vertex, in order to have degree k, must have exactly k nodes attached directly to it. So, there are k vertices at distance 1 in this tree. If we grow the tree to higher d, the central node can have no more vertices attached, as it's already at degree k; we can then conclude, somewhat obviously, that there are k vertices at exactly distance 1.

1.2 Vertices at Exactly Distance 2

Let's consider growing the tree to $d = 2$. Each of the nodes at distance 1 already has degree 1, and so needs to have $k - 1$ nodes attached in order to grow the tree to $d = 2$. As there are k of those, there must then be $k(k - 1)$ nodes at exactly distance 2.

1.3 Vertices at Exactly Distance l

The logic for the general case follows from that for distance 2: if we expand to $d = 3$ we add another $k - 1$ nodes to each leaf, giving us $k(k - 1)^2$, and so on, meaning that there are $k(k - 1)^{l-1}$ nodes at exactly distance l.

1.4 Total Number of Vertices within Distance l

$$n(l) = 1 + \sum_{m=0..l-1} k(k-1)^m$$

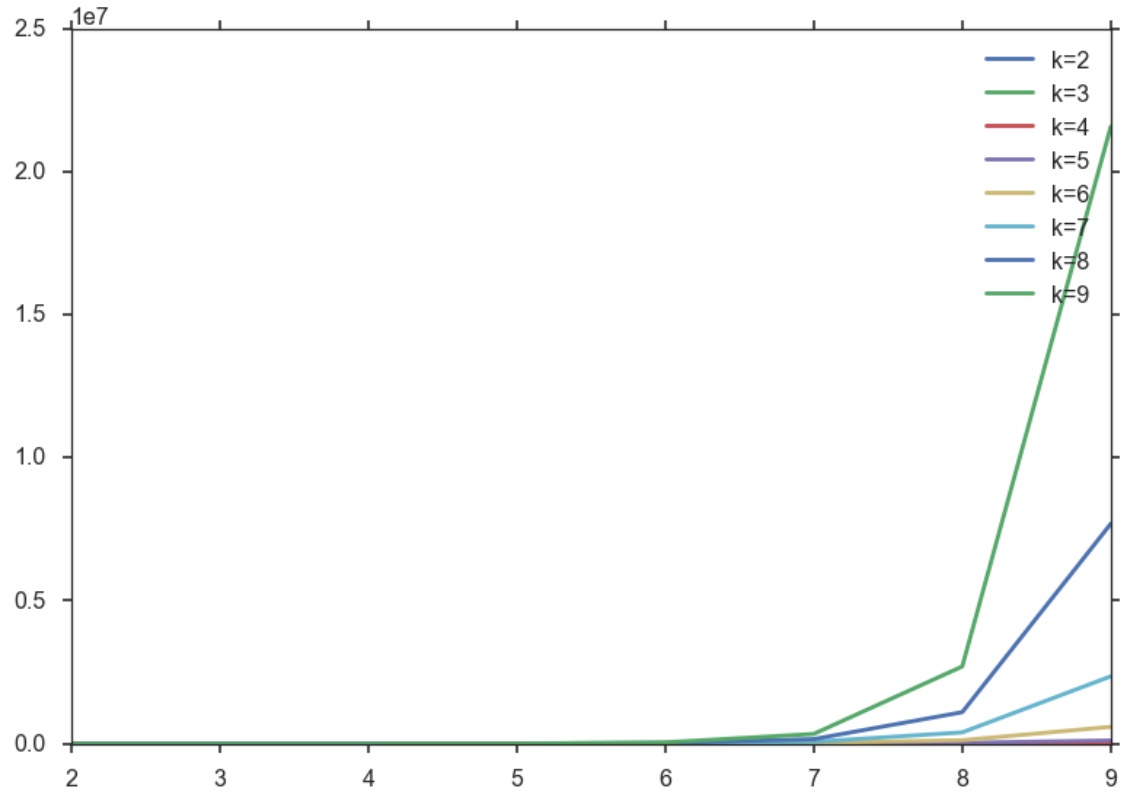
1.5 Small World

We can show the small world structure intuitively and empirically. For one, it's clear that $n(l)$ grows approximately geometrically with d and k. To illustrate this further, I've plotted $n(l)$ for increasing d with a range of k.

```
In [103]: def n_nodes_cayley(d, k):
              n = 1
              for m in range(0,d-1):
                  n += k * (k-1)**m
              return n
```

```
In [104]: for k in range(2,10):
              D = range(2,10)
              plot(D, [n_nodes_cayley(d, k) for d in D], label='k={0}'.format(k))
          legend()
```

```
Out[104]: <matplotlib.legend.Legend at 0x7f7cdb3d3320>
```



It is also clear that the longest shortest path in this graph is always $2d$. With this and the scaling of $n(l)$, then, by approximation, we get that $d \approx log(n)/log(k)$

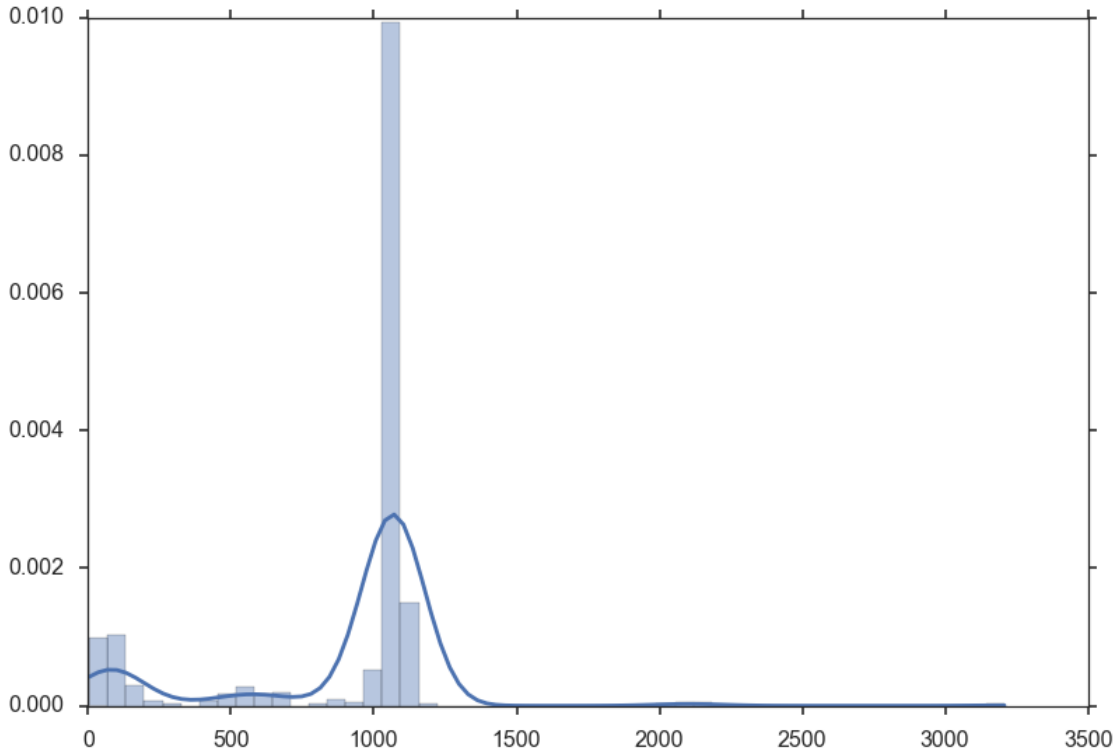
2 Finite size scaling

3 Analysis of a Real-world Network

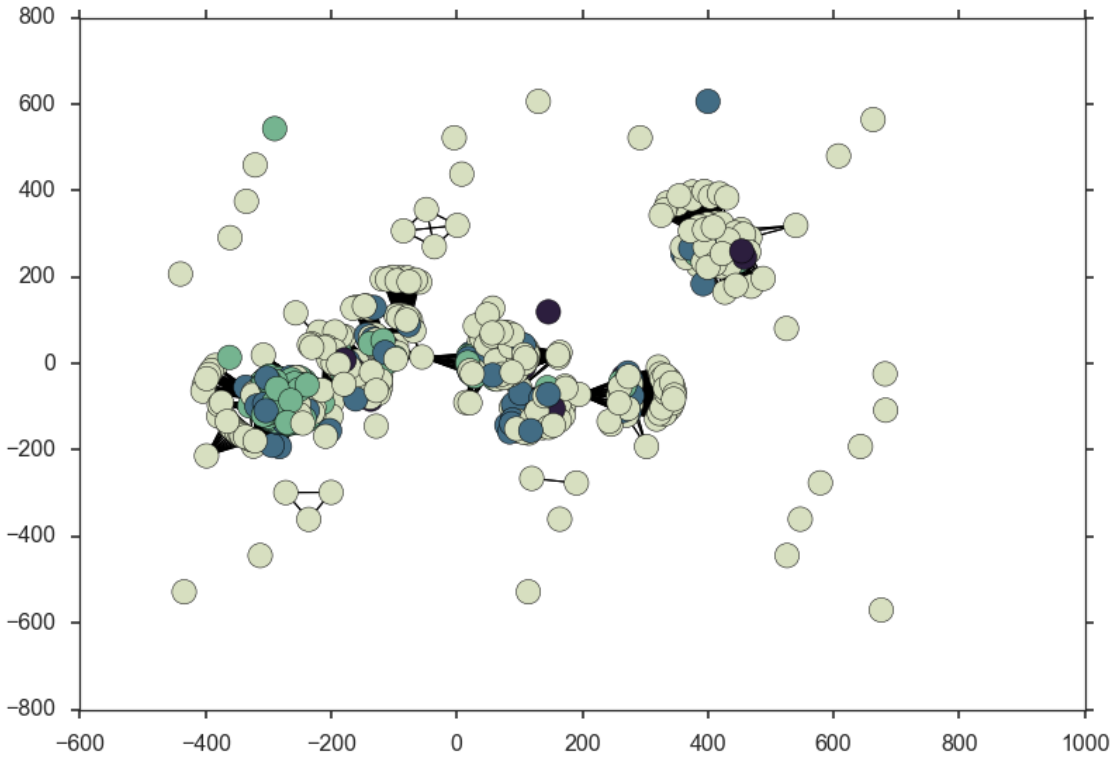
http://www.uniprot.org/uniprot/?query=hox-a&sort=score&columns=id%2Centry%20name%2Creviewed%2Cprotein%20na
id%2Csequence

```
In [81]: import screed
         import pandas as pd
         from networkx.drawing.nx_agraph import graphviz_layout
```

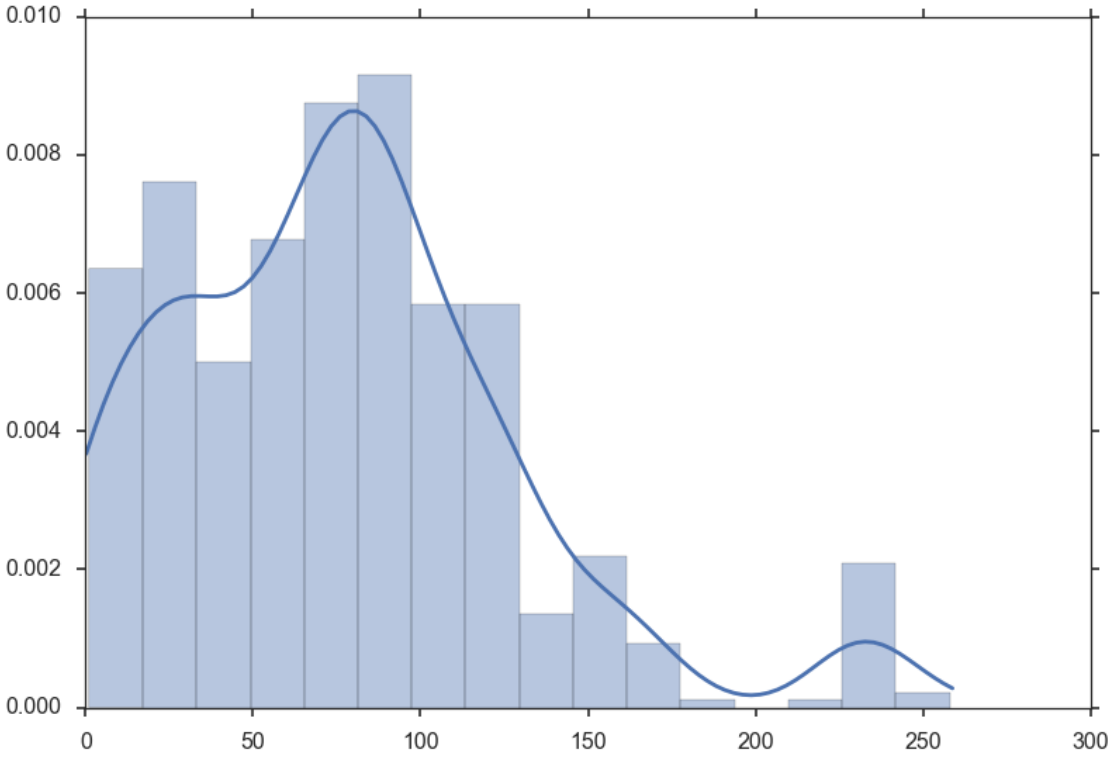
```
In [80]: metadata = []
         for record in screed.open('uniprot-hox-a.fasta'):
             tokens = record.name.split()
             _, entry_ID, entry_name = tokens[0].split('|')
             kvs = tokens[-3:]
             #_, _, species = kvs[0].partition('=')
             _, _, gene_name = kvs[0].partition('=')
             _, _, evidence = kvs[1].partition('=')
             metadata.append({'blast_ID': tokens[0],
                              'entry_ID': entry_ID,
                              'entry_name': entry_name.upper(),
                              'gene_name': gene_name.upper(),
                              'evidence': evidence})
         metadata = pd.DataFrame(metadata)
         metadata.set_index('blast_ID', inplace=True)

In [19]: !makeblastdb -dbtype prot -in uniprot-hox-a.fasta

Building a new DB, current time: 05/03/2016 11:37:15
New DB name:   uniprot-hox-a.fasta
New DB title:  uniprot-hox-a.fasta
Sequence type: Protein
Keep Linkouts: T
Keep MBits: T
Maximum file size: 1000000000B
Adding sequences from FASTA; added 632 sequences in 0.0311999 seconds.

In [20]: !blastp -query uniprot-hox-a.fasta -db uniprot-hox-a.fasta -outfmt 6 -out uniprot-hox-a.self.bl

In [4]: def parse_blast(filename):
            results = pd.read_csv(filename, delimiter='\t',
                                  names=['qseqid', 'sseqid', 'pident', 'length',
                                         'mismatch', 'gapopen', 'qstart', 'qend',
                                         'sstart', 'send', 'evalue', 'bitscore'])
            return results

In [5]: alignments = parse_blast('uniprot-hox-a.self.blastp.tab')

In [19]: def get_graph(data, subset=None, GraphType=nx.MultiGraph):
             if subset is not None:
                 data = data[subset]
             G = GraphType()
             G.add_weighted_edges_from(data[['qseqid', 'sseqid', 'pident']].to_records(index=False))
             return G

In [20]: hoxa_graph = get_graph(alignments)

In [7]: nx.number_connected_components(hoxa_graph)

Out[7]: 2

In [16]: sns.distplot(list(nx.degree(hoxa_graph).values()))

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7cc878c7b8>
```




```
In [23]: hoxa_pruned = get_graph(alignments, alignments.evalue < 1e-30)

In [82]: def plot_degree_dist(G):
             sns.distplot(list(nx.degree(G).values()))

In [88]: def draw(G):
             meta = metadata.ix[G.nodes()]
             cmap = sns.cubehelix_palette(5, start=.5, rot=-.75, as_cmap=True, reverse=True)

             #pos = graphviz_layout(G, )
             nx.spring_layout(G, k=0.9, scale=10.0)

             nx.draw_networkx_nodes(G, pos, node_color=list(meta.evidence), cmap=cmap,
                                    vmin=meta.evidence.min(), vmax=meta.evidence.max())
             nx.draw_networkx_edges(G, pos)

In [89]: draw(get_graph(alignments, alignments.evalue < 1e-40))
```



In [83]: `plot_degree_dist(get_graph(alignments, alignments.evalue < 1e-40))`



```
In [ ]: nx.
```