Social Network Analysis 101 with Python

actually…

Social Network Analysis 101

(with Python and Gephi)

Tonight:

1. What are networks?
2. Network Data
3. SNA with Python Quick Start
4. Why SNA?: Predictive Features of Networks

Tonight:

1. What are networks?
2. ~~Network Data~~
3. SNA with Python Quick Start
4. Why SNA?: Predictive Features of Networks

Goal:

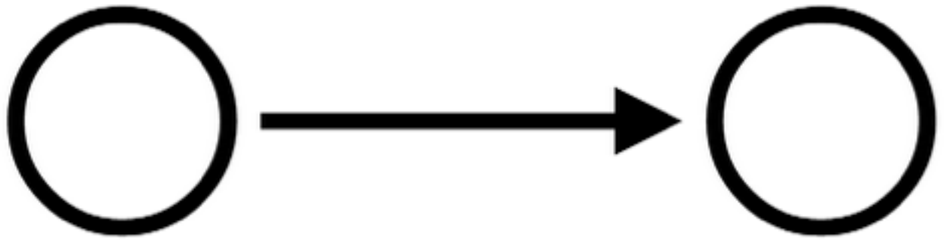
Cliff notes for a semester in computational social science…

“Social network analysis (SNA) is the analysis of social networks.”

-thank you, wikipedia

“Social network analysis views social relationships in terms of network theory, consisting of nodes (representing individual actors within the network) and ties (which represent relationships between the individuals, such as friendship, kinship, organizations, sexual relationships, etc.)”

-that’s better



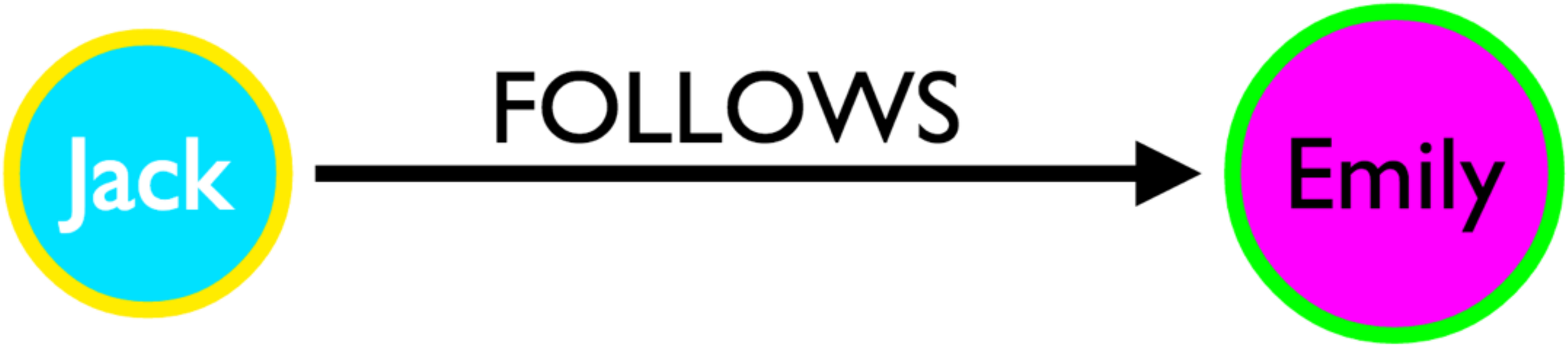
Key and most basic assumption:

People organize into systems (networks).

Therefore, human networks, like many other types of networks, adhere to several predictable patterns.

Networks (graphs) can be directed… e.g. Twitter

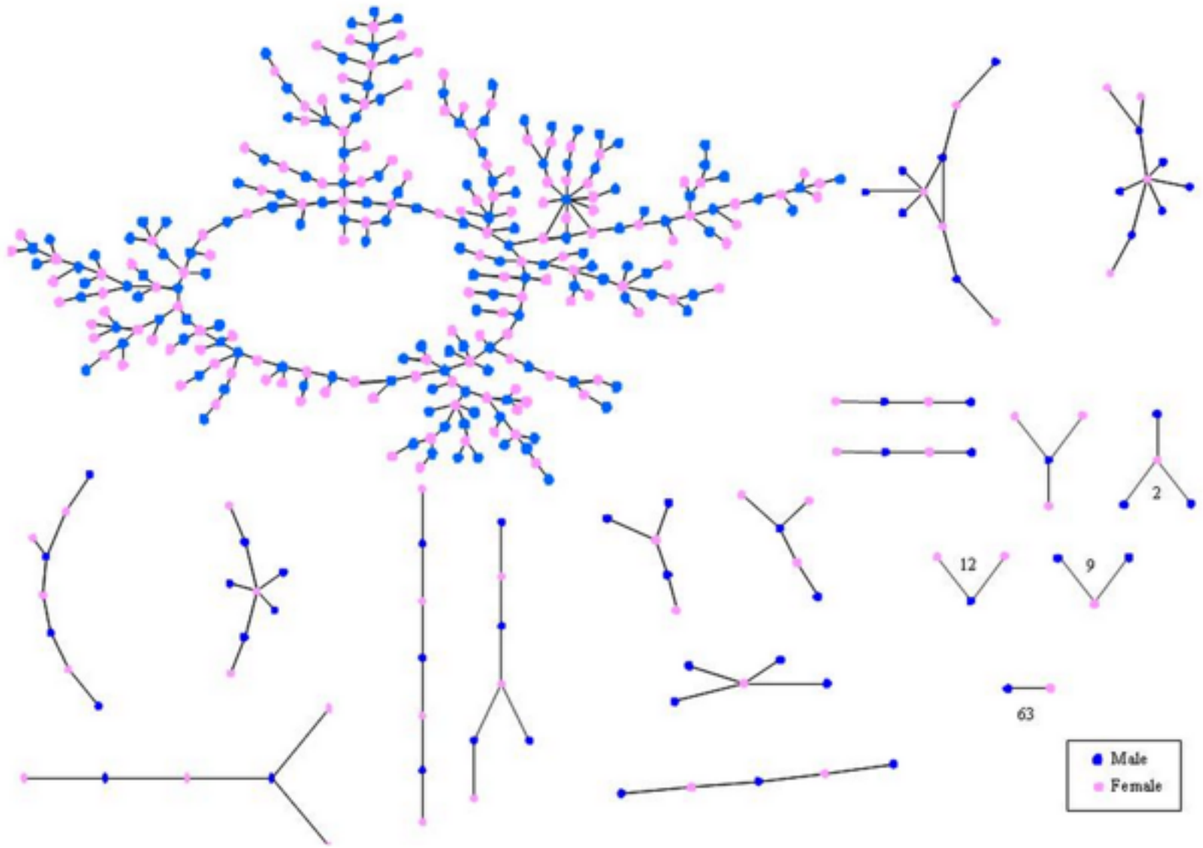
D= (V, A)



Networks (graphs) can be bidirectional or undirected… e.g. Facebook, Linkedin

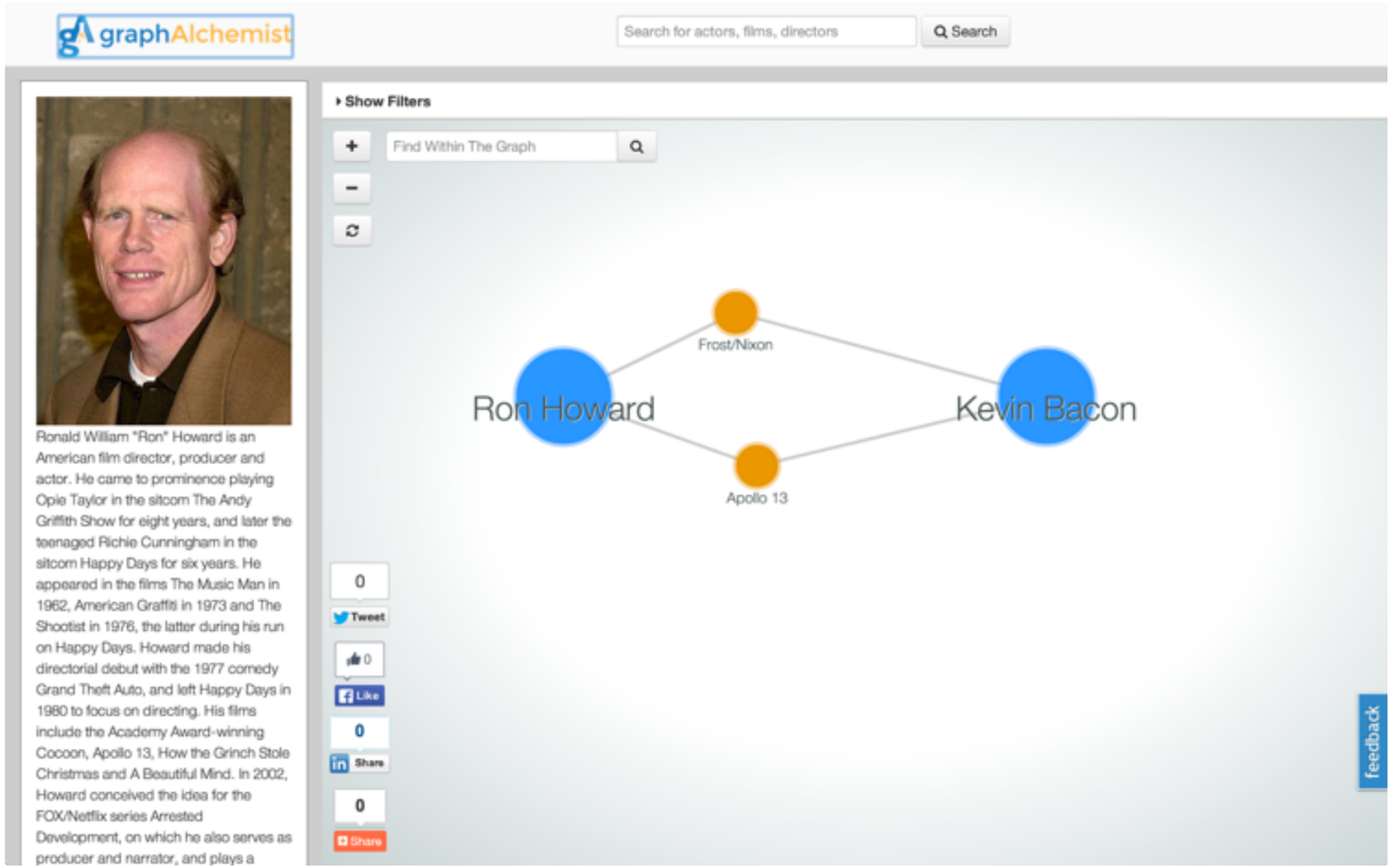
$$G = (V, E)$$

Sexual Network of High School Students



http://www.soc.duke.edu/~jmoody77/chains.pdf

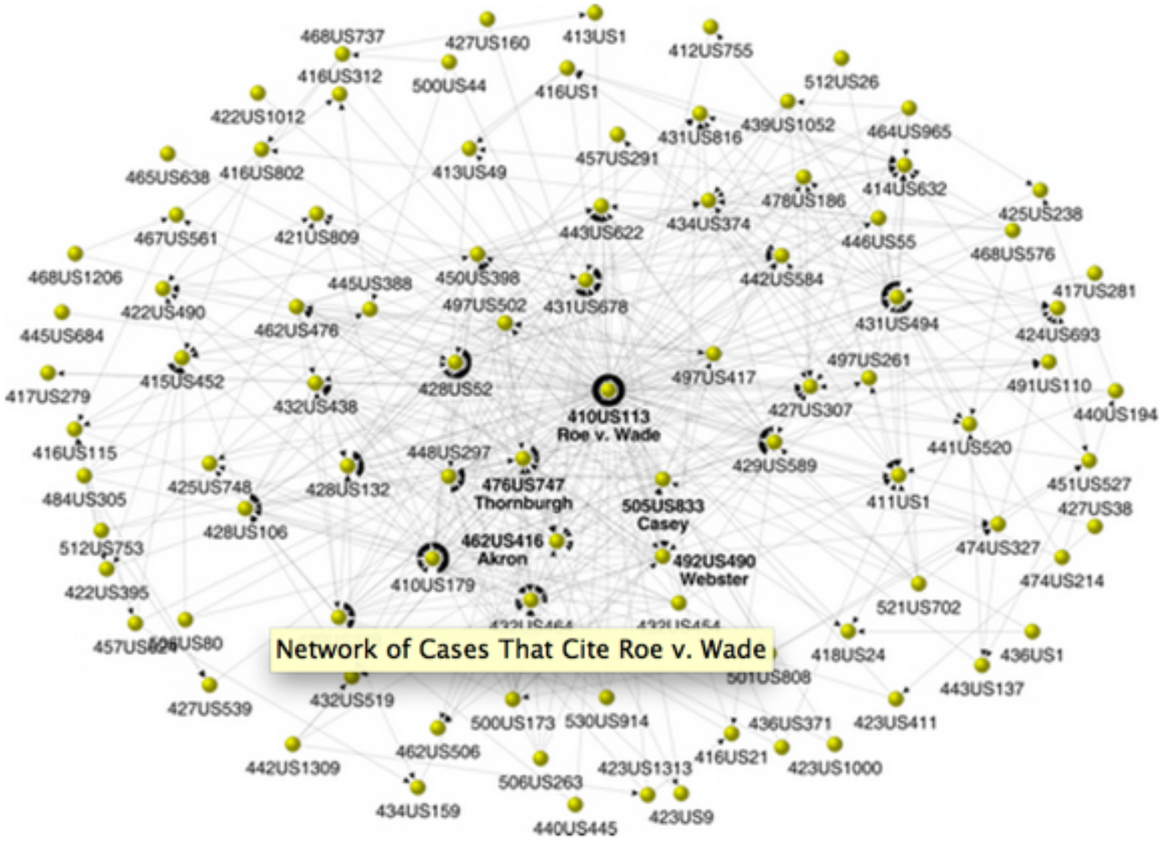
Collaboration Networks



http://movies.graphalchemist.com/movies-profile-id/61016/207116/

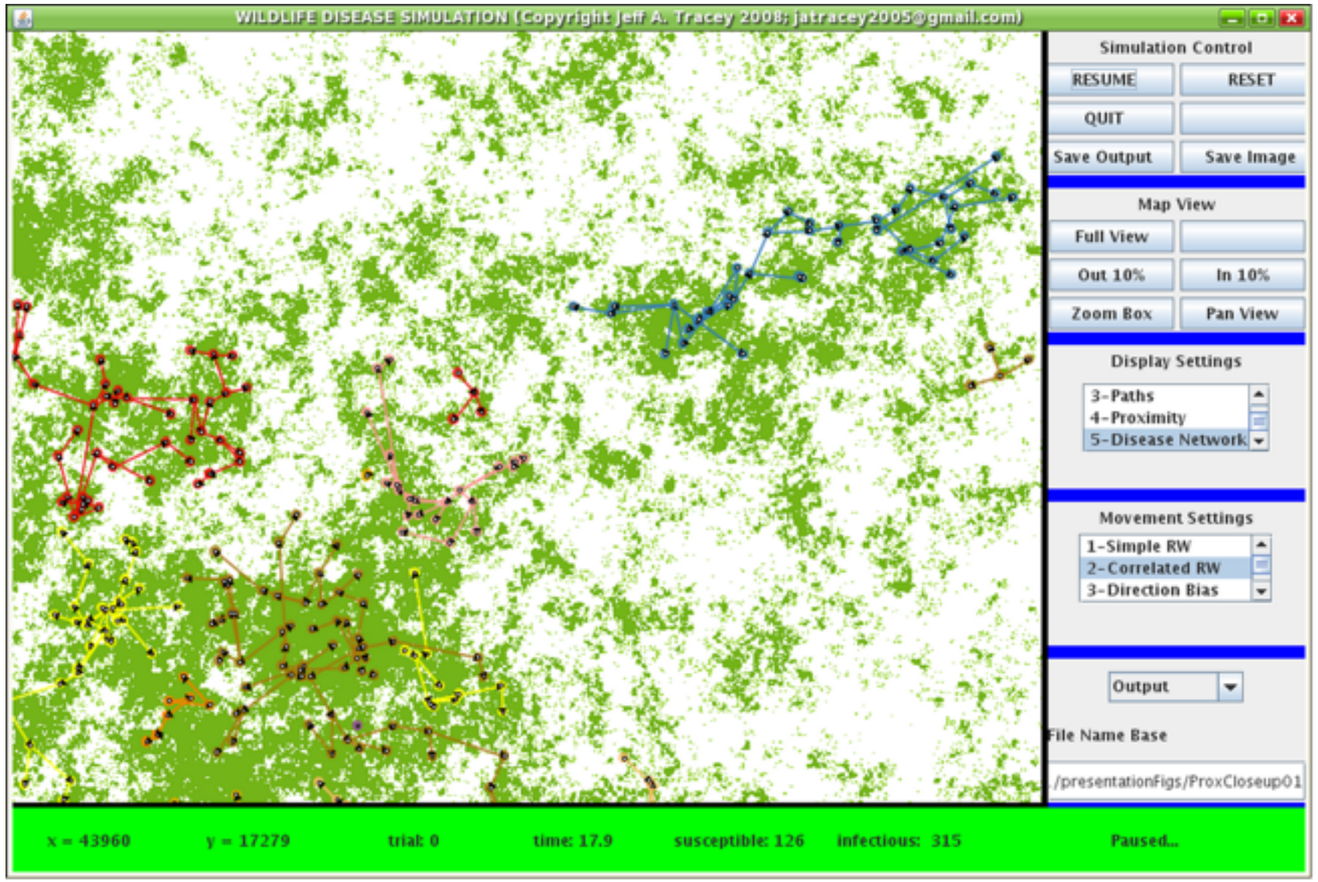
Collaboration Networks

Co-authorship/Citations



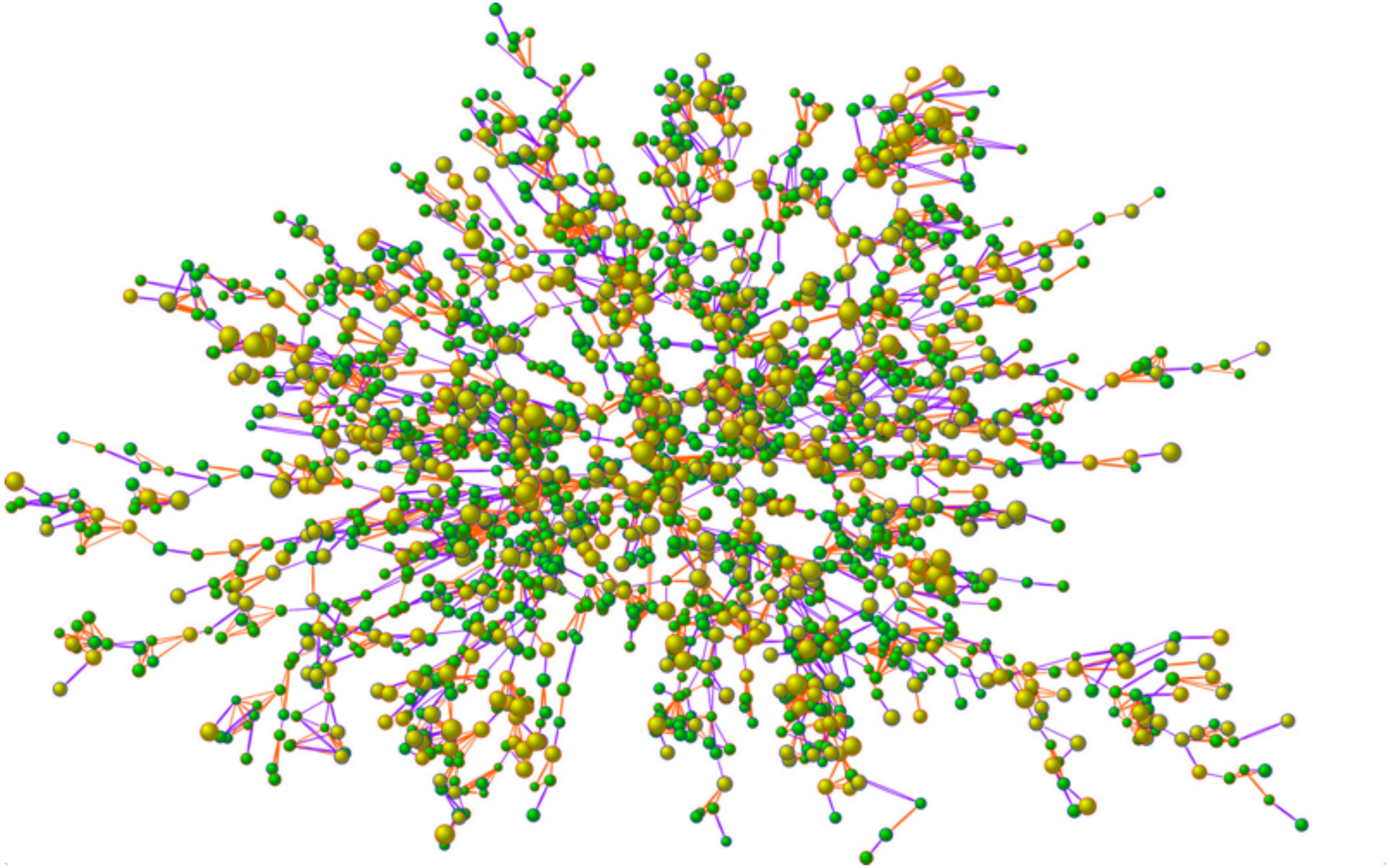
http://jhfowler.ucsd.edu/authority_of_supreme_court_precedent.pdf

Disease Diffusion



http://feline-eid.colostate.edu/eidabm/eid-abm.html

Obesity (and smoking) in Social Networks



Jumping ahead to the end of your first week…

Clone (and star) this repo:

https://github.com/hustonhedinger/SNA101

‘Build’ a Network:

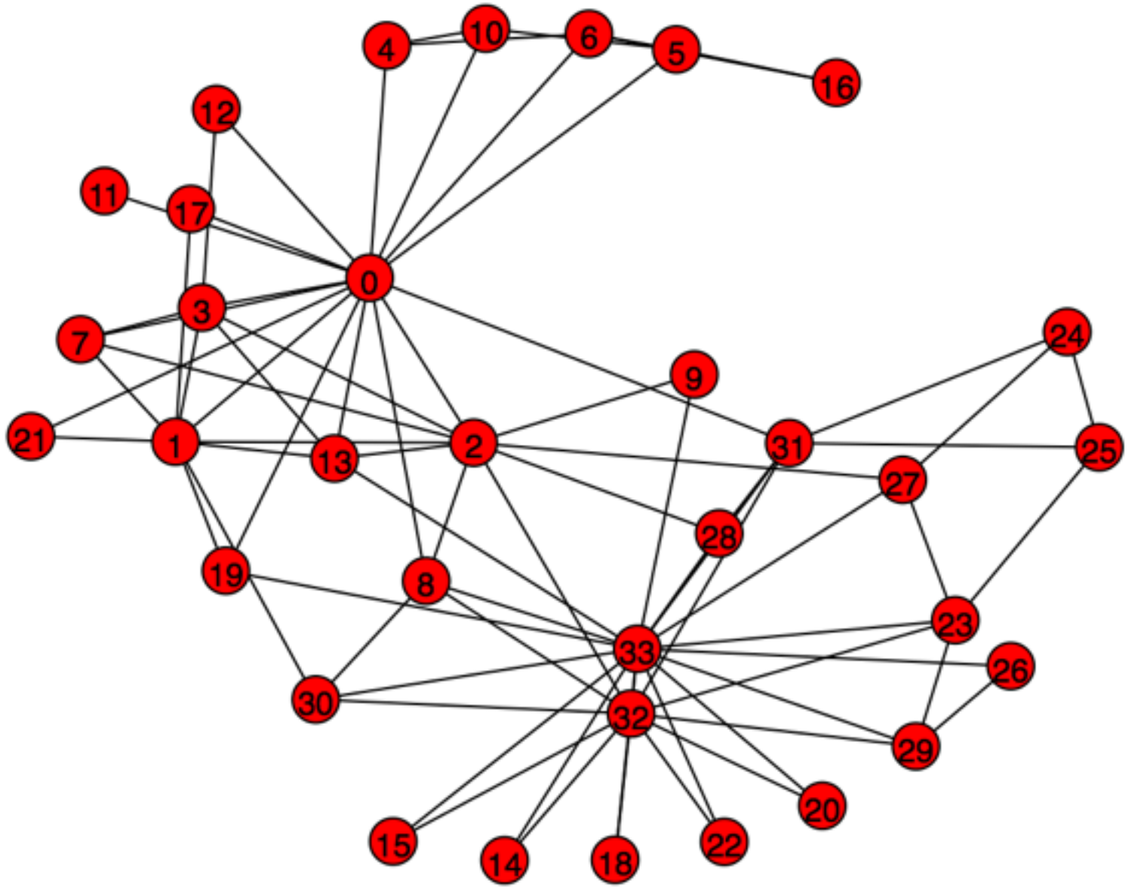
```
In [1]: import networkx as nx

In [2]: G = nx.karate_club_graph()
```

WARNING:

Networkx karate club is indexed with ids starting at 0… the Zachary paper (http://aris.ss.uci.edu/~lin/76.pdf) starts at id 1…

Karate Club:



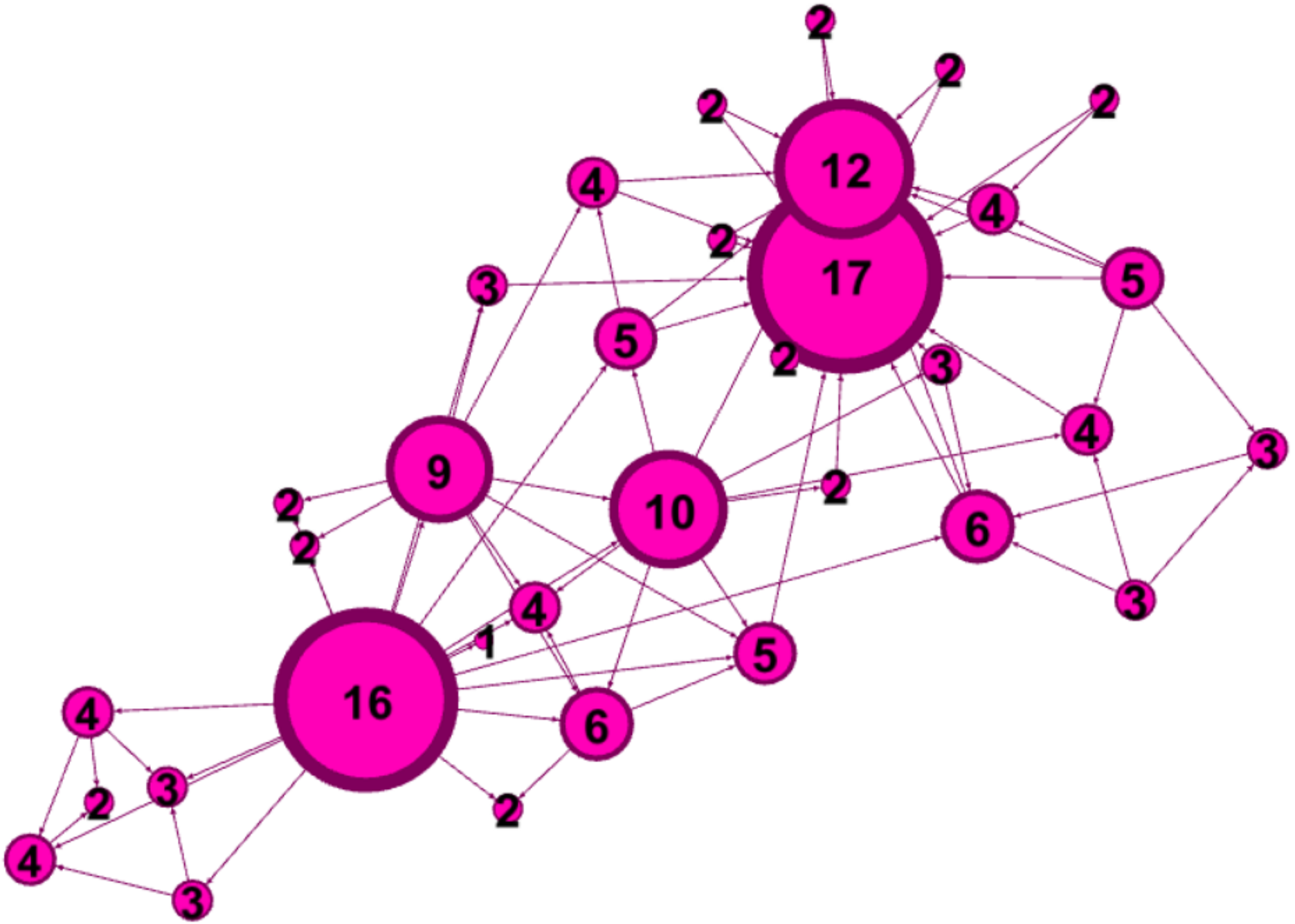
http://aris.ss.uci.edu/~lin/76.pdf

Measure number of connections:

```
In [1]: import networkx as nx

In [2]: G = nx.karate_club_graph()

In [3]: degree = nx.degree_centrality(G)
```


2
2
2
2
12
4
4
2
17
5
3
5
2
3
4
3
9
10
2
2
2
6
3
4
1
16
5
4
6
3
2
2
4
3

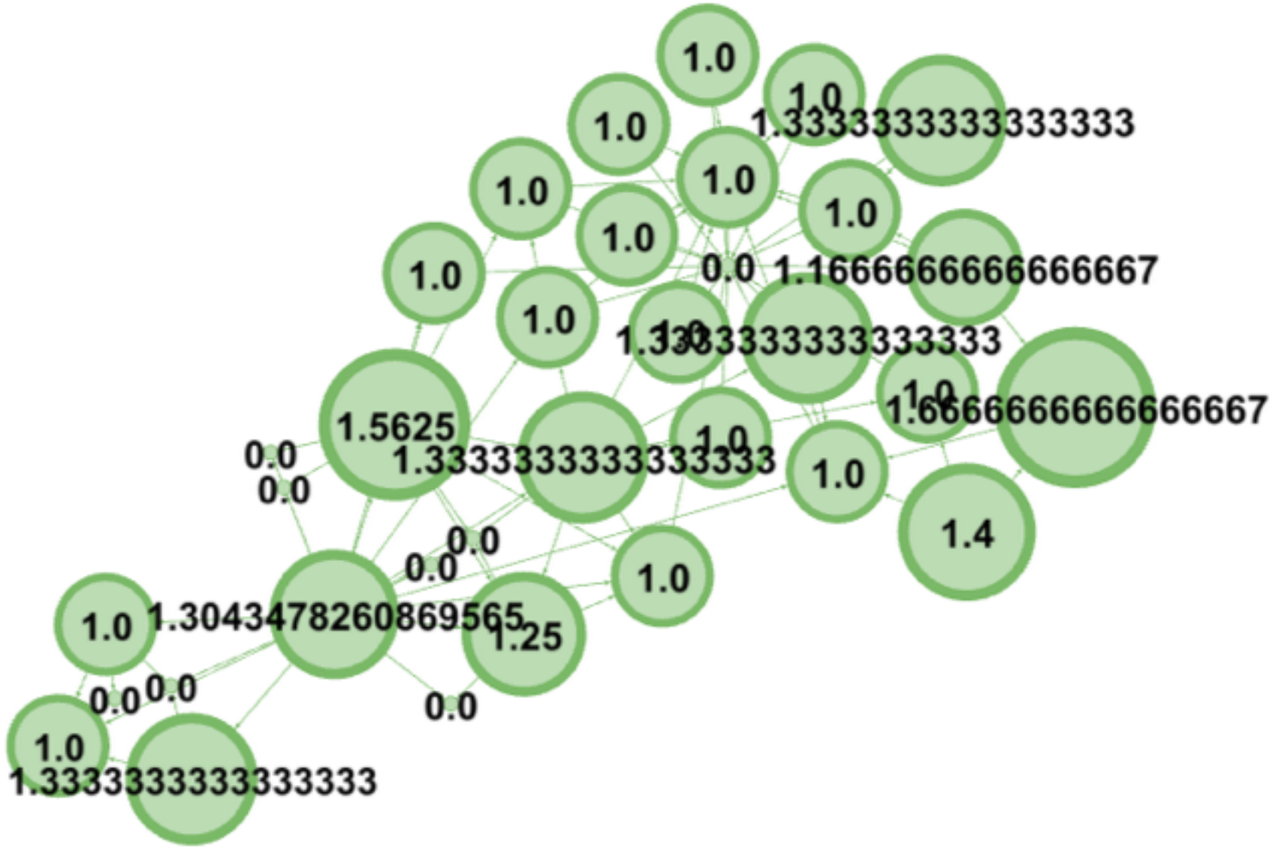
Measure the ‘closeness’ to other nodes in the graph:

```
…

In [4]: closeness = nx.closeness_centrality(G)
```

higher closeness means stronger ability to transmit

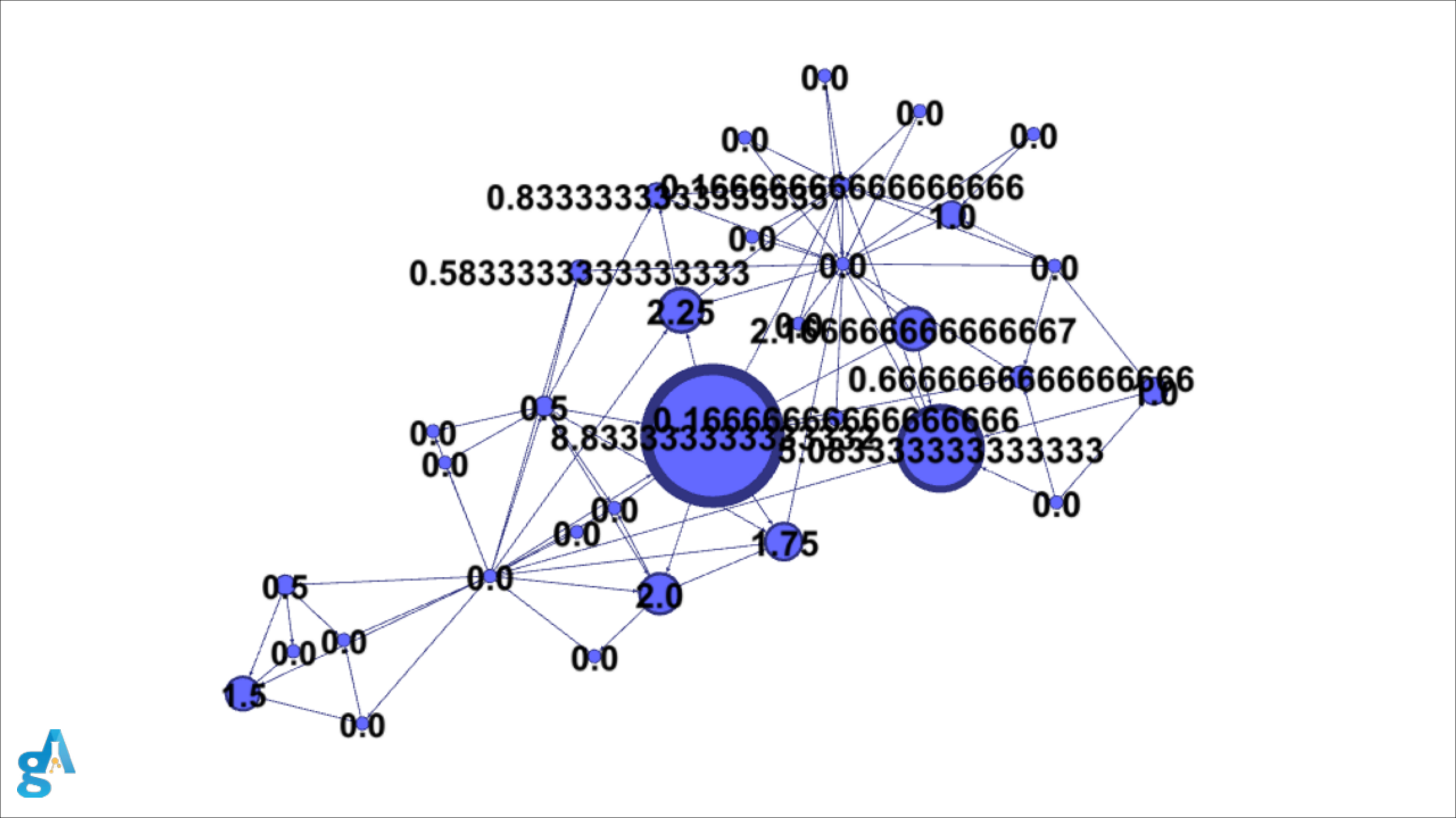
e.g. message, disease


1.0
1.0
1.0
1.3333333333333333
1.0
1.0
1.0
1.0
1.0
0.0
1.1666666666666667
1.0
1.0
1.3333333333333333
1.5625
1.0
1.0
1.1666666666666667
0.0
0.0
1.3333333333333333
1.0
1.4
0.0
0.0
1.0
1.0
1.3043478260869565
1.25
0.0
0.0
0.0
1.0
1.3333333333333333

Measure ‘influence’:

```
…
In [5]: betweeness = nx.betweenness_centrality(G)
```

the number of shortest paths that pass through a node


0.0
0.0
0.0
0.0
0.8333333333333333
0.0
1.0
0.5833333333333333
0.0
0.0
2.25
0.0
2.1666666666666667
0.6666666666666666
1.0
0.5
0.0
0.0
8.833333333333333
8.083333333333333
0.0
0.0
0.0
1.75
0.5
0.0
2.0
0.0
0.0
0.0
1.5
0.0

3: Predictive Features of Networks

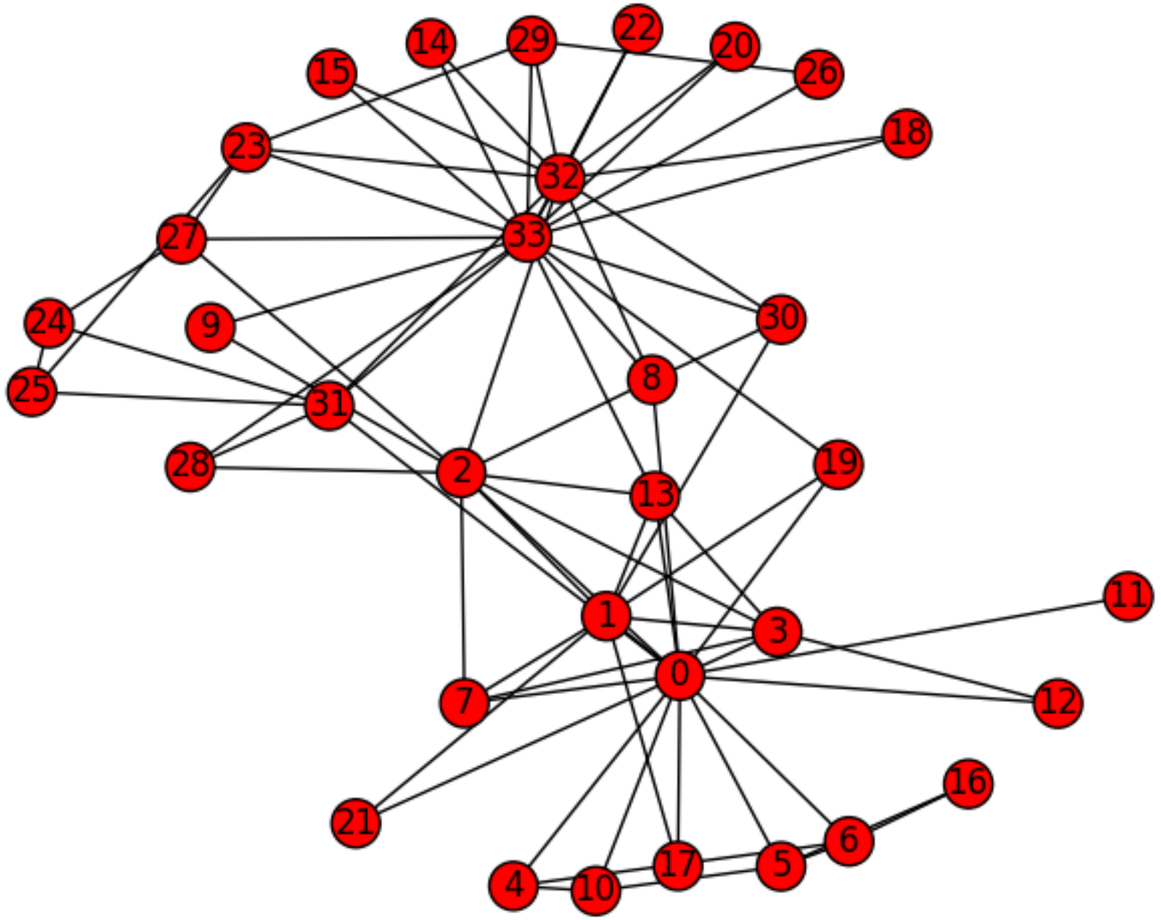
(Jumping ahead to later in the semester…)

Find communities:

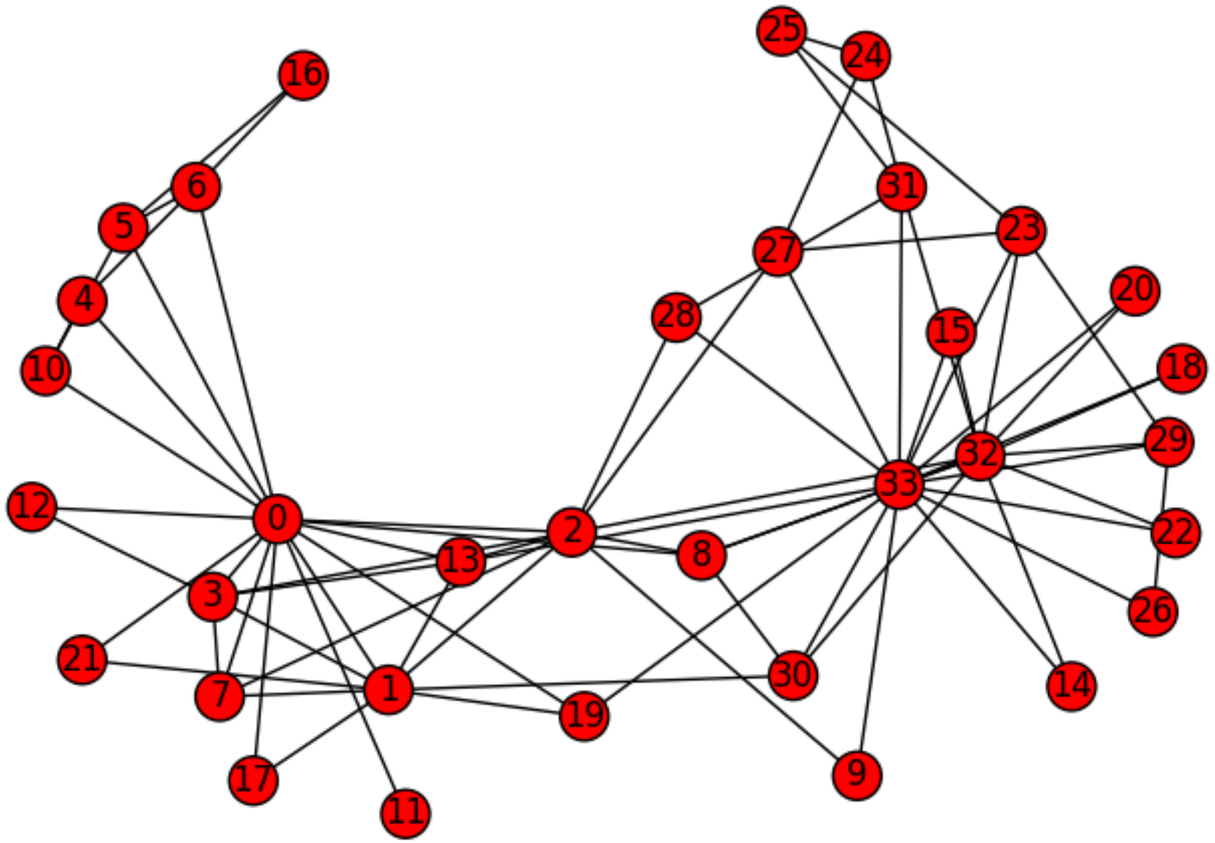
```
def updateGraph(G):
    ebc = NC.edge_betweenness(G)
    maxs = 0
    medge = None
    for k, v in ebc.iteritems():
        if maxs < v:
            medge, maxs = k, v
    G.remove_edge(*medge)
```

Girvan Newman algorithm https://gist.github.com/kenchan0130/3678097

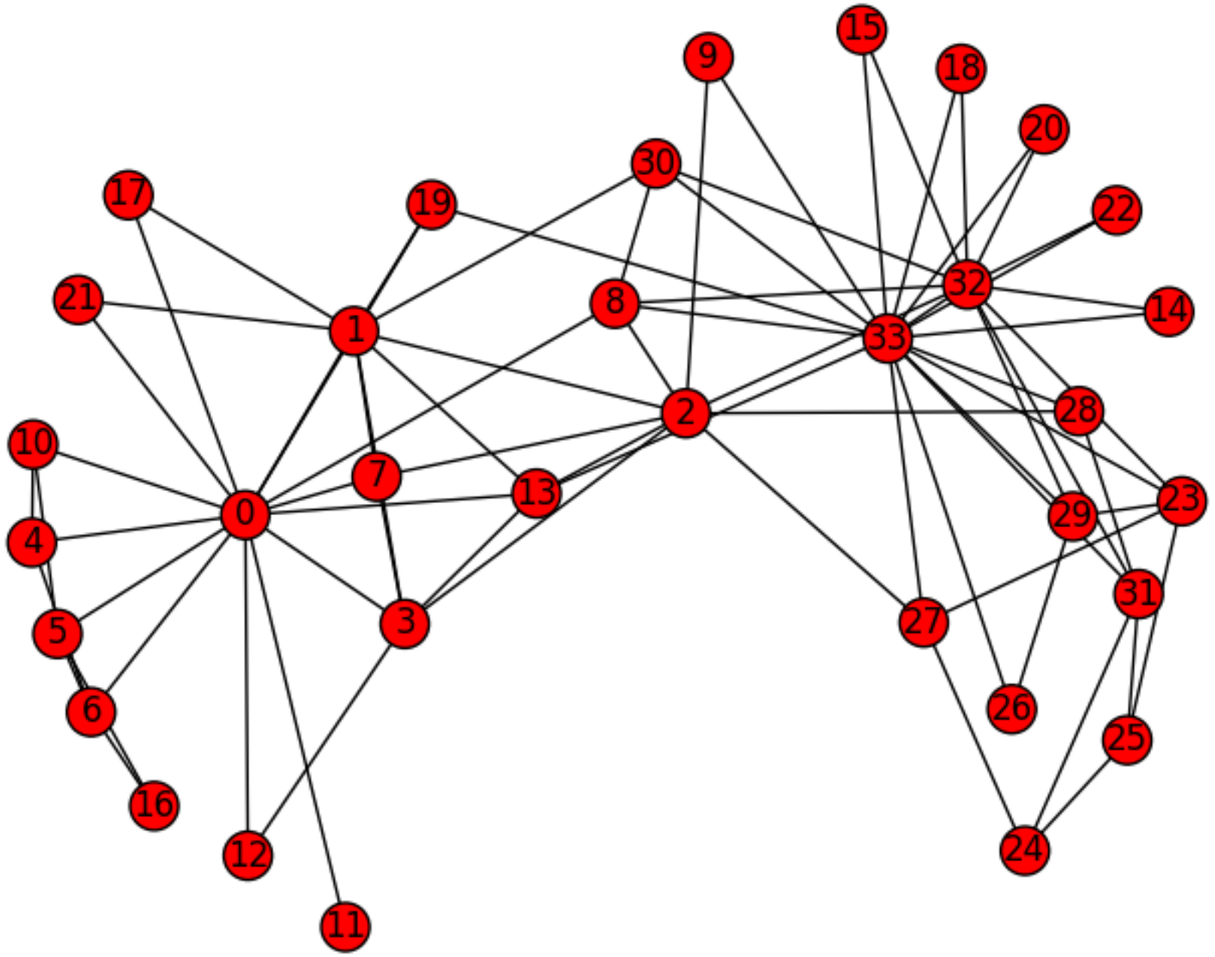
Note: GV can be considered outdated… weigh in on the SO thread: http://stackoverflow.com/questions/5822265/are-there-implementations-of-algorithms-for-community-detection-in-graphs


22
29
14
20
15
26
18
23
32
33
27
30
24
9
8
25
31
19
28
2
13
11
1
3
0
7
12
16
21
6
17
5
4
10

i=0 edge (0, 31) removed

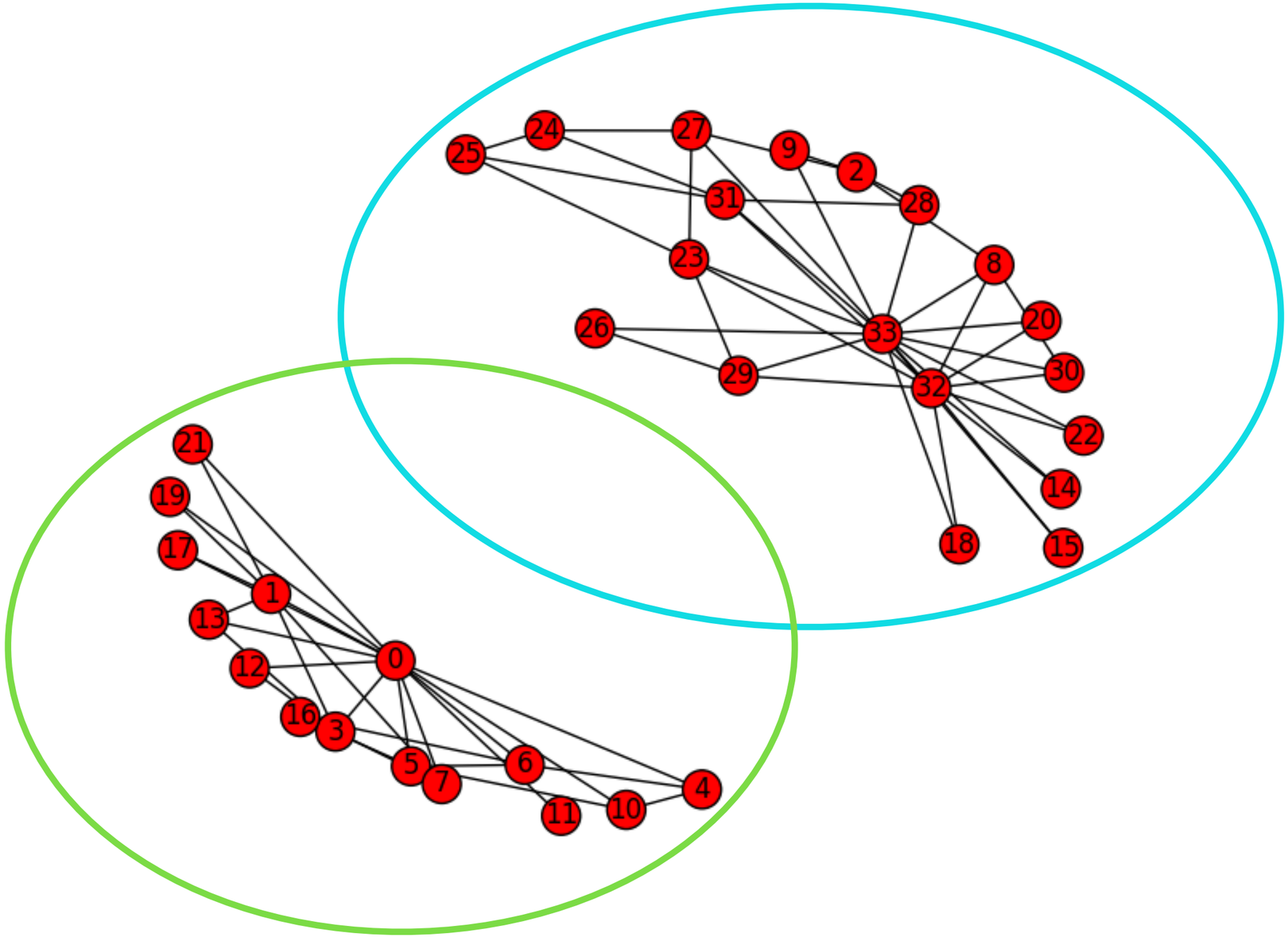


(0, 2)


15
9
18
20
30
17
19
22
32
21
8
14
1
33
2
28
10
7
13
23
0
29
4
31
3
27
5
6
26
25
16
12
24
11

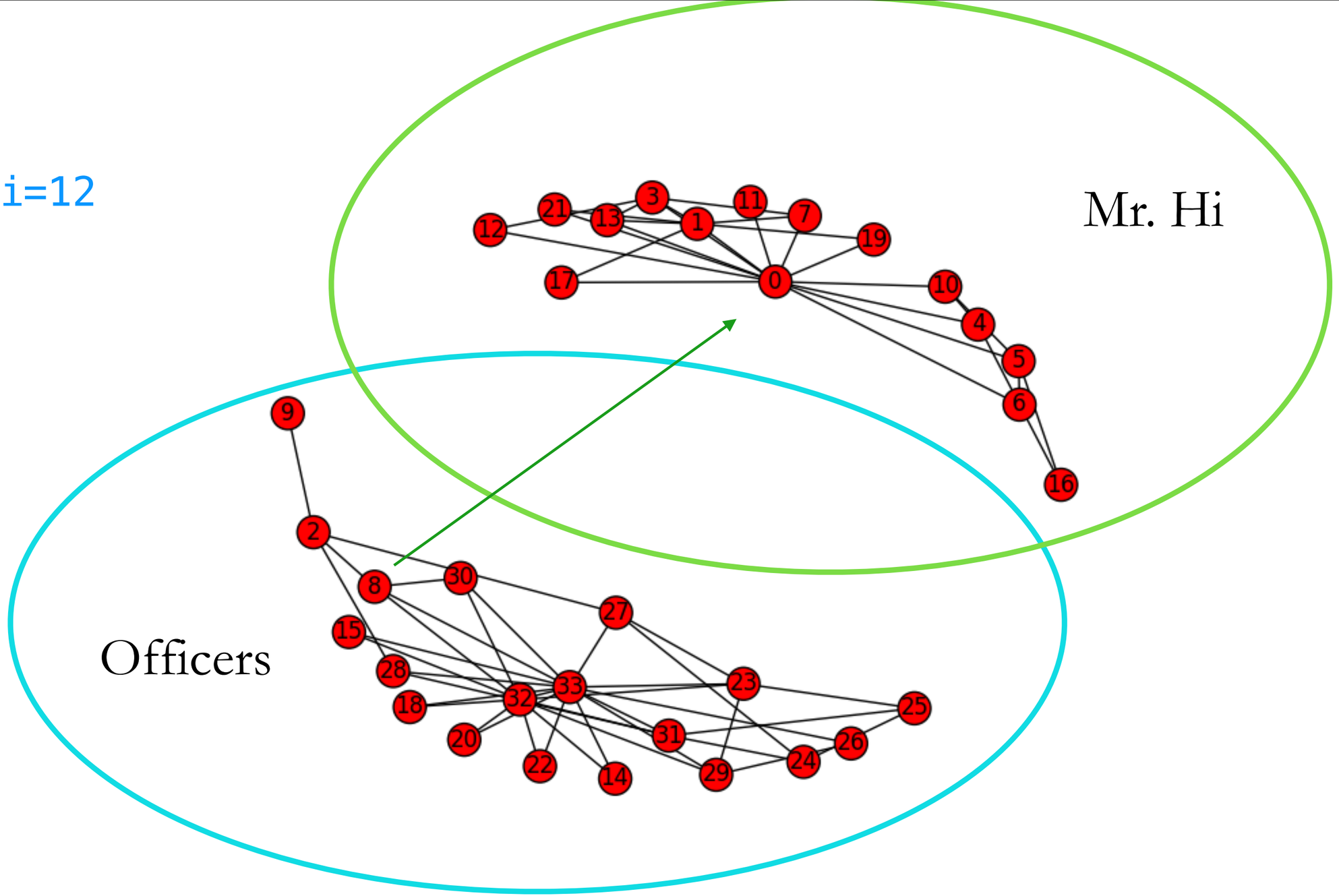
i=11

Officers



Mr. Hi

i=12

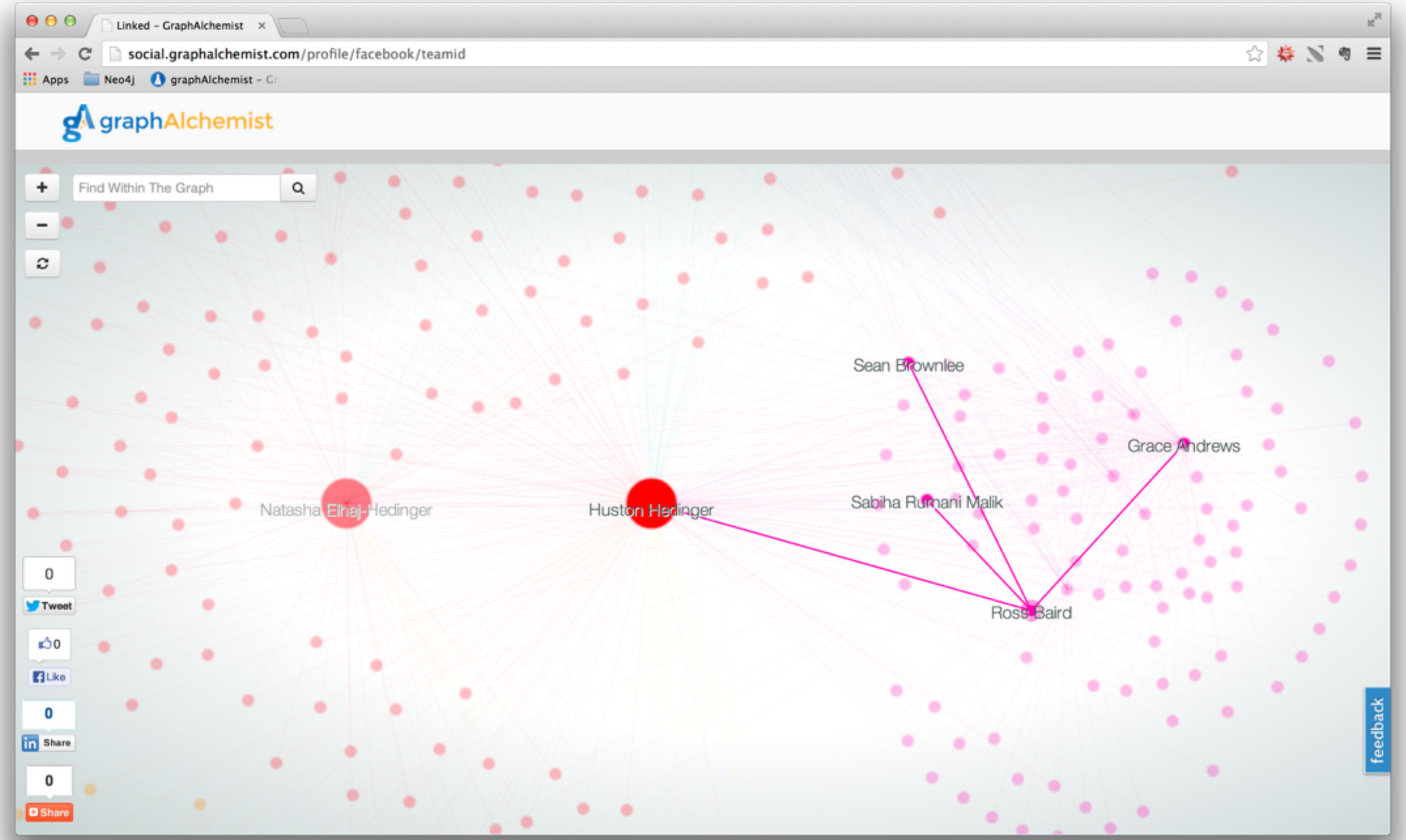

Mr. Hi
Officers
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33

In reality, node 8 goes with Mr. Hi…

a simple explanation…

3: Predictive Features of Networks (continued)

(Finals week…)


Linked - GraphAlchemist
social.graphalchemist.com/profile/facebook/teamid
Apps
Neo4j
graphAlchemist
graphAlchemist
Find Within The Graph
Sean Brownlee
Grace Andrews
Natasha Elhaj-Hedinger
Huston Hedinger
Sabiha Rumani Malik
Ross Baird
0
Tweet
0
Like
0
Share
0
Share
feedback

Predict ‘important’ relationships,

based on Facebook connections

build a NetworkX network from Huston's Facebook network

```
import networkx as nx
import json

file = open('data/network_json.json', 'r').read()
data = json.loads(file)
for edge in GraphJSON['edges']:
    start_node = edge['source']
    end_node = edge['target']
    G_u.add_edge(start_node, end_node)
```

dispersion:

```
In [45]: dispersion = nx.dispersion(G_u, 1)

In [46]: sorted_disp = sorted(dispersion.iteritems(),
key=operator.itemgetter(1), reverse=True)

In [47]: sorted_disp[:10]
Out[47]:
[(414, 39.71212121212121),
 (51, 13.525641025641026),
 (41, 7.25),
 (14, 6.835616438356165),
…
```

a measure of ‘tie strength’ where hi dispersion means that our connections in common are less likely to be connected to each other

```
In [47]: sorted_disp[:10]
Out[47]:
[(414, 39.71212121212121),
 (51, 13.525641025641026),
 (41, 7.25),
 (14, 6.835616438356165),
…
```

Cofounder

Wife

```
def _dispersion(G_u, u, v):
    """dispersion for all nodes 'v' in a ego network G_u of node 'u'"""
    u_nbrs = set(G_u[u])
    ST = set(n for n in G_u[v] if n in u_nbrs)
    set_uv=set([u,v])
    #all possible ties of connections that u and b share
    possib = combinations(ST, 2)
    total = 0
    …
```

```
…
for (s,t) in possib:
        #neighbors of s that are in G_u, not including u and v
        nbrs_s = u_nbrs.intersection(G_u[s]) - set_uv
        #s and t are not directly connected
        if not t in nbrs_s:
            #s and t do not share a connection
            if nbrs_s.isdisjoint(G_u[t]):
                #tick for disp(u, v)
                total += 1
    #neighbors that u and v share
    embededness = len(ST)
```

```
…
    if normalized:
        if embededness + c != 0:
            norm_disp = ((total + b)**alpha)/(embededness + c)
        else:
            norm_disp = (total+b)**alpha
        dispersion = norm_disp

    else:
        dispersion = total

    return dispersion
```

Tips:

1. Keep you networks balanced…

   (let's talk about triads next time?)

2. Avoid homophily…

Questions?

Stay in touch:

h@graphalchemist.com

@hustonhedinger