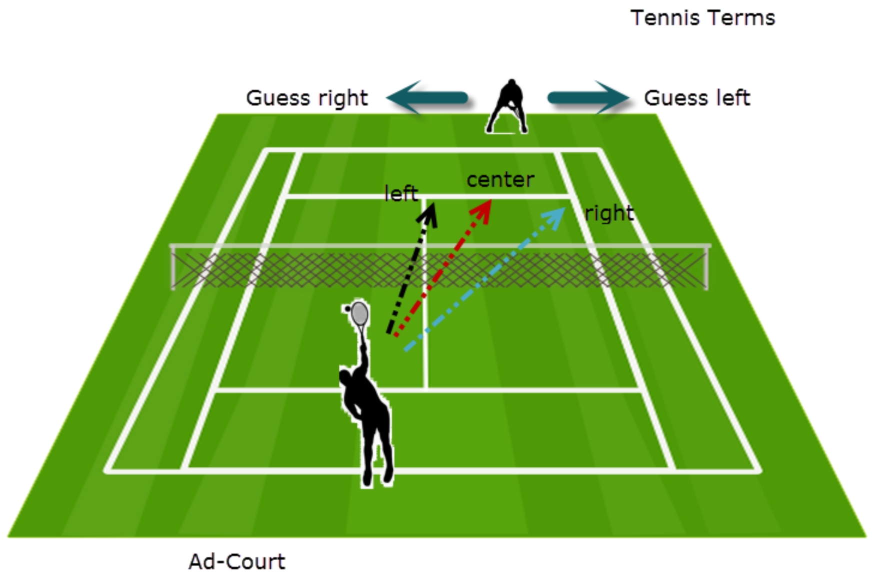
IS609 Group Project

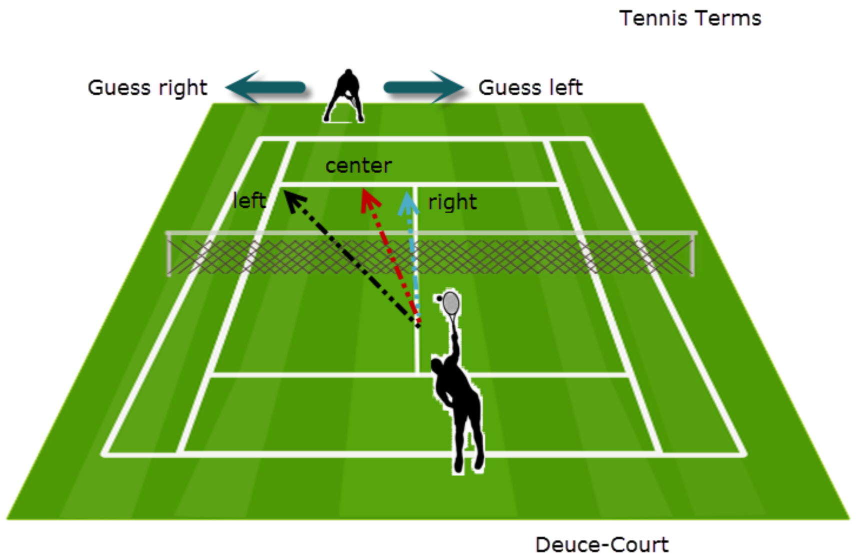
Brian Chu, Aaron Palumbo, James Quacinella

Game theory

- Tennis ‘duel’ between server and returner
- Total conflict:
    - Only one player wins the point at the expense of the other
- Mixed strategy:
    - Server can serve to the left, right, or center
    - Returner can guess to each direction too
    - Assume no pure/dominant strategy

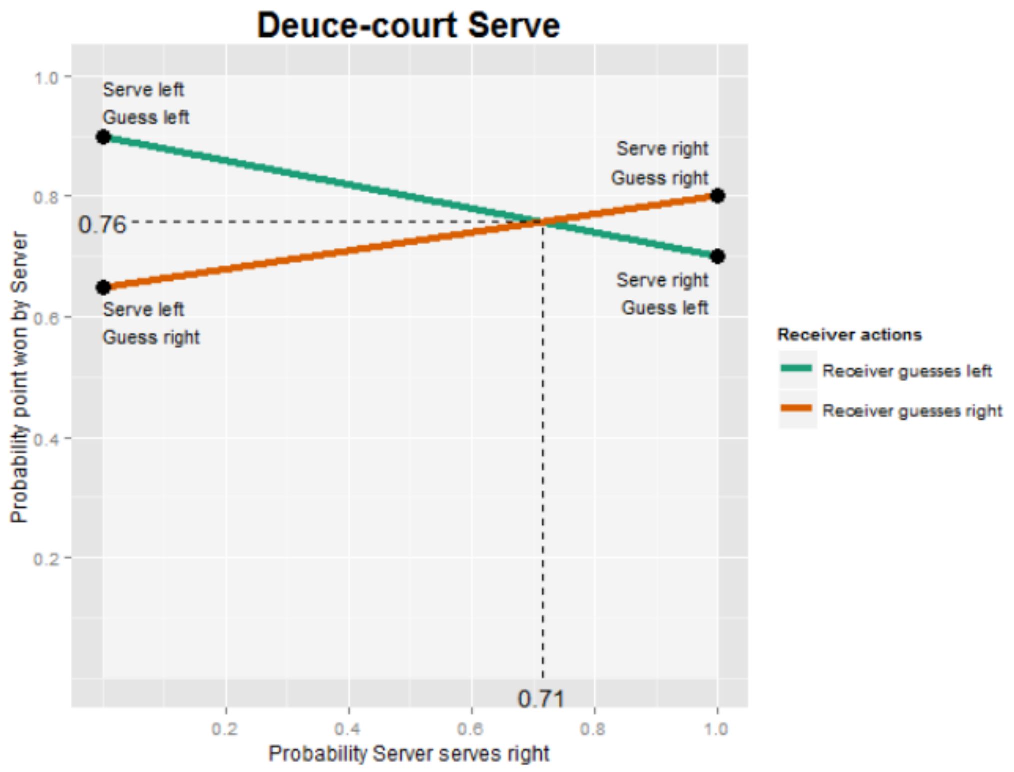
Definition of Terms





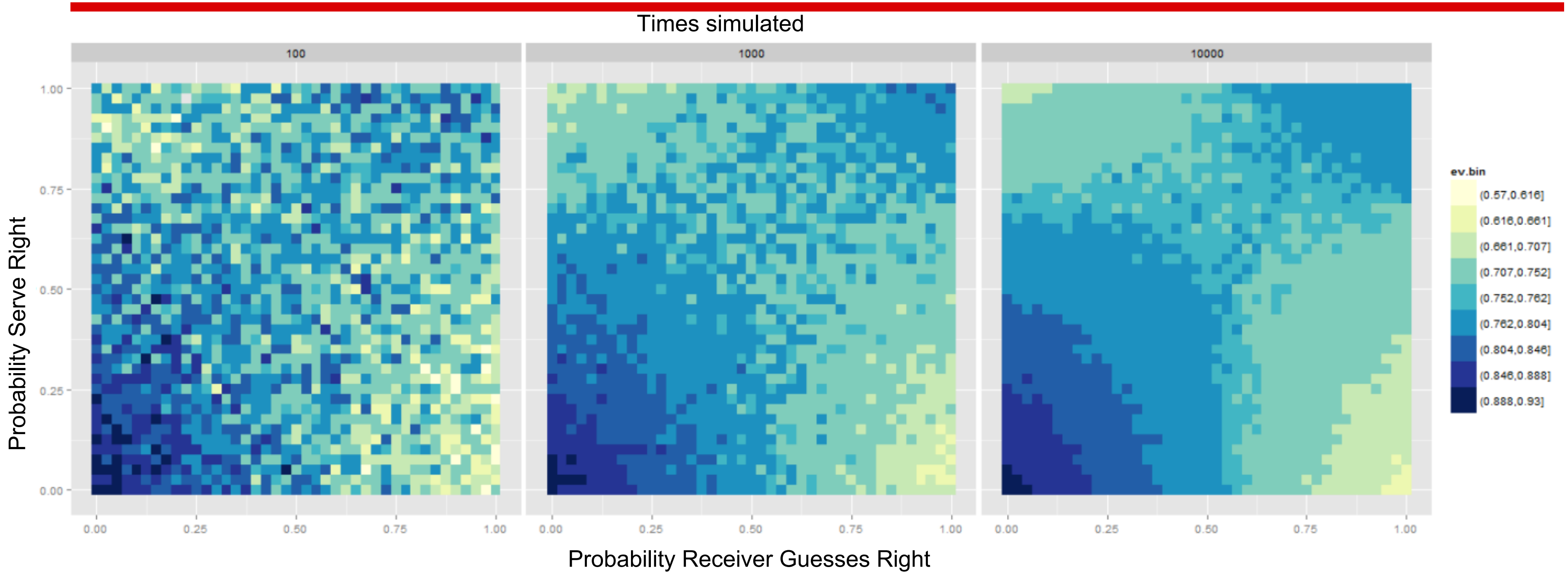
NOTICE: Serve right, Guess left means the receiver guessed correctly! This is confusing!

How to detect equilibrium play?



From game theory, we expect to see this, but we cannot detect whether the receiver is guessing left or right, so how do we determine if they are playing in equilibrium?

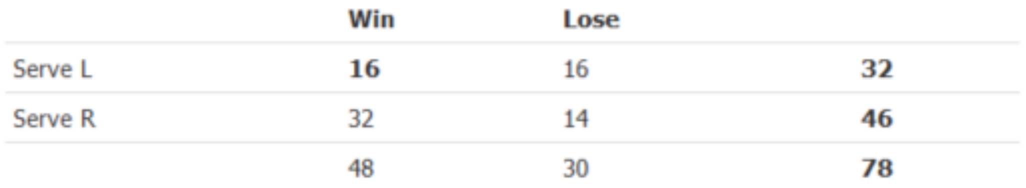
Simulated Game space



Each square is run through the simulation the indicated number of times. We can see the equilibrium play at server=0.71 and receiver=0.58. We also see the need to have many games.

Chi squared

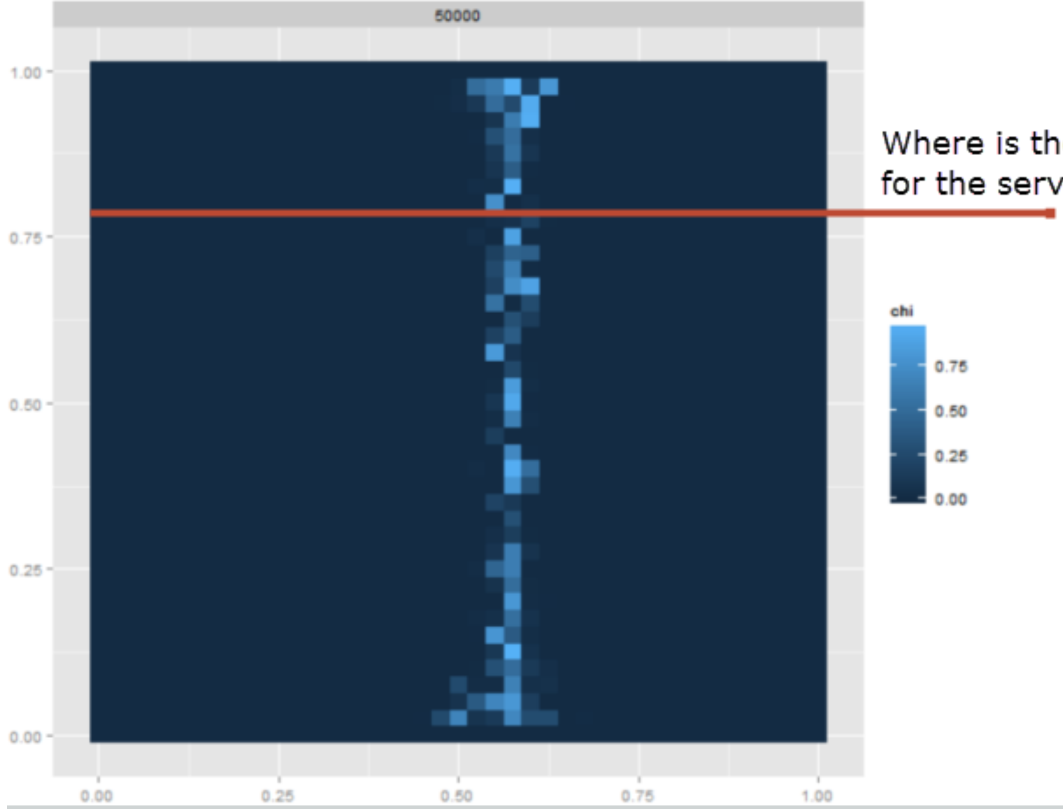
Estimate of Independence



	Win	Lose	
Serve L	**16**	16	**32**
Serve R	32	14	**46**
	48	30	**78**

Hypothesis: When play is in equilibrium, the chi squared statistic will be low, indicating that serving left or right is independent of winning or losing the point.

Chi Squared Over Game Space



Where is the line for the server?

Dataset

- Specific match data > career/aggregated data
- Choose scenario where optimization may be more likely:
  - High stakes match
  - Higher ranked players
  - Players very familiar with each other
  - Longer match (i.e. more service games played)
- Pete Sampras vs. Andre Agassi
  - Longtime rivals
  - 2001 US Open quarterfinal
  - Score: 6-7, 7-6, 7-6, 7-6
  - No service breaks the whole match

Agassi vs Sampras 2001 QF US Open: https://www.youtube.com/watch?v=ek3CqpKQo74#t=5m39s (**Вадим Чижов**)

Data variables

- Distinguish between 1st/2nd serve
    - 470 total serves (352 1st serves,118 2nd serves)
- Distinguish between deuce/ad court service points for each player
    - Sampras: 240 points (Deuce:130, Ad: 110)
    - Agassi: 230 points (Deuce: 123, Ad: 107)
- Record serves to the left (forehand) and right (backhand)
    - Center serves ignored (only ~1.5% of total)
- Record points won/lost after serving to each side
- Calculate percentage of points won on each side
    - Compare using chi-square statistic and p-value (*Walker and Wooders, 2000*)

Sampras

Deuce Court

Serve	%Win Left	%Win Right	Chi-sq	p-val
1	82.1	74.1	0.158	0.691
2	76.9	73.3	0	1
Both	80.5	73.8	0.214	0.644

Ad Court

Serve	%Win Left	%Win Right	Chi-sq	p-val
1	75.0	85.2	0.253	0.615
2	58.8	71.4	0.013	0.908
Both	67.6	82.4	1.34	0.247

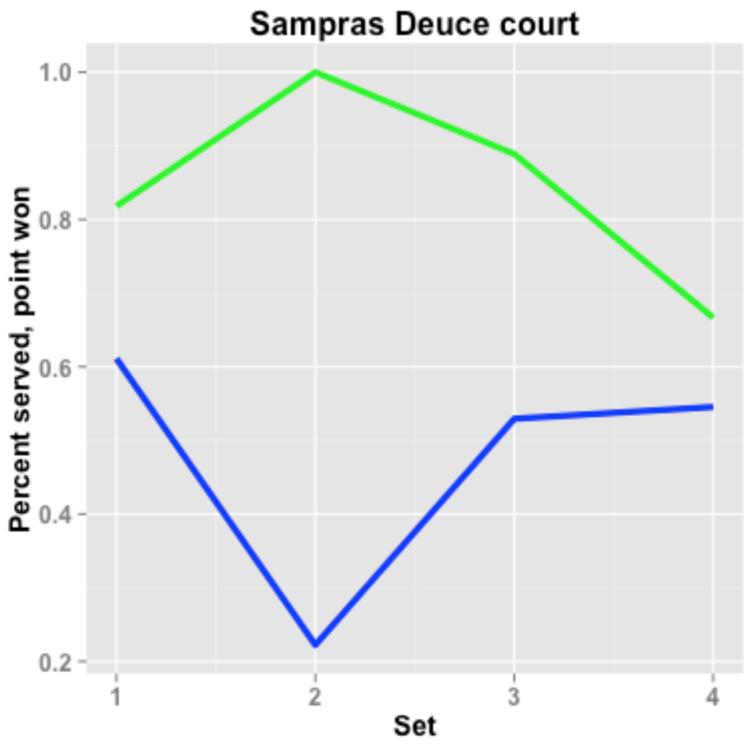
Agassi

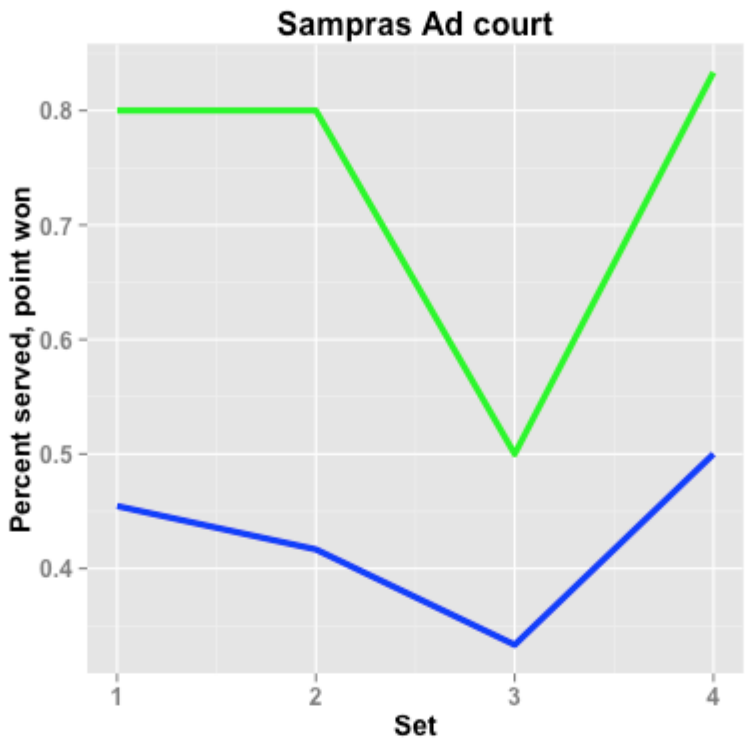
Deuce Court

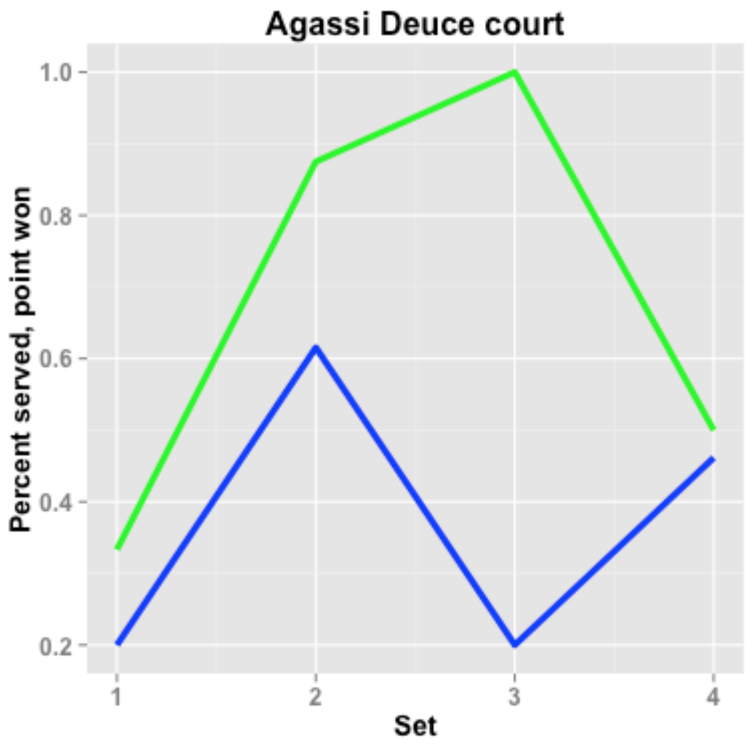
Serve	%Win Left	%Win Right	Chi-sq	p-val
1	70.0	83.3	0.681	0.409
2	80.0	54.5	0.293	0.588
Both	72.0	72.4	0	1

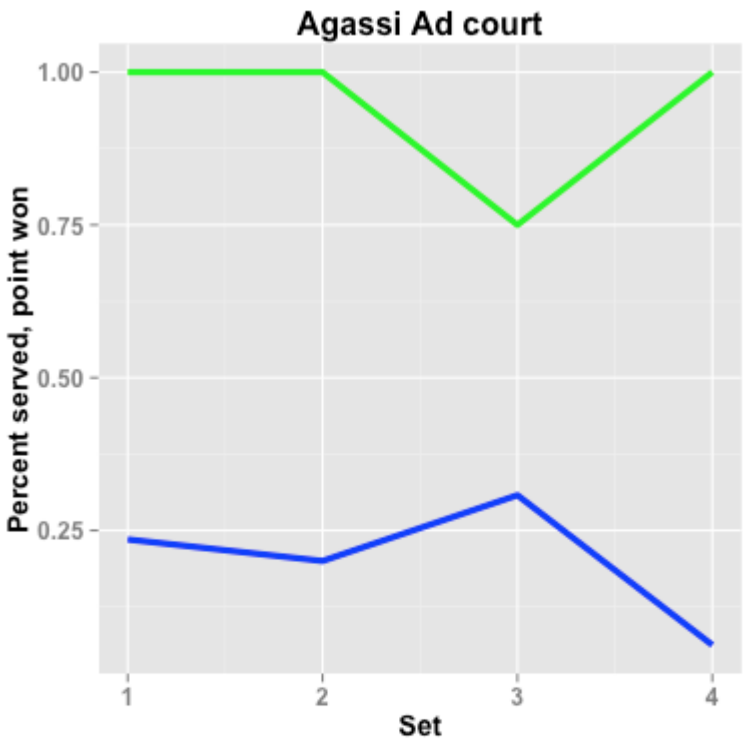
Ad Court

Serve	%Win Left	%Win Right	Chi-sq	p-val
1	90.9	68.9	1.207	0.272
2	40.0	50.0	0.000	1.000
Both	75.0	64.4	0.248	0.618


Sampras Deuce court
Percent served, point won
1.0
0.8
0.6
0.4
0.2
1
2
3
4
Set


Sampras Ad court
Percent served, point won
0.8
0.7
0.6
0.5
0.4
1
2
3
4
Set


Agassi Deuce court
Percent served, point won
1.0
0.8
0.6
0.4
0.2
1
2
3
4
Set


Agassi Ad court
Percent served, point won
1.00
0.75
0.50
0.25
1
2
3
4
Set

Conclusions

Some evidence of:

- Mixed strategy optimization
- Optimal mix varies as match goes on

Model limitations:

- Limited data set compared to simulation results
- Cannot record where returner is guessing
- Points won not perfectly correlated with serve direction accuracy

Graph Theory

For this portion of the project, we will see how can we use graph theory to model social networks?

- For this project, we looked into how we can model aspects of Twitter
- See if we can gain insights into the data science community on Twitter

Summary

1. We will show nodes can represent users and edges represent follow relationships. This basic model can describe the structure of relationships, but we can expand it even further.
    a. This would require us to create a graph where we have different kinds of nodes and different kinds of edges.

2. We will also show how to download data from twitter via the API and to come up with a graph visualization.

Twitter?

Twitter is an online social networking service that enables users to send and read short 140-character messages called "tweets". Users connect to one another by 'following' one another, which allows you to see a stream of tweets from them. We will use graph theory to help model the relationships between people on Twitter.

Modeling Follow Relationships

The graph model will have nodes representing users, and edges representing a follow relationship. Code snippet:

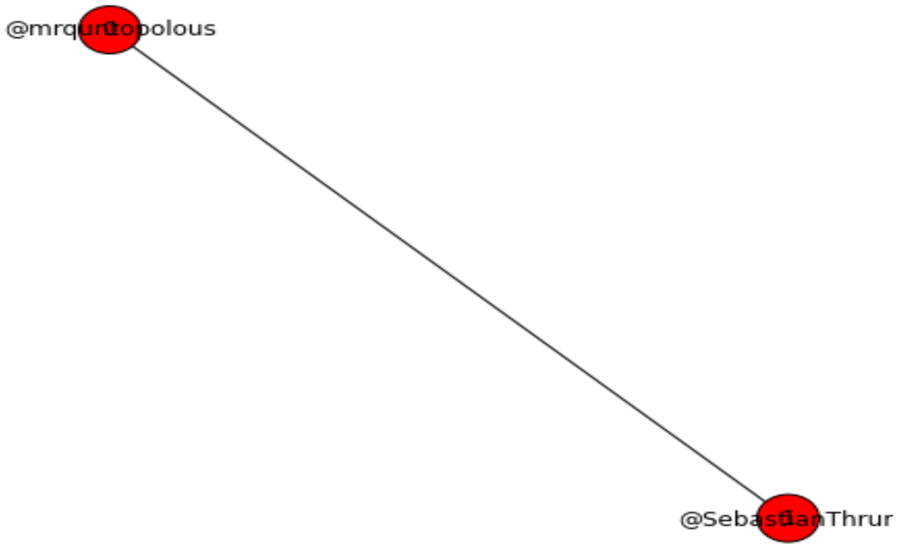
```
import networkx as nx
g = nx.Graph()
g.add_node(0, {"handle": "@mrquntopolous"})
g.add_node(1, {"handle": "@SebastianThrun"})
g.add_edge(0, 1)

node_labels = nx.get_node_attributes(g,'handle')

plt.figure(figsize=(6,6))

pos = nx.spring_layout(g)
nx.draw(g, pos, arrows=True, node_size=900)
nx.draw_networkx_labels(g, pos, labels = node_labels)
```

Graph 1



Not Enough!

But, the edge seems ambiguous! Who is following who? Lets add directionality to the edge!

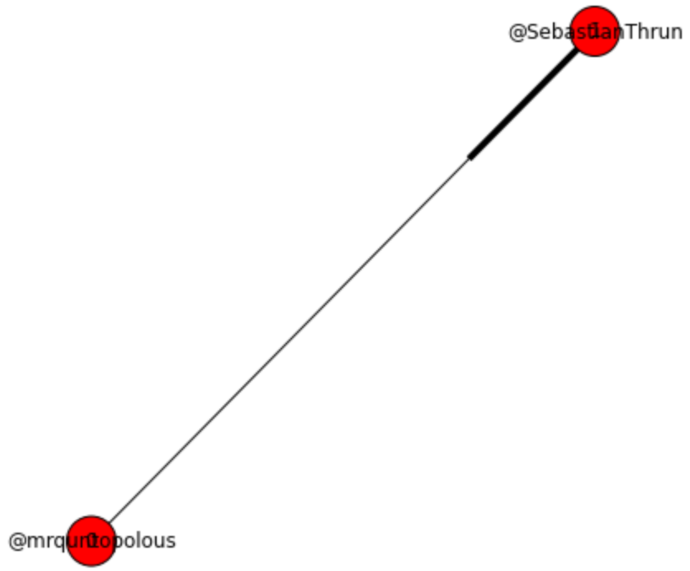
```
g = nx.DiGraph()
g.add_node(0, {"handle": "@mrquntopolous"})
g.add_node(1, {"handle": "@SebastianThrun"})
g.add_edge(0, 1)

node_labels = nx.get_node_attributes(g,'handle')

plt.figure(figsize=(6,6))

pos = nx.spring_layout(g)
nx.draw(g, pos, arrows=True, node_size=900)
nx.draw_networkx_labels(g, pos, labels = node_labels)
```

Graph 2



Extending the model

What if we need to model the tweets that user send out themselves, and how users interact with them? In the graph models we have used prior, typically a node represented one kind of thing. However, the previous nodes represented users, so how will we model tweets or other entities?

Using a property graph, we can add properties to nodes and edges, which can help us differentiate between different kinds of entities and relationships in our graphs.

For example, we can add a node that represents a tweet made by one of these users. We will color it differently to emphasize that its a different kind of node.

Sample code

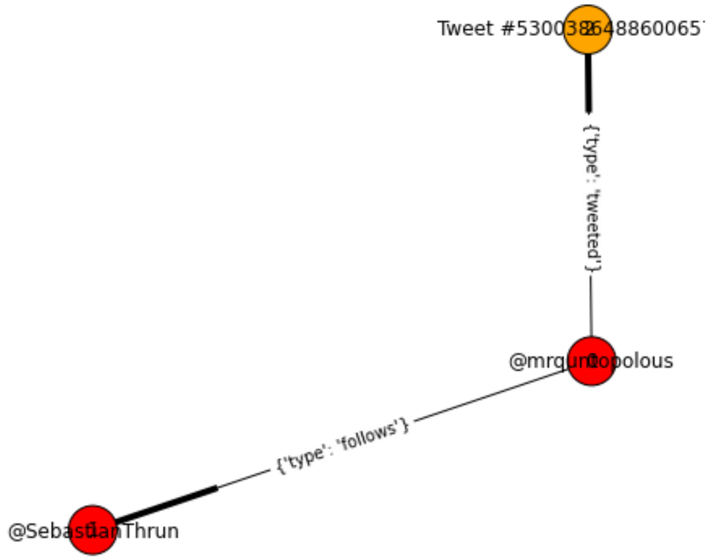
```
g = nx.DiGraph()
g.add_node(0, {"id": "@mrquntopolous", "color": "r"})
g.add_node(1, {"id": "@SebastianThrun", "color": "r"})
g.add_node(2, {"id": "Tweet #530038648860065793", "color": "orange"})
g.add_edge(0, 1, {"type": "follows"})
g.add_edge(0, 2, {"type": "tweeted"})

node_labels = nx.get_node_attributes(g,'id')
edge_labels = nx.get_edge_attributes(g,'type')
colors = nx.get_node_attributes(g, 'color')

plt.figure(figsize=(6,6))

pos = nx.spring_layout(g)
nx.draw(g, pos, arrows=True, node_size=900, node_color=colors.values())
nx.draw_networkx_labels(g, pos, labels = node_labels)
nx.draw_networkx_edge_labels(g, pos, labels = edge_labels)
```

Graph 3


Tweet #530038648860065
2
{'type': 'tweeted'}
@mrqunopolous
0
{'type': 'follows'}
@SebastianThrun
1

Twitter API

Twitter has an official API that allows users to query their service for data. The only thing you need is a developer account with proper OAuth keys.

In the following example, we use the GetUser() API call from Twitter, which returns a response in JSON.

Sample Twitter code

```
import twitter
from config import *
import pickle
import pprint
import json

# Create Twitter API object
api = twitter.Api(consumer_key=CONSUMER_KEY, consumer_secret=CONSUMER_SECRET,
                      access_token_key=ACCESS_TOKEN_KEY,access_token_secret=ACCESS_TOKEN_SECRET)

# Print
hmason_json = api.GetUser(screen_name='hmason')
pprint.pprint(json.loads(str(hmason_json)))
```

Sample JSON for User

```
{u'created_at': u'Sun Feb 11 21:22:24 +0000 2007',
 u'description': u'Founder at @FastForwardLabs. Data Scientist in Residence at @accel. I \u2665 data and cheeseburgers.',
 u'favourites_count': 3549,
 u'followers_count': 50613,
 u'friends_count': 1204,
 u'id': 765548,
 u'lang': u'en',
 u'listed_count': 3290,
 u'location': u'NYC',
 u'name': u'Hilary Mason',
 u'profile_background_color': u'000000',
 u'profile_background_tile': False,
 u'profile_image_url': u'https://pbs.twimg.com/profile_images/1290564266/me_square_normal.jpg',
 u'profile_link_color': u'282F8A',
 u'profile_sidebar_fill_color': u'http://pbs.twimg.com/profile_background_images/73787182/background_steampunk_smaller.jpg',
 u'profile_text_color': u'000000',
 u'protected': False,
 u'screen_name': u'hmason',
 …
 u'statuses_count': 13647,
 u'time_zone': u'Eastern Time (US & Canada)',
 u'url': u'http://t.co/ijjKDNIVOE',
 u'utc_offset': -18000}
```

Crawling Twitter

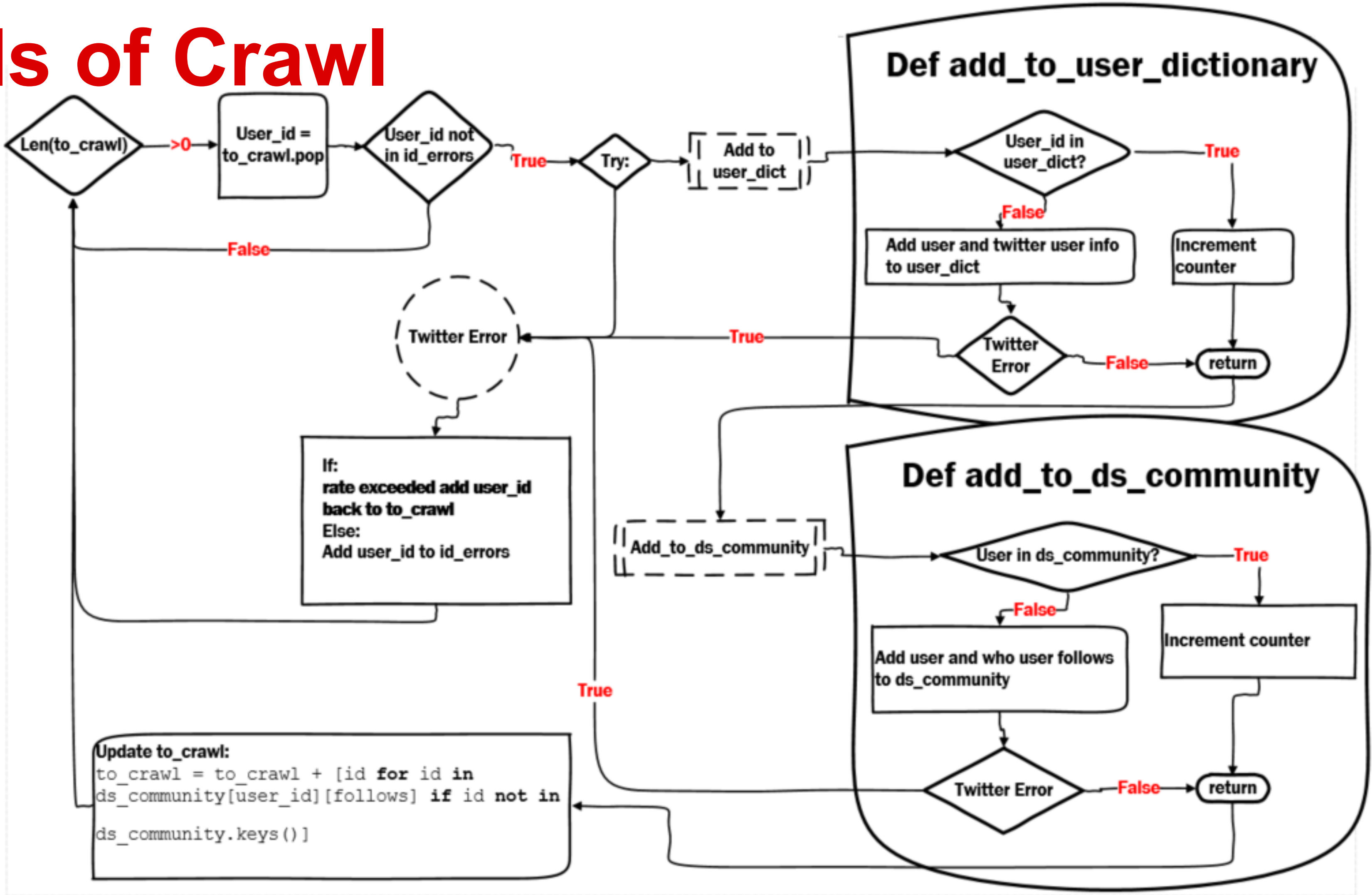
For the project, we decided to map out the Data Science community on twitter. The plan is to construct a graph where nodes represent users and edges represent a follow relationship.

We did this by starting with some known people in the Data Science world, like Hilary Mason, which we call *seed nodes*. We then used the Twitter API to find who follows these known people. These people are known as being *one degree of separation* away from the seed nodes. We also took these users and found out who they follow as well, getting us new nodes that are *two degrees of separation*.

Issues

- Twitter's API has a limitation on the number of API calls that can be made, so crawling is slow
- Need good error handling and backoff logic to handle errors from Twitter's API

Details of Crawl



```
to_crawl = to_crawl + [id for id in
ds_community[user_id][follows] if id not in

ds_community.keys()]
```

Building A Graph From Crawl Results

Given that objects created by the crawl, we can now use them to create a NetworkX graph. Below is a snippet of code that builds the graph from our crawl results (network_graph.py, function graph_construction):

Code Snippet: Building Graph

```
# Initialize directed graph
g = nx.DiGraph()

# Filter out nodes that have low counts
user_dict_filtered = {}
for uid in user_dict:
        if user_dict[uid]['count'] >= 2:
        user_dict_filtered[uid] = user_dict[uid]

uids_in_graph = user_dict_filtered.keys()

# Add nodes to graph
for uid in uids_in_graph:
        g.add_node(uid, {'handle': user_dict_filtered[uid]['name']})

# Add edges to graph
for uid in uids_in_graph:
        follows_in_graph = [i for i in graph_info[uid]['follows'] if i in uids_in_graph]
        for i in follows_in_graph:
                g.add_edge(uid, i, color='blue')
```

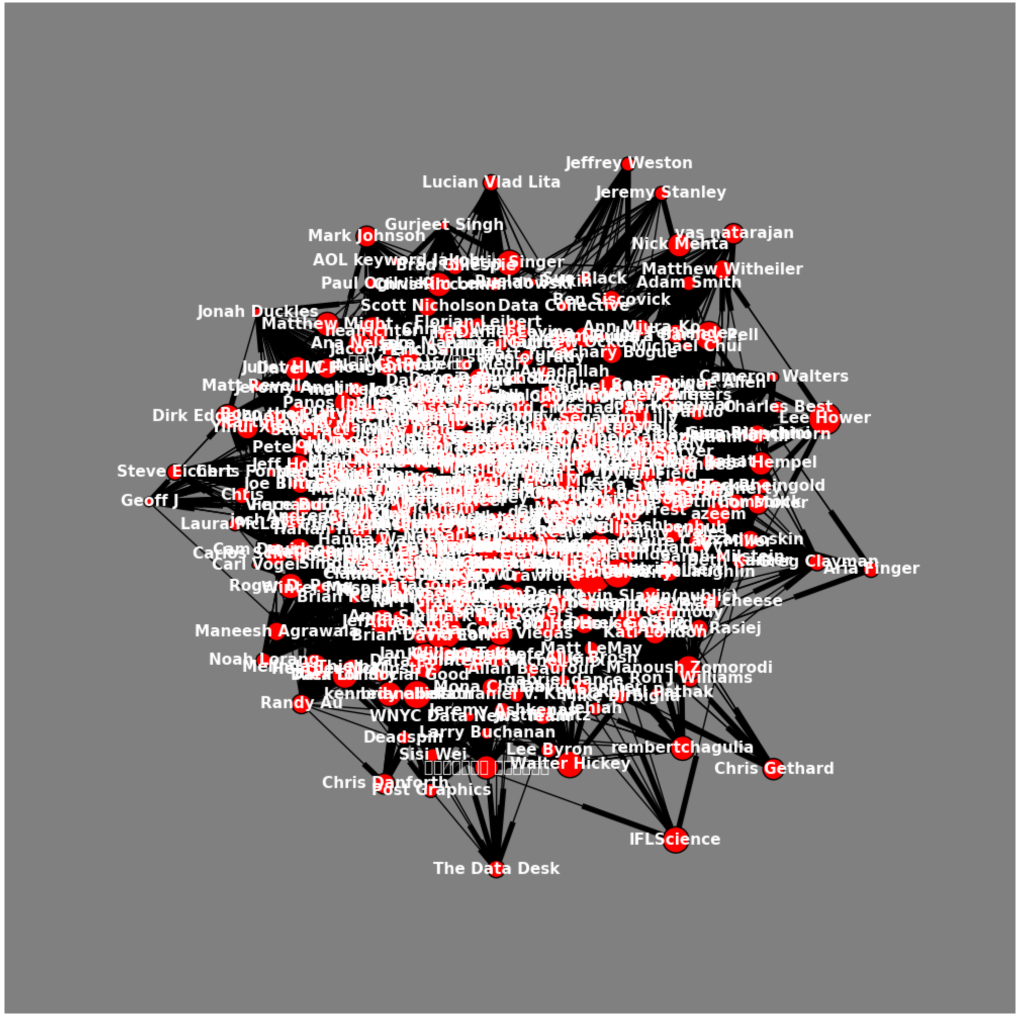
Displaying the Graph

There are issues with displaying large graphs like this:

- Too many nodes to see details correctly
- Too many edges overlapping

Solution is to :

- trim the graph as best you can
- layout the nodes using NetworkX's positioning features


Jeffrey Weston
Lucian Vlad Lita
Jeremy Stanley
vas natarajan
Gurjeet Singh
Mark Johnson
Nick Mehta
Matthew Witheiler
Adam Smith
Sue Black
Ben Siscovick
Scott Nicholson
Data Collective
Jonah Duckles
Florian Leibert
Michael Chui
Cameron Walters
Charles Best
Lee Hower
Hempel
Steve Eichert
Geoff J
Chris
Carl Vogel
Maneesh Agrawala
Noah Lorang
Randy Au
Fernanda Viegas
Matt LeMay
Allan Beaufour
Manoush Zomorodi
Ron J Williams
Kati London
Rasiej
Greg Clayman
Mona Chalabi
WNYC Data News
Larry Buchanan
Deadspin
Sisi Wei
Lee Byron
Walter Hickey
rembertchagulia
Chris Gethard
Chris Danforth
Post Graphics
IFLScience
The Data Desk

Yikes!

Not very informative! It would be great to organize the graph based on which nodes are ‘more central’ or ‘important’ to the structure of the graph. Is there any way to measure this property on nodes?

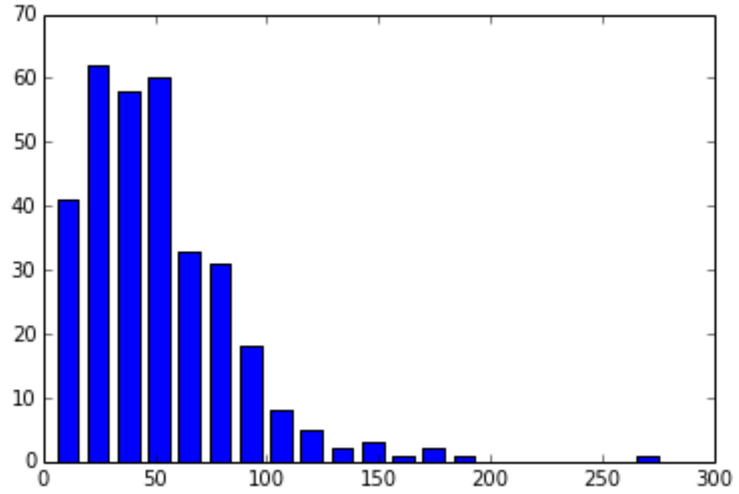
Graph Centrality

In graph theory, *centrality* refers to various metrics which identify which of the vertices within a graph are the most important. For our purposes, we would like to identify the most influential people in the data science community that we crawled. There are many different ways of defining centrality, including:

- **Degree Centrality** - One metric that can help define which vertices are important is using the number of links incident upon a node. The more edges on a node, the more 'important' it might be. This is one the simplest metrics for centrality.
- **Closeness Centrality** - In graph theory, the *shortest path* is the minimal amount of hops needed to travel between two nodes. We define another distance metric that is defined as the sum of its shortest path distances to all other nodes. The smaller the number, the closer this node is to all other nodes, so we can define *closeness* as the reciprocal of this metric. We will not use this metric.
- **Betweenness Centrality** - this metric quantifies the number of times a node acts as a 'bridge' along the shortest path between two other nodes. The more often a node is along shortest path in the graph, the more likely it is to be important. Notice how both closeness and betweenness are defined in terms of the whole graph, while degree centrality is defined locally per node.
- **PageRank** - is an iterative algorithm that outputs a probability distribution over nodes that represents the likelihood that a person randomly moving along edges will arrive at any particular node. After each iteration, the node's value is an approximate PageRank value that approaches the theoretical true value over time.

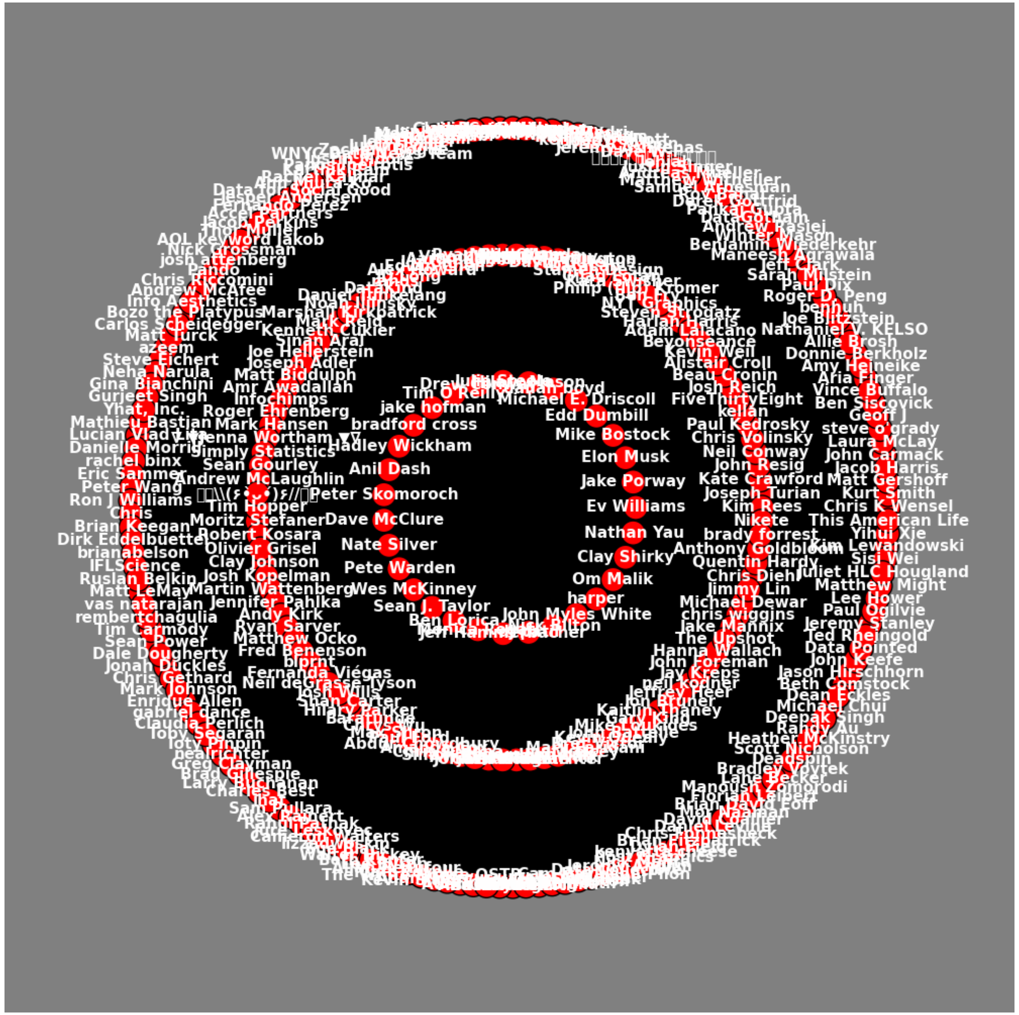
Degree Centrality

Lets look at our crawled graph and see if we can find nodes with a high number of edges coming in. Using networkx's in_degree() function, we can look at a histogram of in-degree values. Notice the long tail of nodes with large number of incoming edges.



Lets Try Visualizing Again

We can use this information to organize nodes into layers, based on their in-degree. The results are a lot more readable:


Tim O'Reilly
jake hofman
Michael E. Driscoll
Edd Dumbill
bradford cross
Mike Bostock
Hadley Wickham
Elon Musk
Anil Dash
Jake Porway
Peter Skomoroch
Ev Williams
Dave McClure
Nathan Yau
Nate Silver
Clay Shirky
Pete Warden
Om Malik
Wes McKinney
harper
Sean J. Taylor
John Myles White
Ben Lorica
Daniel Tunkelang
Marshall Kirkpatrick
Kenneth Cukier
Sinan Aral
Joe Hellerstein
Joseph Adler
Matt Biddulph
Amr Awadallah
Infochimps
Roger Ehrenberg
Mark Hansen
Simply Statistics
Sean Gourley
Andrew McLaughlin
Tim Hopper
Moritz Stefaner
Robert Kosara
Olivier Grisel
Clay Johnson
Josh Kopelman
Martin Wattenberg
Jennifer Pahlka
Andy Kirk
Ryan Sarver
Matthew Ocko
Fred Benenson
Fernanda Viégas
Neil deGrasse Tyson
Josh Wills
Shan Carter
Hilary Parker
Philip (flip) Kromer
NYT Graphics
Steven Strogatz
Adam Laiacano
Beyonseance
Kevin Weil
Alistair Croll
Beau Cronin
Josh Reich
FiveThirtyEight
kellan
Paul Kedrosky
Chris Volinsky
Neil Conway
John Resig
Kate Crawford
Joseph Turian
Kim Rees
Nikete
brady forrest
Anthony Goldbloom
Quentin Hardy
Chris Diehl
Jimmy Lin
Michael Dewar
chris wiggins
Jake Mannix
The Upshot
Hanna Wallach
John Foreman
Jay Kreps
neil kodner
jeffrey heer
Jon Bruner
Kaitlin Thaney
WNYC Data News Team
Data for Social Good
Accel Partners
Jacob Perkins
AOL keyword Jakob
Nick Grossman
josh attenberg
Pando
Chris Riccomini
Andrew McAfee
Info Aesthetics
Bozo the Platypus
Carlos Scheidegger
Matt Turck
azeem
Steve Eichert
Neha Narula
Gina Bianchini
Gurjeet Singh
Yhat, Inc.
Mathieu Bastian
Lucian Vlad Lita
Danielle Morrill
rachel binx
Eric Sammer
Peter Wang
Ron J Williams
Chris
Brian Keegan
Dirk Eddelbuettel
brianabelson
IFLScience
Ruslan Belkin
Matt LeMay
vas natarajan
rembertchagulia
Tim Carmody
Sean Power
Dale Dougherty
Jonah Duckles
Chris Gerhard
Mark Johnson
Enrique Allen
gabriel dance
Claudia Perlich
Toby Segaran
neal richter
Greg Clayman
Brad Gillespie
Larry Buchanan
Charles Best
Sam Pullara
Cameron Waters
Samuel Arbesman
Derek Gottfrid
Benjamin Wiederkehr
Maneesh Agrawala
Jeff Clark
Sarah Milstein
Paul Dix
Roger D. Peng
benhuh
Joe Blitzstein
Nathaniel V. KELSO
Allie Brosh
Donnie Berkholz
Amy Heineike
Aria Finger
Vince Buffalo
Ben Siscovick
Geoff J
steve o'grady
Laura McLay
John Carmack
Jacob Harris
Matt Gershoff
Kurt Smith
Chris K Wensel
This American Life
Yihui Xie
Kim Lewandowski
Sisi Wei
Juliet HLC Hougland
Matthew Might
Lee Hower
Paul Ogilvie
Jeremy Stanley
Ted Rheingold
Data Pointed
John Keefe
Jason Hirschhorn
Beth Comstock
Dean Eckles
Michael Chui
Deepak Singh
Randy Au
Heather McKinstry
Scott Nicholson
Deadspin
Bradley Voytek
Lane Becker
Manoush Zomorodi
Brian David Eoff

Any Useful Information?

Ev Williams, co-founder of twitter

Jake Porway, Founder and Executive Director @ DataKind

Elon Musk, Not really in DS community but in broader tech community

Edd Dumbill, VP Strategy at Silicon Valley Data Science

Michael E. Driscoll, Founder + CEO @Metamarkets; data; analytics; visualization

danah boyd, Microsoft Researcher; focuses on people's everyday practices involving social media

Julie Steele, Director of Communications at Silicon Valley Data Science

Tim O'Reilly, Founder of O'Reilly Media and a supporter of the free / open source movements

jake hofman, research scientist at msr nyc, interested in learning from data.

bradford cross, I’m a founder CEO focused on consumer products, investing, and data technologies.

Anil Dash, CoFounder CEO of ThinkUp, the first analytics tool for social media

Peter Skomoroch, data scientist @ Data Collective

Dave McClure, Entrepreneur and prominent angel investor

Pete Warden, Engineer at Google

Wes McKinney, Author of "Python for Data Analysis"

Sean J. Taylor, Social Scientist. Hacker. Facebook Data Science Team

Ben Lorica, Chief Data Scientist at O'Reilly Media, Inc

Monica Rogati, Data scientist with a passion for turning data into products

Jeff Hammerbacher, Founder and Chief Scientist of Cloudera

Nick Bilton, Journalist and author; formerly lead blogger for the New York Times' bits blo

John Myles White, Authors of Machine Learning for Hackers

harper, Former CTO @ Obama for America; founder of Modest

Om Malik, Web and technology writer; founder of and senior writer for GigaOM.

Clay Shirky, Writer on the social and economic effects of Internet technologies

Nathan Yau, Owner of FlowingData. Author of Visualize This

Summary of Central Nodes

The vast majority of them seem to be good candidates for being in the Data Science community.

Notice that some users are not really in the data science community, like Clay Shirky, Dave McClure and Elon Musk. This highlights another aspect of graph theory, and that is of *communities* and *community detection*. Communities are groups of nodes in a graph that are highly connected, indicated that they 'belong' together for some reason.

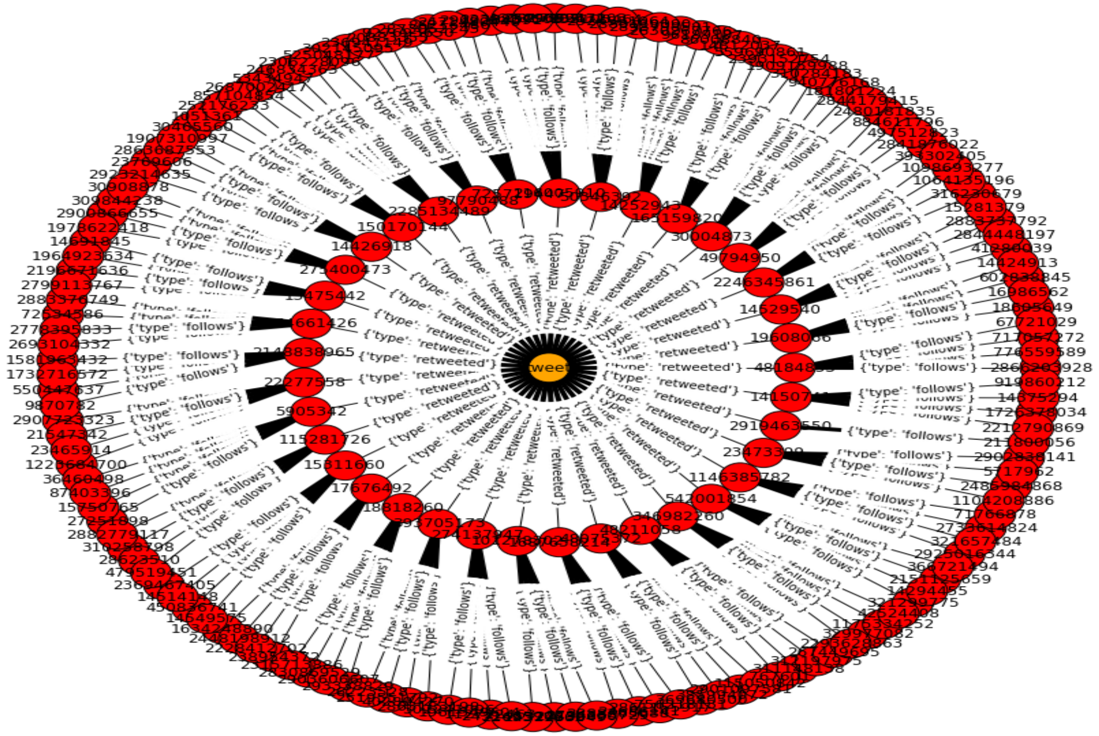
For example, we have tried to crawl some part of the data science community with our crawl above. However, in graphs, communities can overlap, where nodes can belong to more than one group. In this case, Elon Musk is followed by a lot of people in the DS community, even though one wouldn't consider him to be a member.

Modeling ‘Reach’ and ‘Influence’

Another common thing to do with social networks is to measure how influential a user is. One metric to help determine this is the idea of reach, or how many people see you social media messages that you post.

As an example, we can create a graph model of a tweet, the users who retweeted it, and the number of followers those users have. This gives us an upper bound on the number of people who saw a tweet in their stream.

Here is a very small snippet of a graph built from an example tweet from Hilary Mason


tweet
{'type': 'retweeted'}
{'type': 'follows'}
22851344
97790488
150170144
14426918
15475442
2143838965
22277558
5905342
115281726
15311660
17676492
18818260
48211058
346982260
542001854
1146385782
2919463550
19608066
146295540
2246345861
49794950
30004873
2844179415
2480181835
884611796
497512823
2841876022
393302405
1098693277
1064135196
316230679
15281379
2883737792
2844448197
41280039
14424913
602838845
16986562
18605649
67721029
717057272
776559589
2866203928
919860212
14675294
1726378034
2212790869
211800056
2902838141
5717962
2486984868
1104208886
71766878
2733614824
2925016344
368721494
2151125659
14294455
1907310097
2863687553
23760606
2923214635
30908878
309844238
2900866655
1978622418
14691845
1964923634
2196671636
2799113767
2883376749
72634586
2778395833
2693104332
1581963432
1732716572
550447637
2907723323
21647342
23465914
1226684700
36460498
87403396
15750765
2882779117
310258798
28623510
479519451
2369467405
450836741
1634248890

Measure Reach?

The ‘reach’ therefore would be the sum of the number of incident links on the user nodes who retweeted this tweet. We can then measure this for all tweets from a user, to get an average ‘reach’ measurement for a user.

Questions?