A Fast-and-Dirty* Intro to NetworkX (and D3)

Lynn Cherny

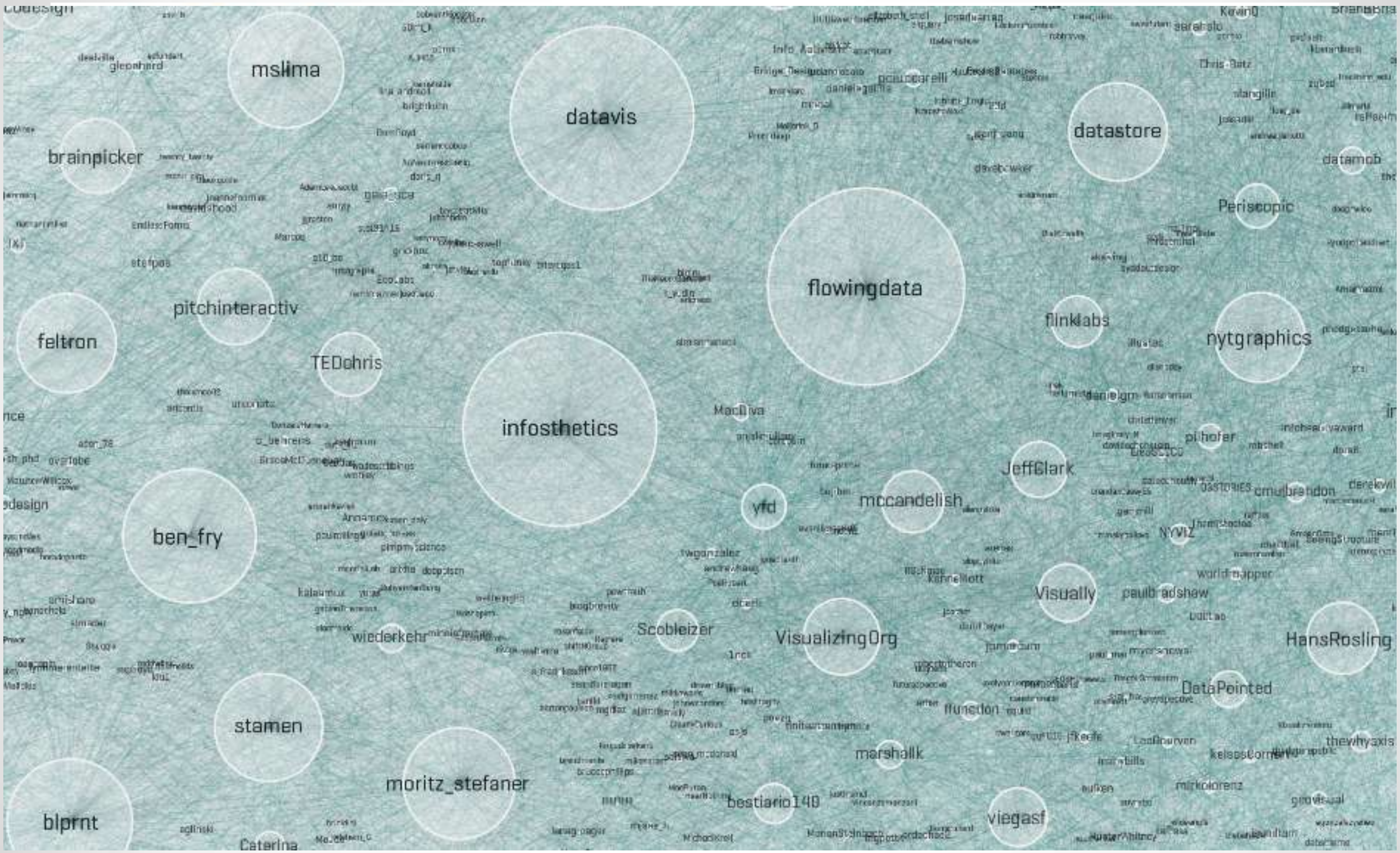
*And, hopefully, practical

Plan

- The Problem: Hairballs.
- NetworkX – one tool
- Stats on networks (and getting them from NetworkX)
- Visualizing networks – some options
- D3 demos of several

- Lots of Links for Learning More

The Problem: Moritz Stefaner's Dataset on Twitter "Infovis" Folks

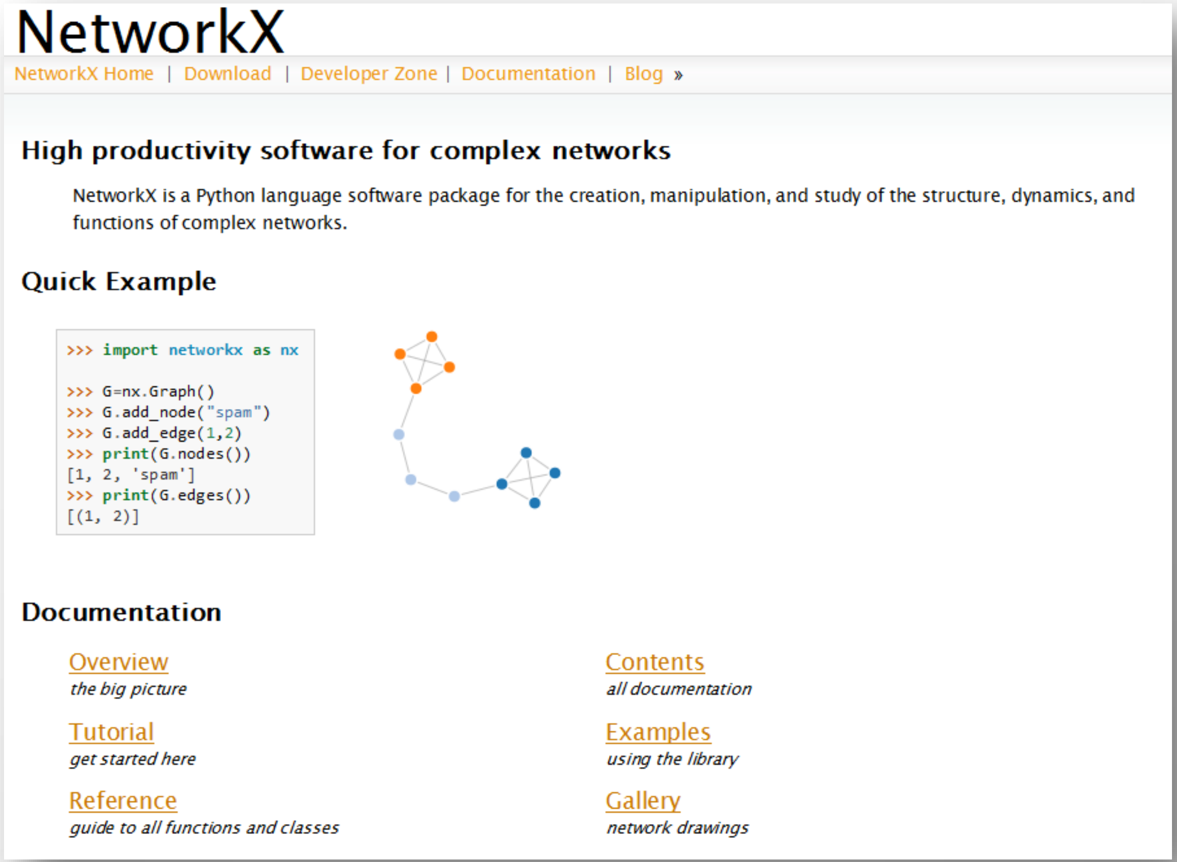


- See http://well-formed-data.net/archives/642/the-vizosphere

Intro to NetworkX

A Python Library for Network / Graph analysis and teaching, housed and documented well at:

http://networkx.lanl.gov/index.html



NetworkX

NetworkX Home | Download | Developer Zone | Documentation | Blog »

High productivity software for complex networks

NetworkX is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

Quick Example

```
>>> import networkx as nx

>>> G=nx.Graph()
>>> G.add_node("spam")
>>> G.add_edge(1,2)
>>> print(G.nodes())
[1, 2, 'spam']
>>> print(G.edges())
[(1, 2)]
```

Documentation

Overview *the big picture*	Contents *all documentation*
Tutorial *get started here*	Examples *using the library*
Reference *guide to all functions and classes*	Gallery *network drawings*

Aside on My Overall Code Strategy

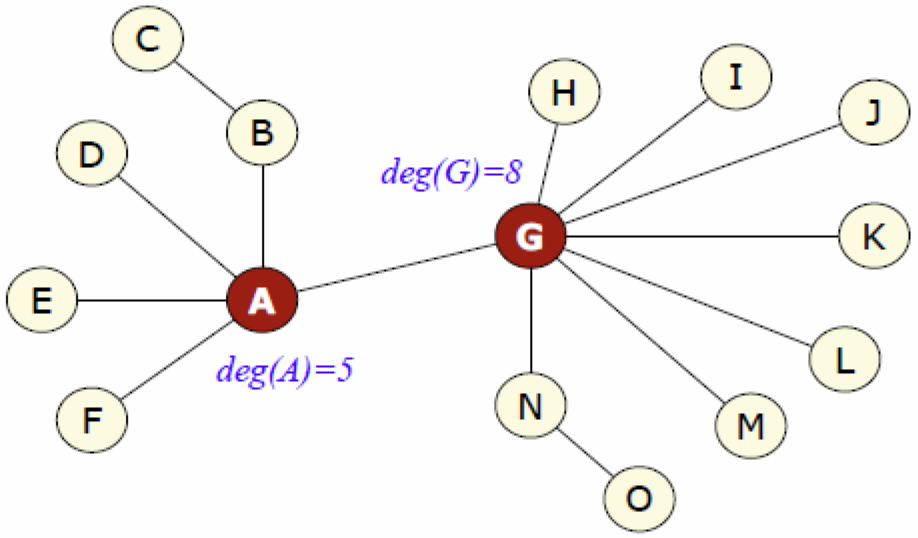
1. Read in edgelist to NetworkX / (or read in JSON)
2. Convert to NetworkX graph object
3. Calculate stats & save values as node attributes in the graph

   (Verify it's done with various inspections of the objects)
4. Write out JSON of nodes, edges and their attributes to use elsewhere
5. Move to D3 to visualize.
6. Go back to 1 and restart to revise stats.

Reduce the problem: send fewer nodes to JSON; or filter visible nodes in UI/vis

Degree (Centrality)

- A variety of different measures exist to measure the importance, popularity, or social capital of a node in a social network.
- Degree centrality focuses on individual nodes - it simply counts the number of edges that a node has.
- Hub nodes with high degree usually play an important role in a network. For directed networks, in-degree is often used as a proxy for popularity.



http://mlg.ucd.ie/files/summer/tutorial.pdf

Example Code from NetworkX

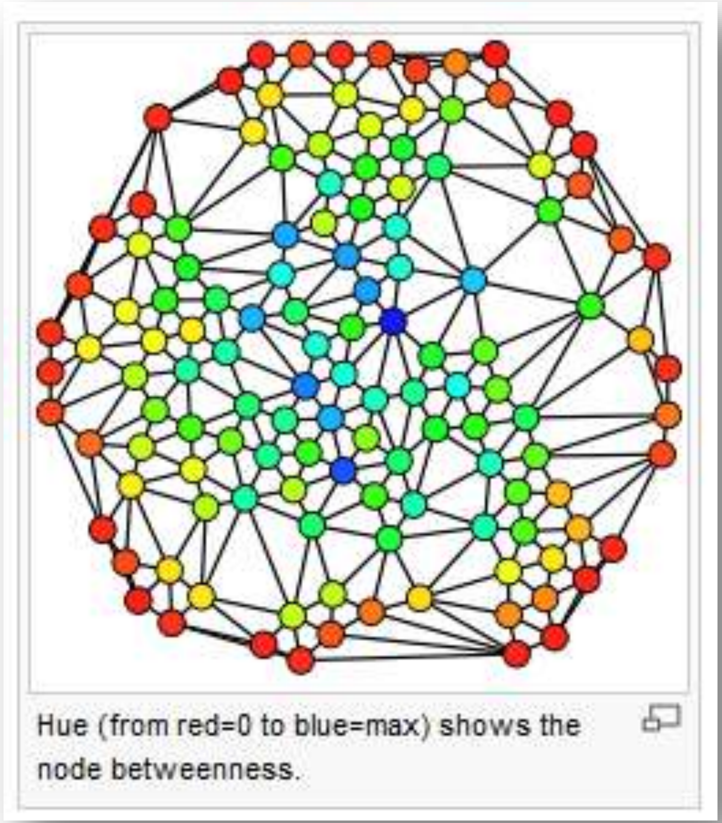
```
def calculate_degree_centrality(graph):
        g = graph
        dc = nx.degree_centrality(g)
        nx.set_node_attributes(g,'degree_cent',dc)
        degcent_sorted = sorted(dc.items(), key=itemgetter(1), reverse=True)
        for key,value in degcent_sorted[0:10]:
                print "Highest degree Centrality:", key, value
        return graph, dc
```

```
Highest degree Centrality: flowingdata 0.848447961047
Highest degree Centrality: datavis 0.837492391966
Highest degree Centrality: infosthetics 0.828971393792
Highest degree Centrality: infobeautiful 0.653682288497
Highest degree Centrality: blprnt 0.567255021302
Highest degree Centrality: ben_fry 0.536822884967
Highest degree Centrality: moritz_stefaner 0.529519172246
Highest degree Centrality: eagereyes 0.524041387705
Highest degree Centrality: mslima 0.503956177724
Highest degree Centrality: VizWorld 0.503956177724
```

There are similar functions for other stats in my code outline.

Betweenness

A measure of connectedness between (sub)components of the graph

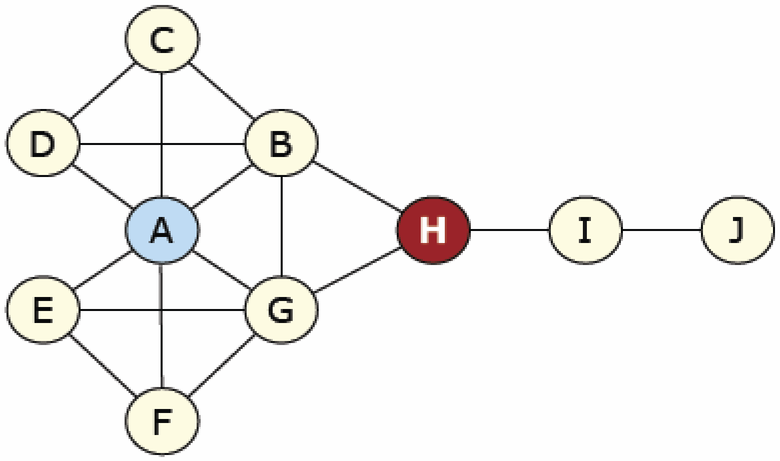

Hue (from red=0 to blue=max) shows the node betweenness.

http://en.wikipedia.org/wiki/Centrality#Betweenness_centrality

Betweenness (Centrality)

- A path in a graph is a sequence of edges joining one node to another. The path length is the number of edges.
- Often want to find the shortest path between two nodes.
- A graph's diameter is the longest shortest path over all pairs of nodes.

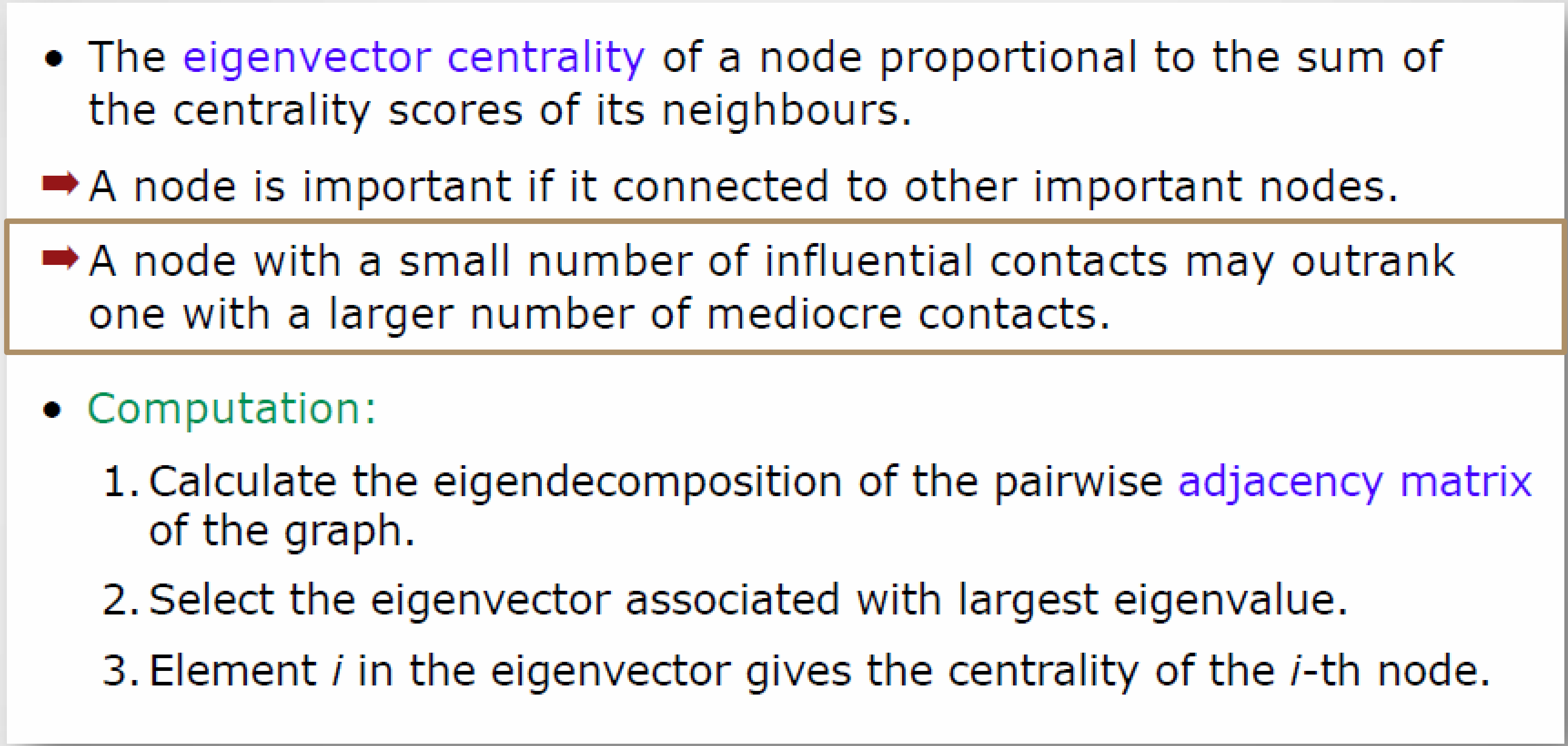
- Nodes that occur on many shortest paths between other nodes in the graph have a high betweenness centrality score.



Node "A" has high degree centrality than "B", as "B" has few direct connections.

Node "H" has higher betweenness centrality, as "H" plays a broker role in the network.

Eigenvector Centrality



- The eigenvector centrality of a node proportional to the sum of the centrality scores of its neighbours.

➡ A node is important if it connected to other important nodes.

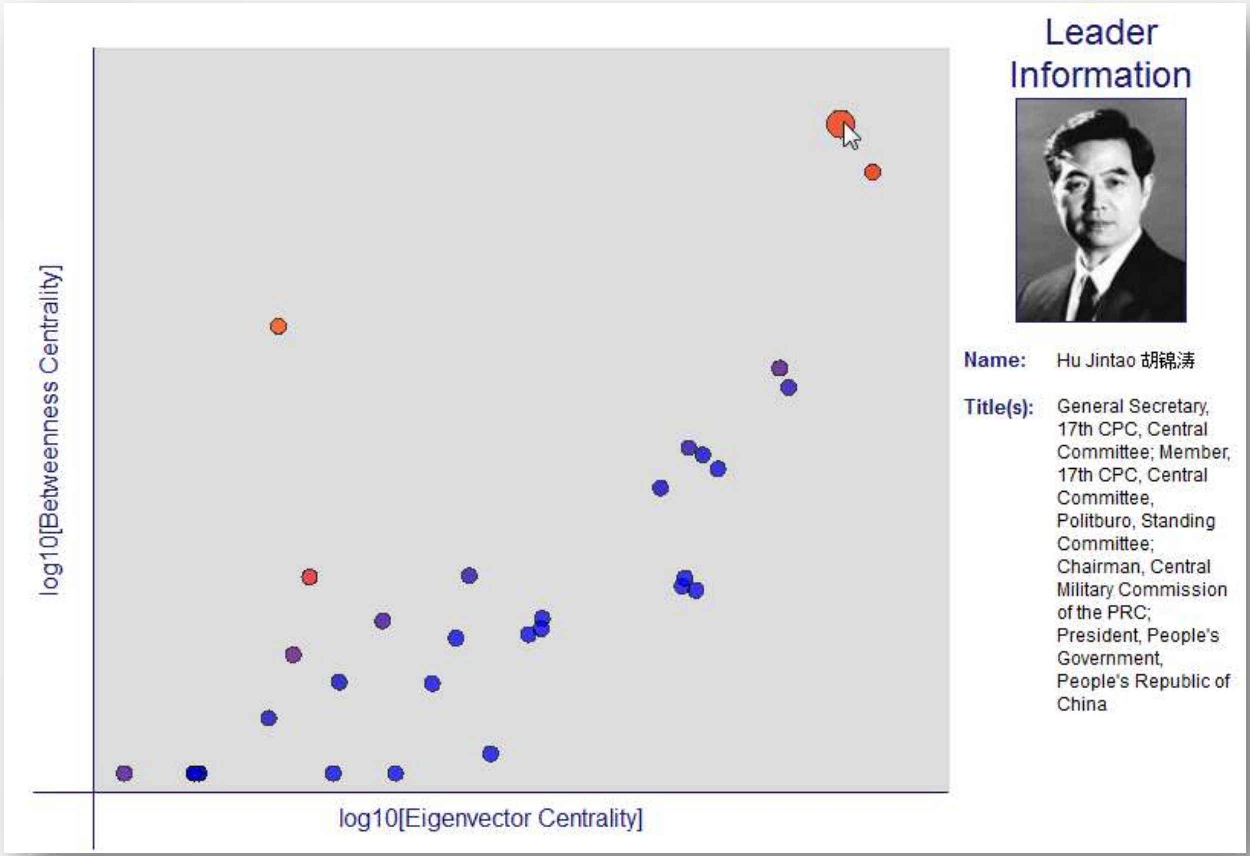
➡ A node with a small number of influential contacts may outrank one with a larger number of mediocre contacts.

- Computation:
  1. Calculate the eigendecomposition of the pairwise adjacency matrix of the graph.
  2. Select the eigenvector associated with largest eigenvalue.
  3. Element *i* in the eigenvector gives the centrality of the *i*-th node.

http://mlg.ucd.ie/files/summer/tutorial.pdf

Use Multiple Stats…

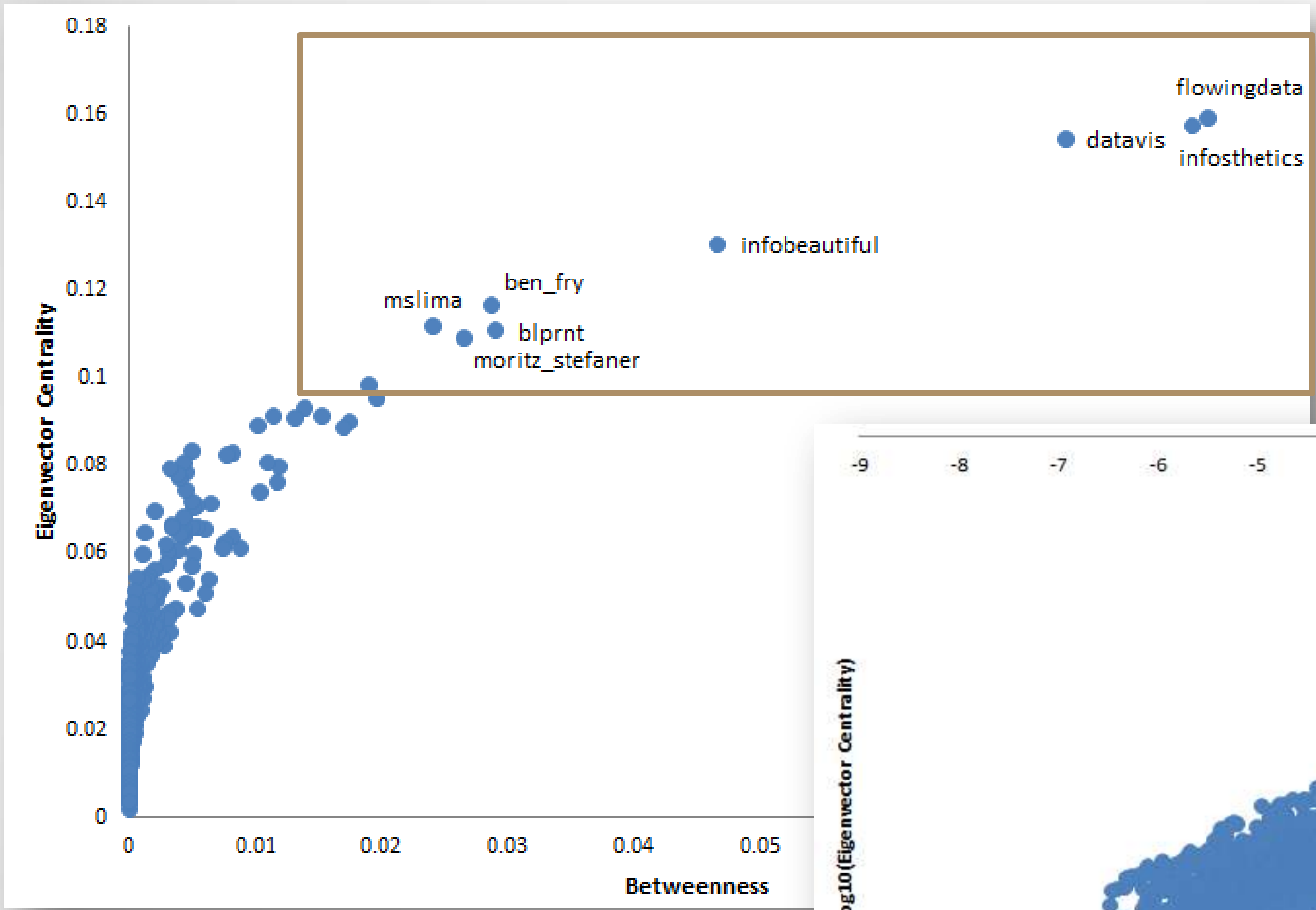
Drew Conway's recent post on central leaders in China:

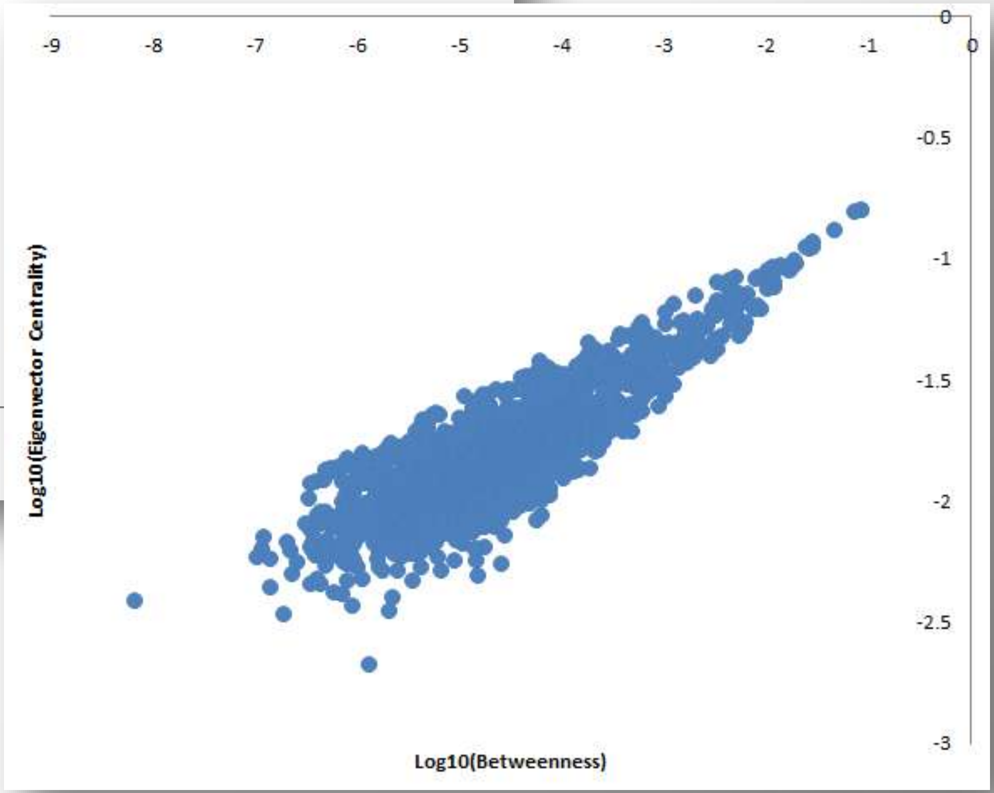


Name:	Hu Jintao 胡锦涛
Title(s):	General Secretary, 17th CPC, Central Committee; Member, 17th CPC, Central Committee, Politburo, Standing Committee; Chairman, Central Military Commission of the PRC; President, People's Government, People's Republic of China

See also the excellent article by Valid Krebs in First Monday on terrorist networks, using other network metrics in conjunction.

Graph: Infovis tweeters from Moritz…

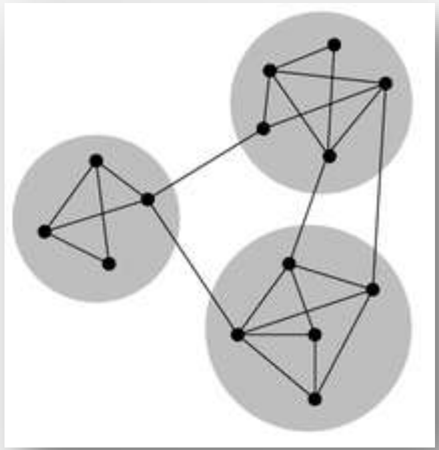




Lynn Cherny,
lynn@ghostweather.com

Community Detection Algorithms

E.g., the Louvain method, implemented in a lib that works with NetworkX

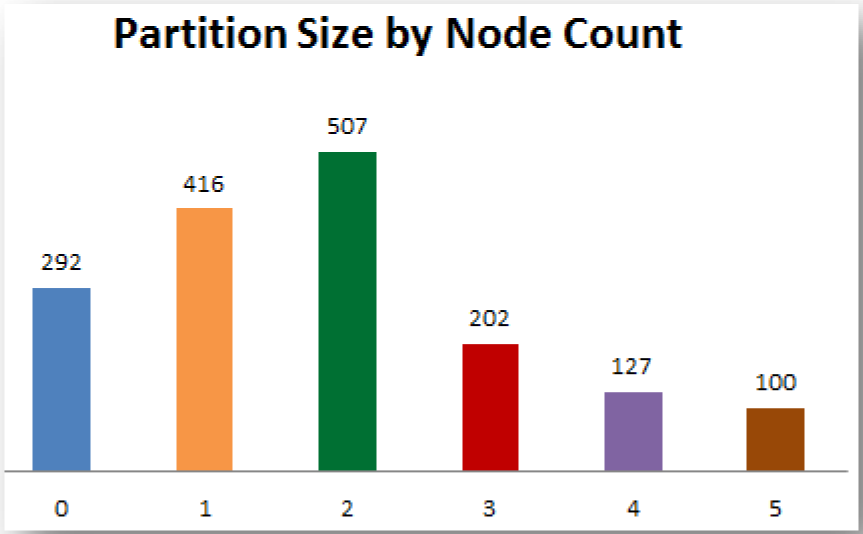


```
def find_partition(graph):
        # from http://perso.crans.org/aynaud/communities/
        g = graph
        partition = community.best_partition( g )
        print "Partitions found: ", len(set(partition.values()))
        print "Partition for node Arnicas: ", partition["arnicas"]
        nx.set_node_attributes(g,'partition',partition)
        return g, partition
```

http://en.wikipedia.org/wiki/File:Network_Community_Structure.png

Dump Partition Number by Node

```
def write_node_attributes(graph, attributes):
        # utility function to let you print the node + various attributes in a csv format
        if type(attributes) is not list:
                attributes = [attributes]
        for node in graph.nodes():
                vals = [str(dict[node]) for dict in [nx.get_node_attributes(graph,x) for x in attributes]]
                print node, ",", ",".join(vals)
```



Get Yer Stats, Visualize.

• • •

Repeat.

Aside: NetworkX I/O utility functions

- Input -- List of edge pairs in txt file (e.g., “a b”)

  Networkx.read_edgelist converts a file of node pairs to a graph:

```
def read_in_edges(filename):
        g_orig = nx.read_edgelist(filename, create_using=nx.DiGraph())
        print "Read in edgelist file ", filename
        print nx.info(g_orig)
        return g_orig
```

- Input or Output -- JSON

  NetworkX.readwrite.json_graph.node_link_data

```
def save_to_jsonfile(filename, graph):
        g = graph
        g_json = json_graph.node_link_data(g)
        json.dump(g_json, open(filename,'w'))
```

  NetworkX.readwrite.json_graph.load

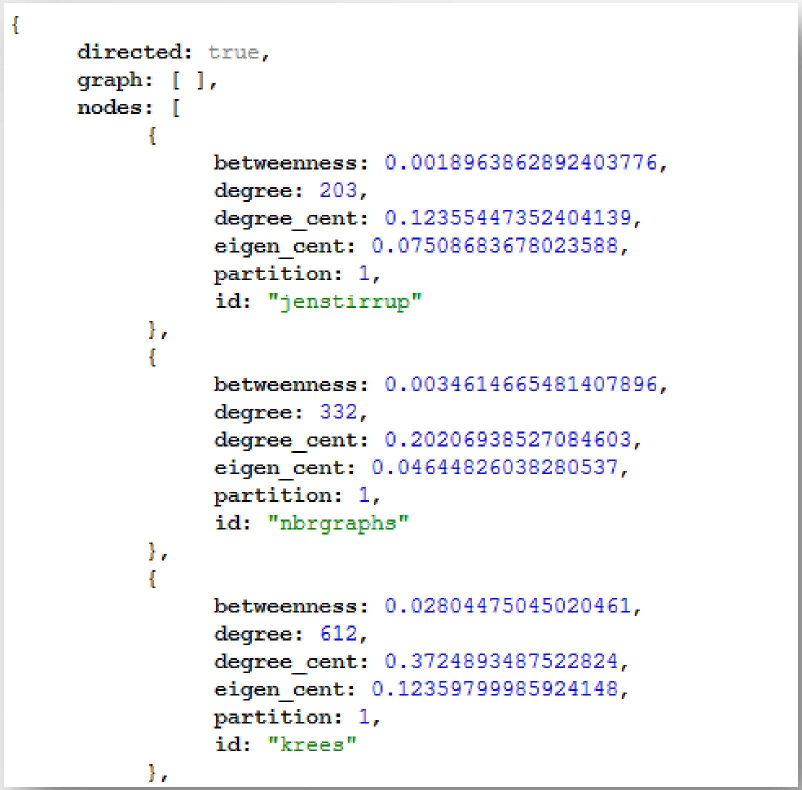
```
def read_json_file(filename):
        graph = json_graph.load(open(filename))
        print "Read in file ", filename
        print nx.info(data)
        return graph
```

Saving a Subset…

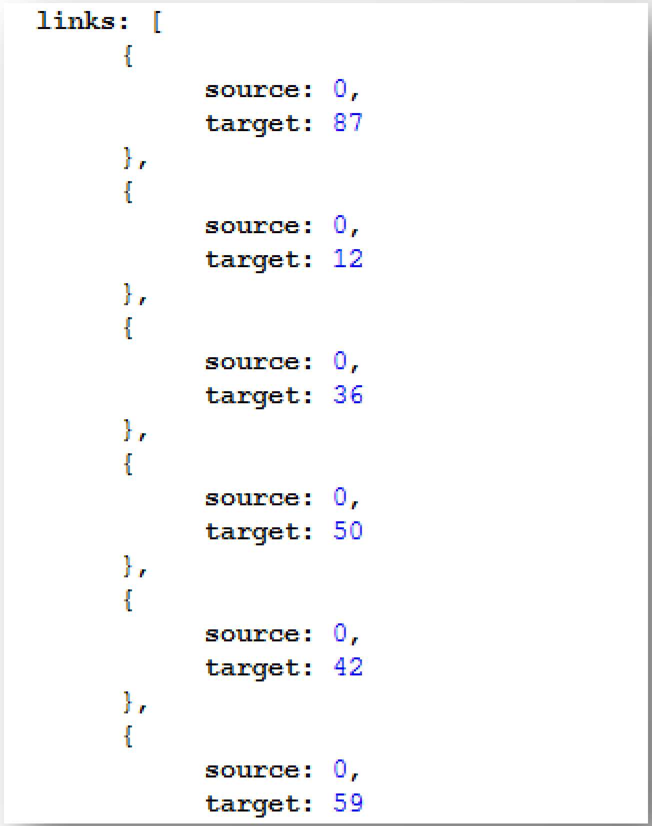
- For most of my visualization demos, I used a subset of the full dataset.   I sorted the 1644 nodes by eigenvector centrality score and then saved only the top 100.
- Code from my networkx_functs.py file:

```
eigen_sorted = sorted(eigen.items(), key=itemgetter(1), reverse=True)

for key, val in eigen_sorted[0:5]:
                print “Highest eigenvector centrality nodes:", key, val

# for trimming the dataset, you want it reverse sorted, with low values on top.
eigen_sorted = sorted(eigen.items(), key=itemgetter(1), reverse=False)
small_graph = trim_nodes_by_attribute_for_remaining_number(undir_g, eigen_sorted, 100)

print nx.info(small_graph)

#save as json for use in javascript - small graph, and full graph if you want
save_to_jsonfile(path+outputjsonfile, small_graph)
```

Dump JSON of a graph
(after my NetworkX calcs)



```
{
    directed: true,
    graph: [ ],
    nodes: [
        {
            betweenness: 0.0018963862892403776,
            degree: 203,
            degree_cent: 0.12355447352404139,
            eigen_cent: 0.07508683678023588,
            partition: 1,
            id: "jenstirrup"
        },
        {
            betweenness: 0.0034614665481407896,
            degree: 332,
            degree_cent: 0.2020693852708460З,
            eigen_cent: 0.04644826038280537,
            partition: 1,
            id: "nbrgraphs"
        },
        {
            betweenness: 0.02804475045020461,
            degree: 612,
            degree_cent: 0.3724893487522824,
            eigen_cent: 0.12359799985924148,
            partition: 1,
            id: "krees"
        },
```



```
links: [
    {
        source: 0,
        target: 87
    },
    {
        source: 0,
        target: 12
    },
    {
        source: 0,
        target: 36
    },
    {
        source: 0,
        target: 50
    },
    {
        source: 0,
        target: 42
    },
    {
        source: 0,
        target: 59
```

Works with all D3 examples I'll show…

Gotchas to be aware of here

- If you don’t use the “DiGraph” (directed graph) class in NetworkX, you will lose some links. This changes some visuals.

- Your json links are based on index of the node. If/when you do filtering in JSON based on, say, UI controls, you need to redo your indexing on your links!
  [e.g., See my code in demo full_fonts.html]

Visualizing Networks

NetworkX isn't really for vis – can use graphViz and other layouts for static pics.

Use Gephi to explore and calculate stats, too.

See my blog post and slideshare with UI screencaps of Gephi, using this same data set!

Apart from the hairball, there are other methods to visualize graphs:

- See Robert Kosara's post: http://eagereyes.org/techniques/graphs-hairball
- Lane Harrison's post: http://blog.visual.ly/network-visualizations/
- MS Lima's book Visual Complexity

Like many big data problems, use multiple stats and multiple methods to explore!

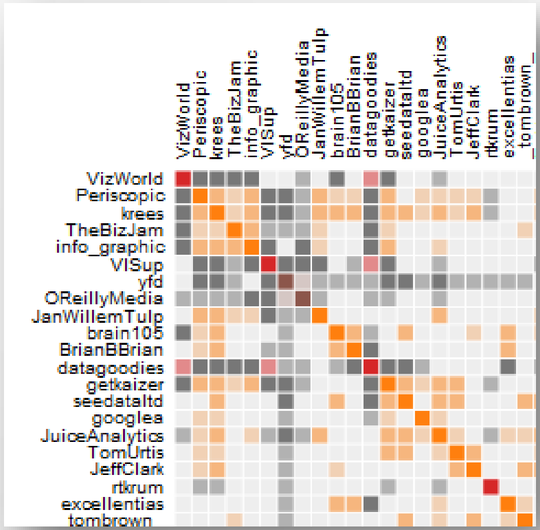
D3.js by Mike Bostock

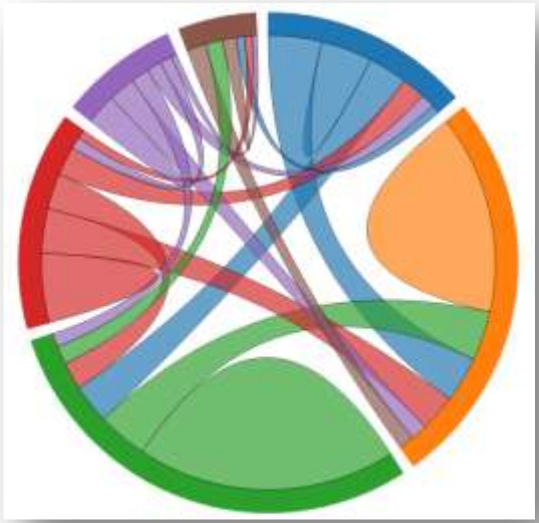
D3 allows creation of interactive visualizations…

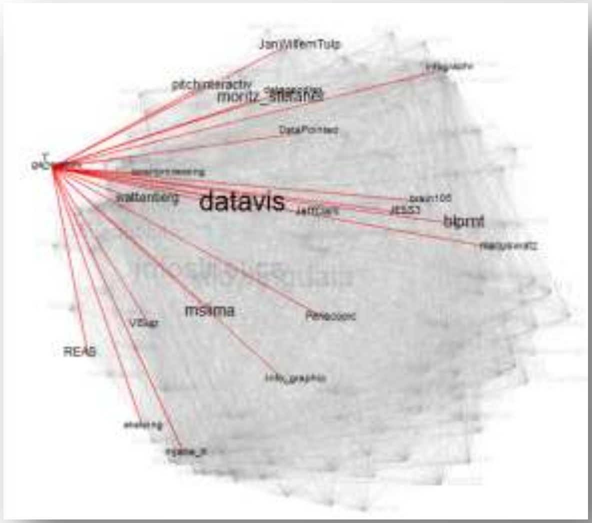
Adjacency Matrix

Chord Diagram

Networks

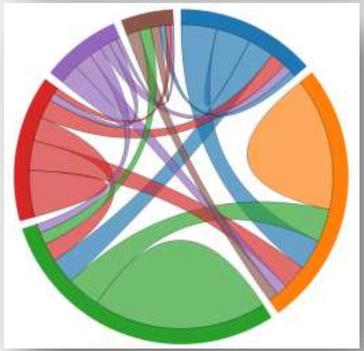




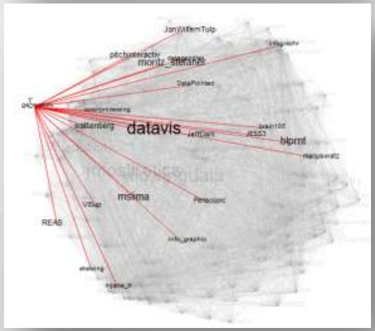


Aside on Data Set Size

Adjacency matrix only holds a small number of nodes at a time – I used 88 of the top 100 selected by eigenvector centrality for this demo.



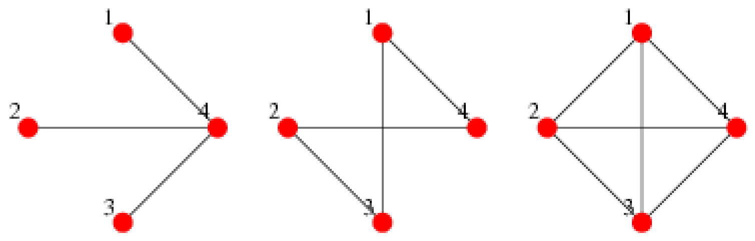
Chord diagrams are simplified reps of a dataset – comparing the relations between the top 100 by eigenvector centrality vs. the whole 1644 nodes set reveals a most interesting insight!



Interactive network vis is limited by browser performance – and complexity of hairballs. If you want it to be interactive (live) and not a static image, you probably need to reduce your data before or after loading.

Adjacency Matrix

Square matrix of nodes; cells constitute edge count.



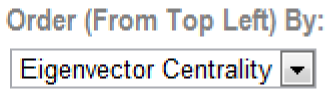
$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

http://mathworld.wolfram.com/AdjacencyMatrix.html

[Demo adjacency.html]

What did this show?

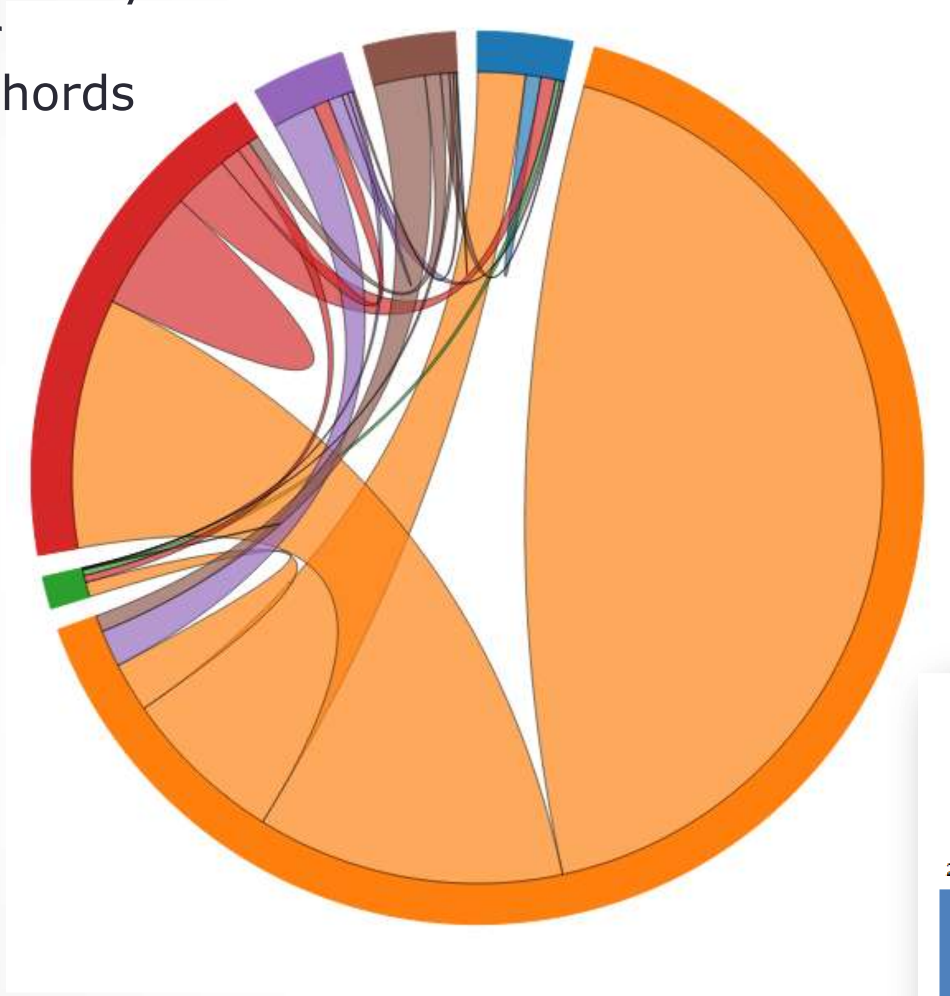
- Be sure to change the sort order on the right side:


Order (From Top Left) By:
Eigenvector Centrality

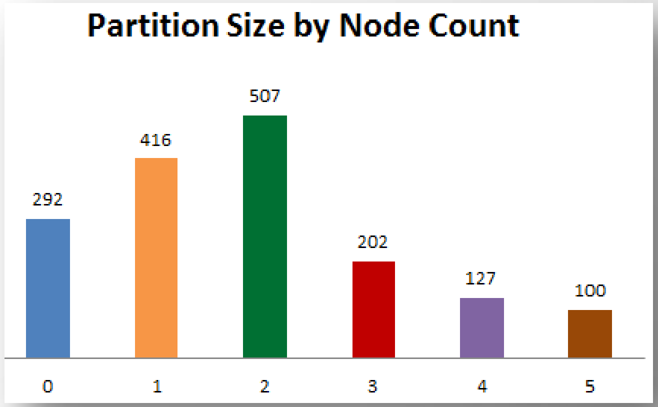
- The largest partition represented is the orange partition, when you sort by partition (= subcommunity)
- Some partitions (colors) have very few representatives in the matrix of the top 88. We can suppose these partitions are not composed of people with the highest eigenvector centrality scores.
- Node **VizWorld** is near or at the top in all the sort-by-attribute methods offered, and is in the red partition, not orange.

Chord Diagram: Summarize Relations

Top 100 nodes by eigenvector centrality, chords by target:

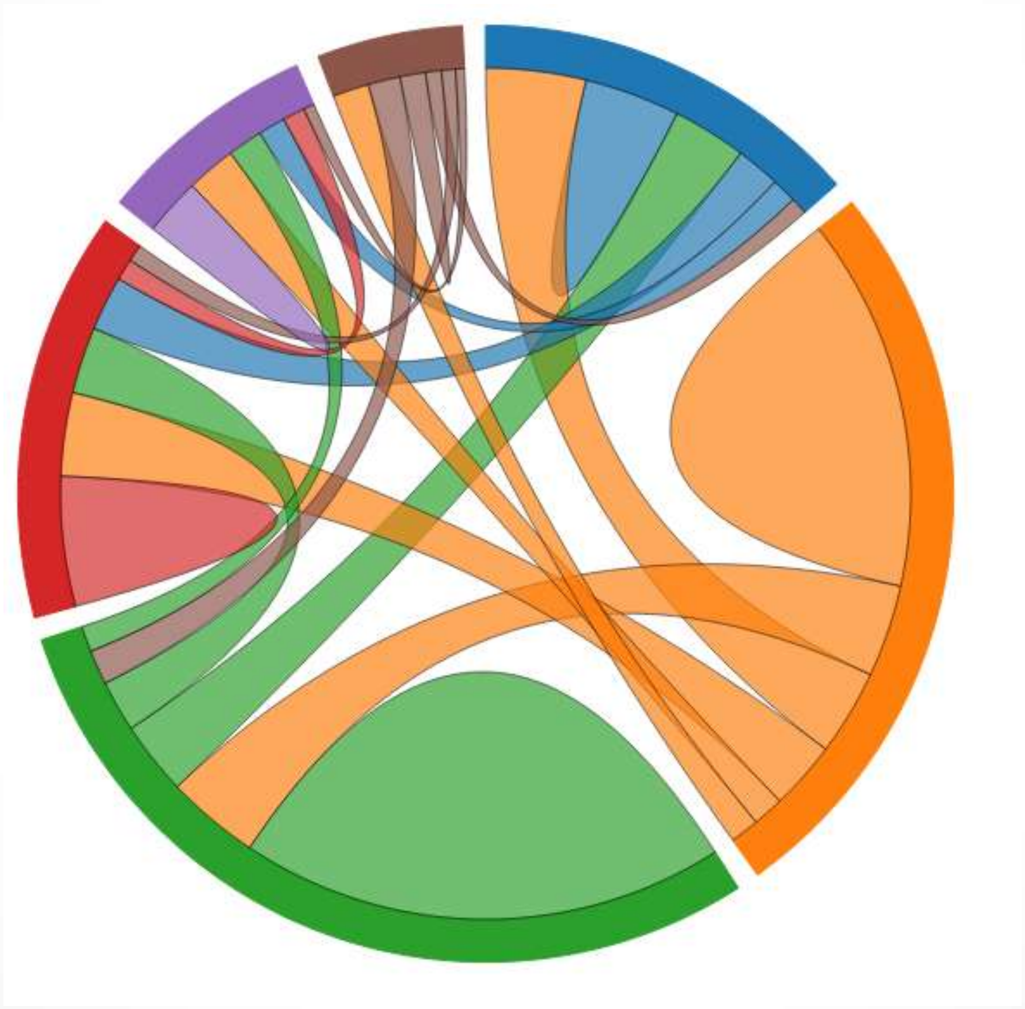


Not representative of full network… Interesting!



Demo

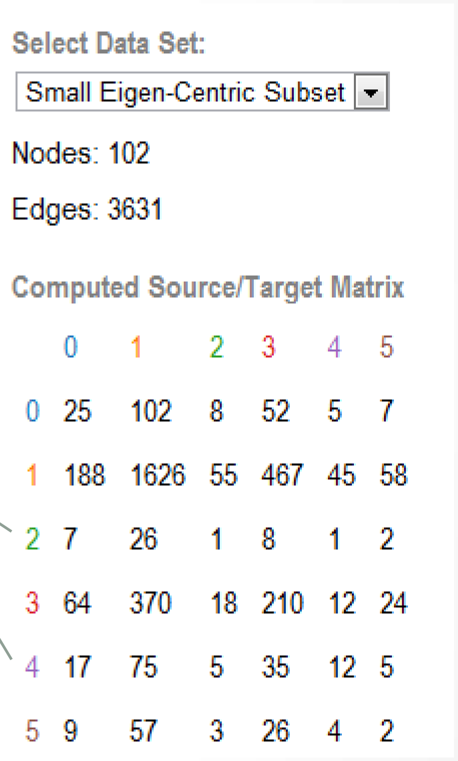
Redo Chord Diagram With All Data…



The demo allows you to swap json files to see the change.

Insights from Comparing Them

- The top 100 nodes by eigenvector centrality are mostly the orange partition. The green partition, however, is the largest group in the unfiltered set (the whole 1644 nodes).
  - Notice how few green and purple partition members "make" the top eigencentric list:
- You can see supporting evidence of the orange eigenvector centrality by looking at how many people link to them from other partitions. Change the target/source radio button on the demo to see this in action.



Select Data Set:

Small Eigen-Centric Subset

Nodes: 102

Edges: 3631

Computed Source/Target Matrix

	0	1	2	3	4	5
0	25	102	8	52	5	7
1	188	1626	55	467	45	58
2	7	26	1	8	1	2
3	64	370	18	210	12	24
4	17	75	5	35	12	5
5	9	57	3	26	4	2

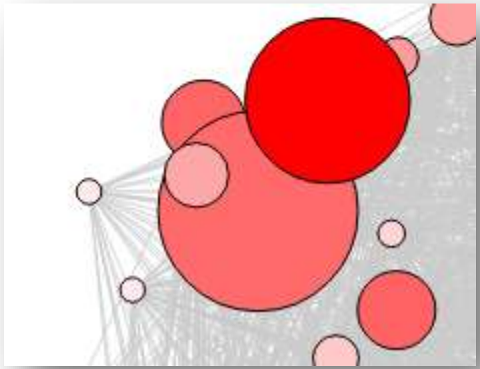
Handling with Graph Rendering…

- Typical Nodes/Edges, with sizing/coloring – slow, still a hairball, not much visible useful info.

  Avoid this simplistic vis method if you can…

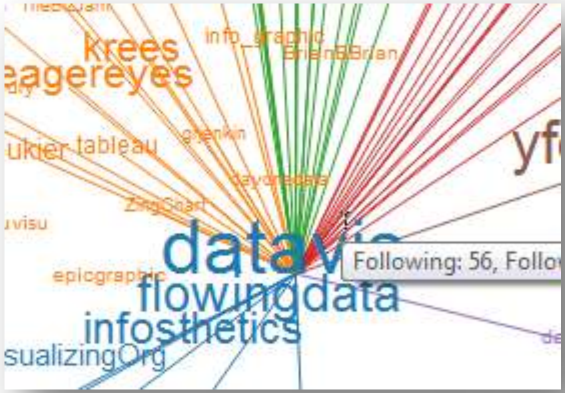
  Note this takes a little while to calm down!

  [Demo redballs]



- Alternate, slightly better: Names, color by Partition, UI options and edges on click.

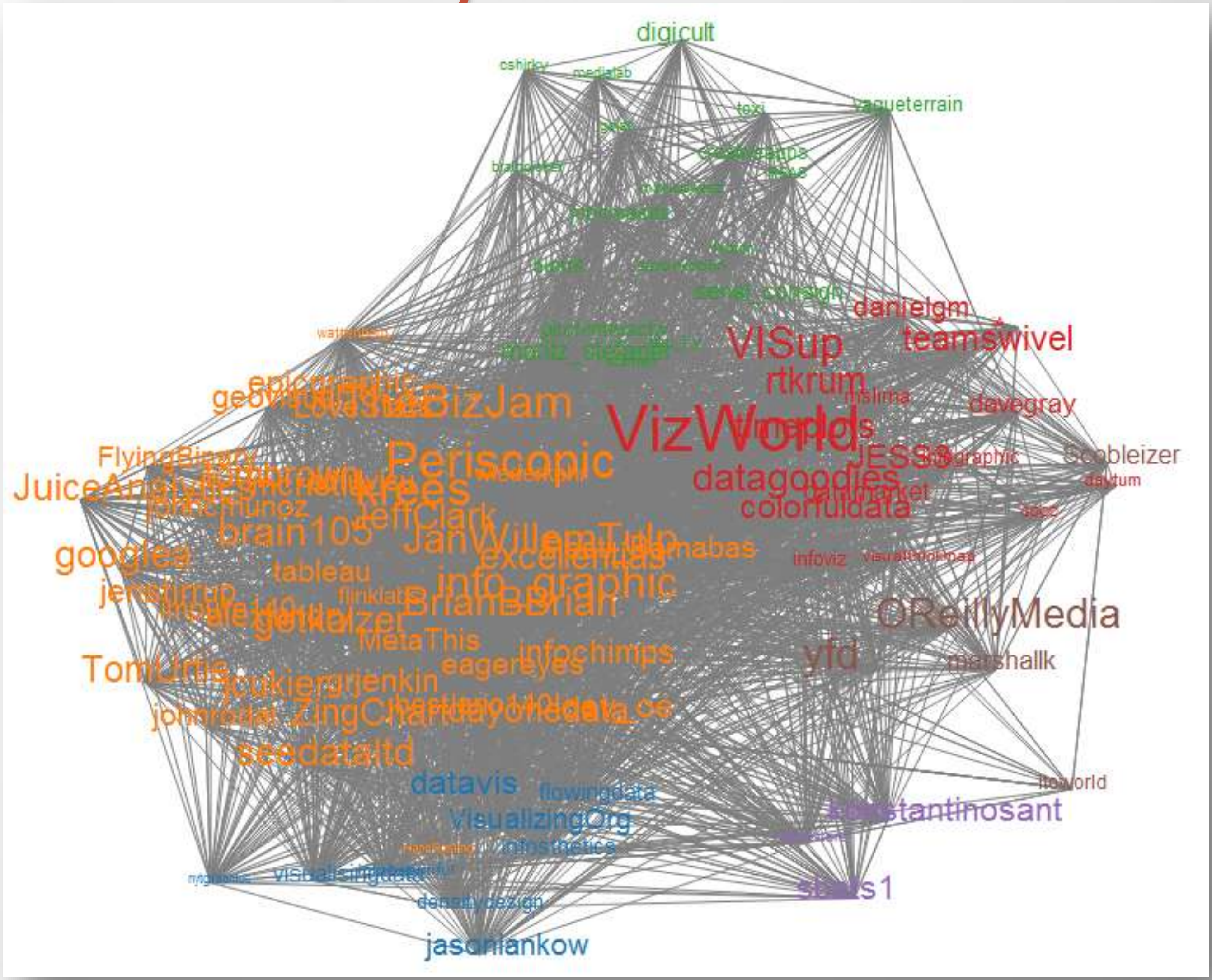
  [Demo force_fonts]



Viewing the top scoring subset only....

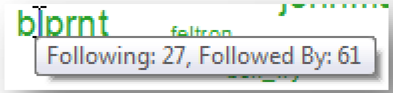
Even with a small subset and partition clustering, showing all the links is a visual mess…

So only show them on demand.



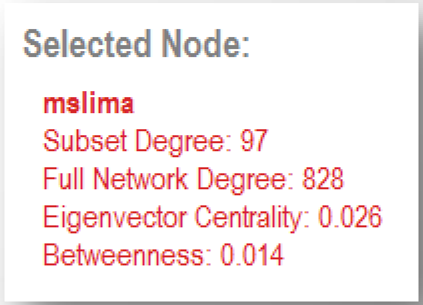
Design Tweaks Made To Make It (More) Useful

- Add a click-action to
  - Fade out nodes unrelated to clicked node
  - Show lines indicating who they follow
  - Show the names (unlinked) of who follows them
- Add a tooltip showing on-screen degree (i.e., following and followed-by numbers for the subset)



Following: 27, Followed By: 61

- Heavily adjusted layout to separate clusters visually (lots of trial and error, see following slides)
- Add stats on the sidebar showing some numbers, to allow you to compare, for instance, onscreen degree vs. degree in the whole set of 1644 nodes:

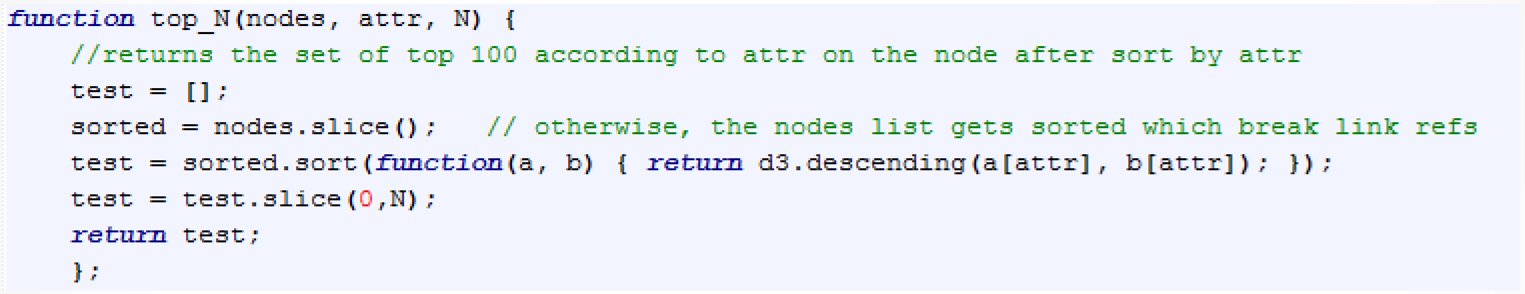


Selected Node:

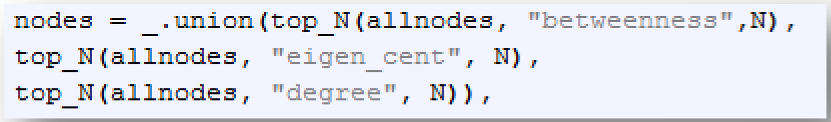
mslima
Subset Degree: 97
Full Network Degree: 828
Eigenvector Centrality: 0.026
Betweenness: 0.014

Creating the subset in JS instead of NetworkX

- To create the union of the top N by each attribute, I shortcutted and used underscore.js's union function:



```
function top_N(nodes, attr, N) {
    //returns the set of top 100 according to attr on the node after sort by attr
    test = [];
    sorted = nodes.slice();   // otherwise, the nodes list gets sorted which break link refs
    test = sorted.sort(function(a, b) { return d3.descending(a[attr], b[attr]); });
    test = test.slice(0,N);
    return test;
    };
```

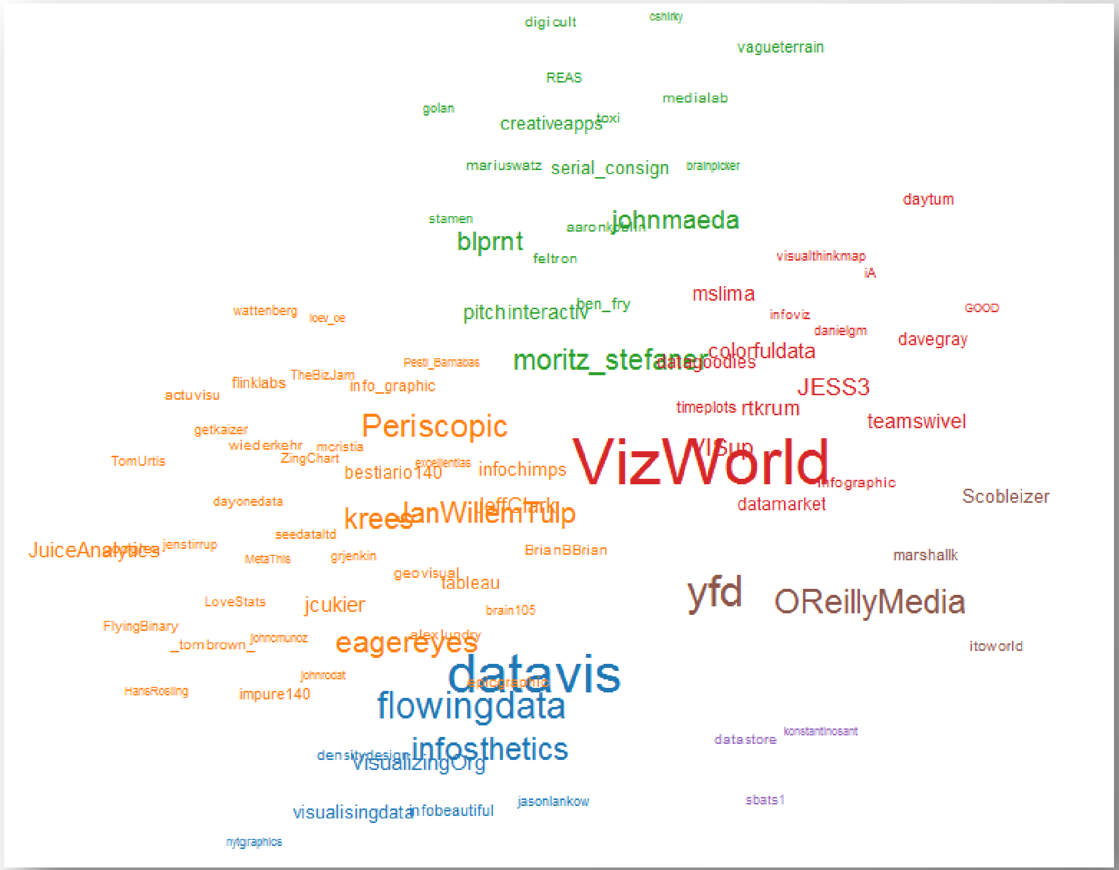


```
nodes = _.union(top_N(allnodes, "betweenness",N),
top_N(allnodes, "eigen_cent", N),
top_N(allnodes, "degree", N)),
```

- Then you need to update your links by filtering for the links referring to the subset of nodes, and fix the indices!

Insights from Subset

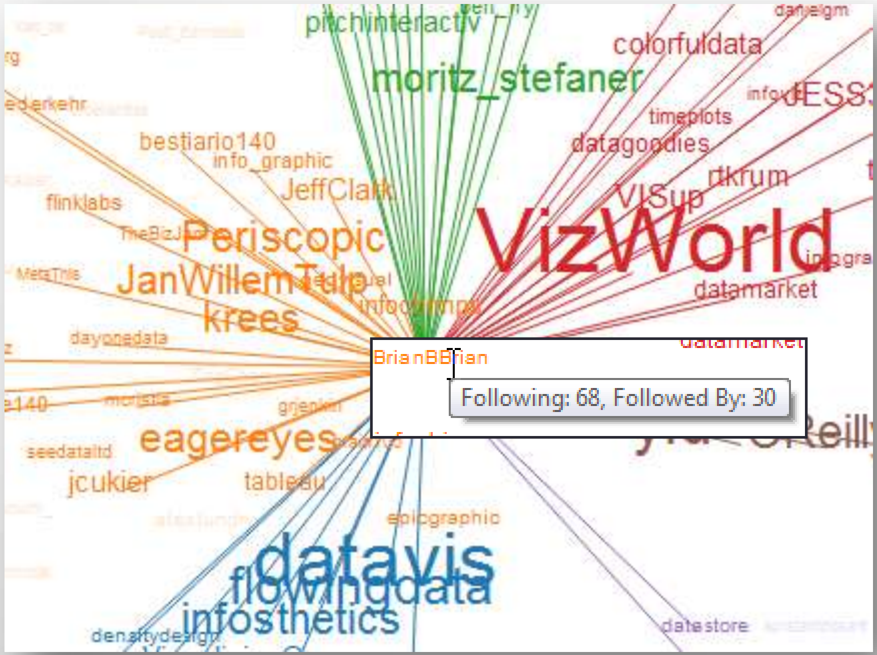
The most "readable" view is with fonts sized by "Betweenness" because of the large discrepancies:



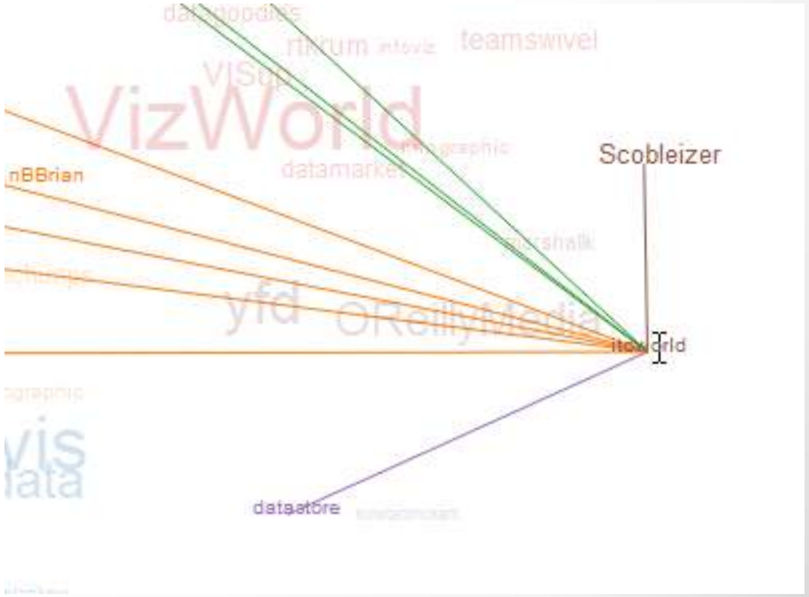
Note that these look familiar from the Adjacency Matrix view!

How to "read" the layout

- Nodes that drift towards the middle are linked to more partition colors and nodes in the visible subset. Tooltips show the following/follower relations for the subset only.
- Nodes towards the fringes are less linked in general inside this subset.



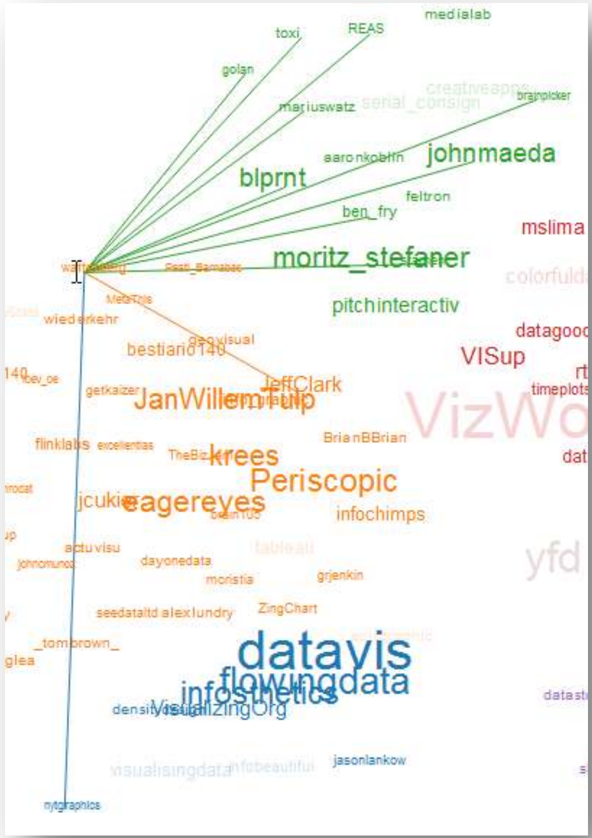
Itoworld, on the edge:



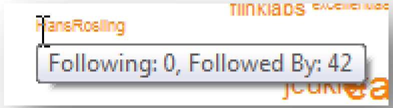
Interesting Oddities

wattenberg is in the orange partition, but within the top N nodes, follows* mostly green:

In general, the top N of the green partition follow each other. They're the artists!

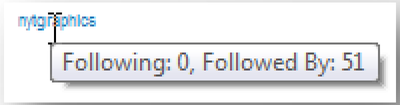


HansRosling follows no one but is followed by quite a few of the visible subset:


HansRosling

Following: 0, Followed By: 42

Ditto nytgraphics:

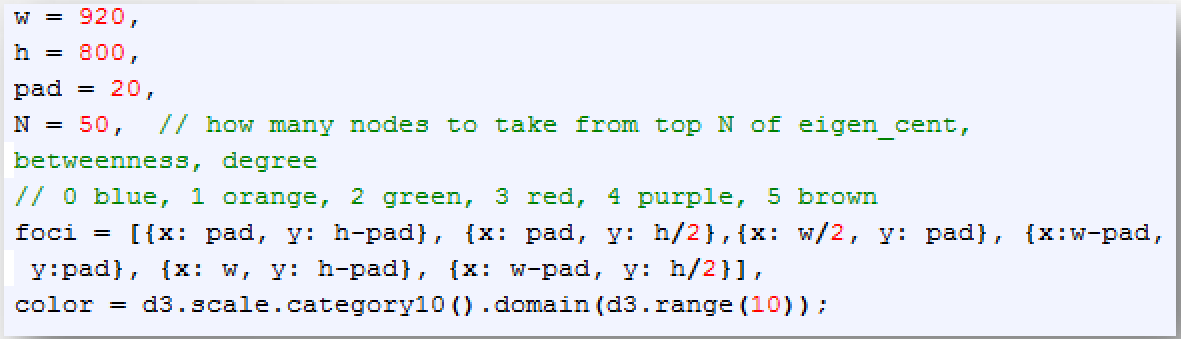

nytgraphics

Following: 0, Followed By: 51

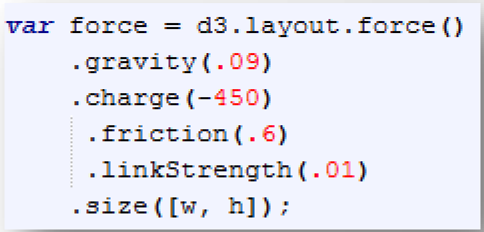
* This data set is from mid-2011!

Accomplishing the layout in D3

- Lots of screwing with the foci locations to move the nodes to and force settings:

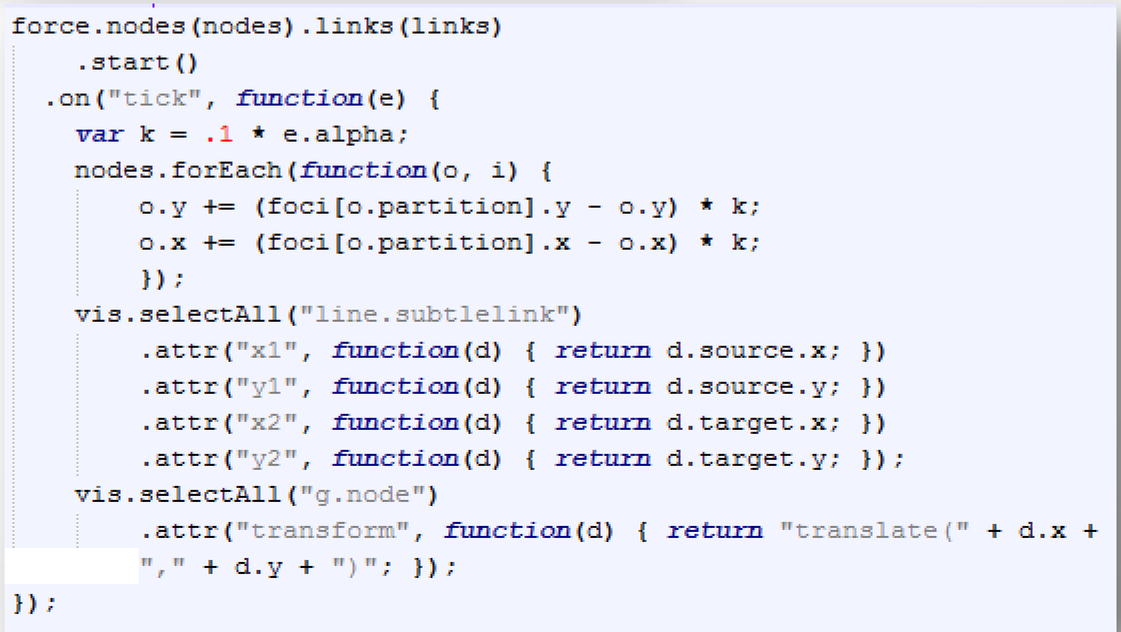


```
w = 920,
h = 800,
pad = 20,
N = 50,  // how many nodes to take from top N of eigen_cent,
betweenness, degree
// 0 blue, 1 orange, 2 green, 3 red, 4 purple, 5 brown
foci = [{x: pad, y: h-pad}, {x: pad, y: h/2},{x: w/2, y: pad}, {x:w-pad,
 y:pad}, {x: w, y: h-pad}, {x: w-pad, y: h/2}],
color = d3.scale.category10().domain(d3.range(10));
```



```
var force = d3.layout.force()
    .gravity(.09)
    .charge(-450)
     .friction(.6)
     .linkStrength(.01)
    .size([w, h]);
```

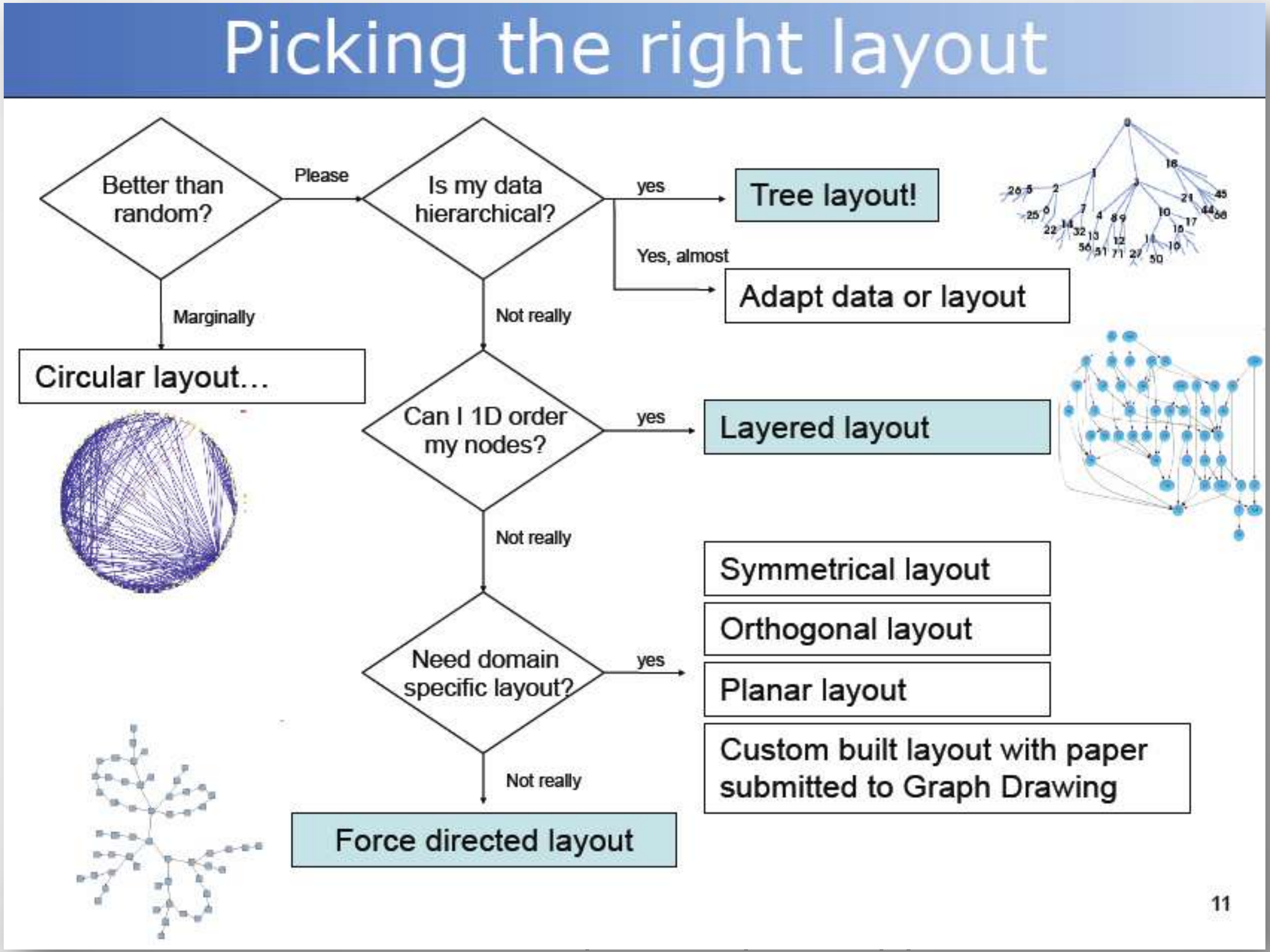
- Moving them:



```
force.nodes(nodes).links(links)
    .start()
  .on("tick", function(e) {
    var k = .1 * e.alpha;
    nodes.forEach(function(o, i) {
        o.y += (foci[o.partition].y - o.y) * k;
        o.x += (foci[o.partition].x - o.x) * k;
        });
    vis.selectAll("line.subtlelink")
        .attr("x1", function(d) { return d.source.x; })
        .attr("y1", function(d) { return d.source.y; })
        .attr("x2", function(d) { return d.target.x; })
        .attr("y2", function(d) { return d.target.y; });
    vis.selectAll("g.node")
        .attr("transform", function(d) { return "translate(" + d.x +
        "," + d.y + ")"; });
});
```

Lynn Cherny,
lynn@ghostweather.com

Wrap up with an aside from a Frank van Ham Talk



Reminder(s)

The map is not the territory.

Just cuz social media software tools allow links between people doesn't mean they reflect the true – or complete – social network of relationships.

(Also, this data set is no doubt out of date with respect to current follower relations!)

A Bunch of Links

(In the non-network sense)

General Network Primer Material

- MIT OpenCourseware on Networks, Complexity, Applications (many references!)
- Frank van Ham's slides from a recent datavis meetup
- CRAN R code/links for handling graphs/networks
- Chapter 10 of Rajaraman & Ullman and book on Data Mining of Massive Datasets
- Graph Theory with Applications by Bondy and Murty
- Intro to Social Network Methods by Hanneman and Riddle
- Networks, Crowds, and Markets by Easley and Kleinberg

- My lists of sna / networks papers on delicious

NetworkX Info/Tutorials

- NetworkX site docs/tutorial: http://networkx.lanl.gov/tutorial/index.html
- UC Dublin web science summer school data sets, slides, references: http://mlg.ucd.ie/summer
- Stanford basic intro tutorial: http://www.stanford.edu/class/cs224w/nx_tutorial/nx_tutorial.pdf

D3 Example Links (for networks)

- D3.js – Mike Bostock
- Super Useful force attributes explanation from Jim Vallandingham
- D3 Demo Talk Slides with embedded code by MBostock
- Chicago Lobbyists by Manning
- Mobile Patent Suits by Mbostock
- Rollover selective highlight code by Manning
- D3 Adjacency Matrix by Mbostock
- Chord diagram: http://bost.ocks.org/mike/uberdata/

- My giant, growing list of D3 links on delicious

Community Detection (a couple)

Overlapping Community Detection in Networks: State of the Art and Comparative Study by Jierui Xie, Stephen Kelley, Boleslaw K. Szymanski

Empirical Comparison of Algorithms for Network Community Detection by Leskovec, Lang, Mahoney

Sources of "Canned" Network Data

- Koblenz Network Collection
- CMU's CASOS
- INSNA.org's member datasets

Blog Post and Links

- Zip file of slides, networkx code, and edgelist:
  - http://www.ghostweather.com/essays/talks/networkx/source.zip
- Blog post with links and more text:
  - http://blogger.ghostweather.com/2012/03/digging-into-networkx-and-d3.html

GhostWeather
Research & Design

Lynn Cherny

lynn@ghostweather.com

@arnicas

http://www.ghostweather.com