

# CS 163 & CS 265: Graph Algorithms

## Week 6: Social networks

### Lecture 16: Centrality, degeneracy, and cores

**David Eppstein**

University of California, Irvine

Fall Quarter, 2022



This work is licensed under a Creative Commons Attribution 4.0 International License

# Centrality

# Centrality

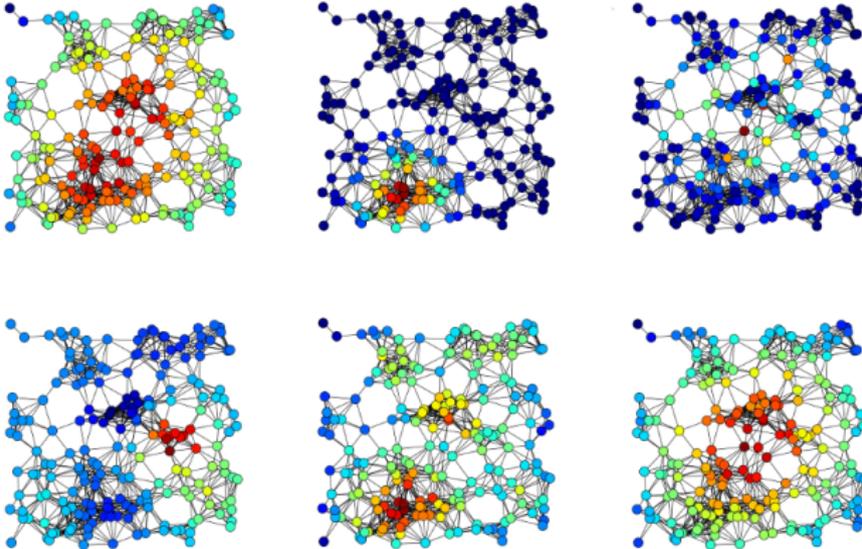
We want to measure the importance of vertices by giving them numeric scores, bigger numbers for vertices that are bigger or more central

For example

- ▶ In the actor network, Kevin Bacon should have high centrality
- ▶ Among mathematicians, Paul Erdős should have a big number

# There are many types of centrality

Two that we've already seen: degree and pagerank



Top: harmonic centrality, eigenvector cent., betweenness cent.

Bottom: Katz cent., degree cent., closeness cent.

[https://commons.wikimedia.org/wiki/File:6\\_centrality\\_measures.png](https://commons.wikimedia.org/wiki/File:6_centrality_measures.png)

# Closeness centrality

Intuition: central vertices are the ones close to everyone else  
(like Bacon and Erdős)

$$\text{centrality}(v) = \frac{1}{\text{average distance to other vertices}}$$

The  $1/x$  part of the formula makes small distances into big centralities and vice versa

To compute this exactly, at all vertices:

- ▶ Compute all-pairs shortest paths (repeated BFS)
- ▶ Average the distances, compute  $1/\text{average}$
- ▶ Time  $O(mn)$

## Fast approximate closeness centrality

Instead of computing all-pairs distances,

- ▶ Choose a sample of  $k$  random vertices
- ▶ Compute single-source distances (BFS) from each sample vertex, time  $O(km)$
- ▶ Estimate centrality as  $1/(\text{average distance to samples})$  instead of averaging the distance to all vertices

Distance to samples is an **unbiased estimator** of distance to all vertices (has correct expected value), could have high variance

For graphs **with small world property** (all distances  $\leq D$ ), standard results on averaging small random variables tell us the variance is low  $\Rightarrow$  estimate is likely to be close to correct with high probability

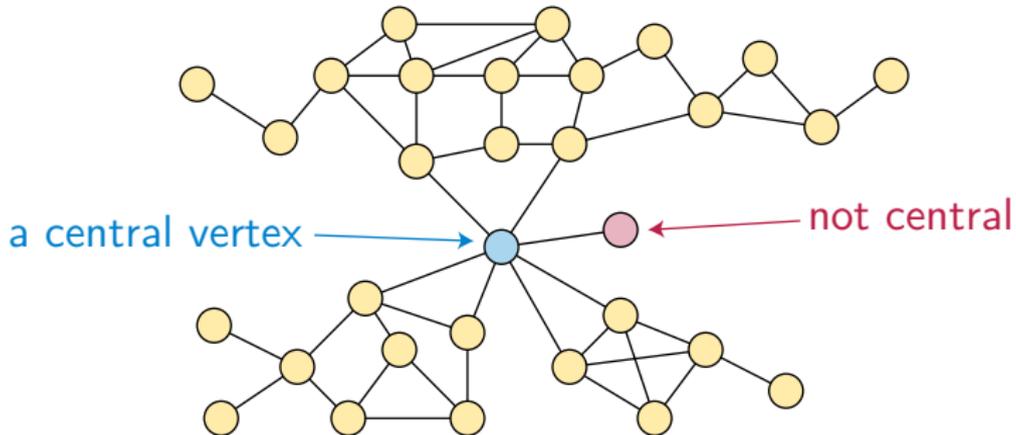
So setting  $k \approx D \log n / \varepsilon^2$  gives a  $(1 + \varepsilon)$ -approximation w.h.p.

[Eppstein and Wang 2004]

# Betweenness centrality

Main idea:

A central vertex should be on many shortest paths, not just nearby



# Betweenness centrality

Definition:

$$\text{centrality}(v) = \sum_{u, w \neq v} \frac{\# \text{ shortest paths from } u \text{ to } w \text{ through } v}{\# \text{ shortest paths from } u \text{ to } w}$$

Traditional algorithm,  $O(n^3)$ :

- ▶ For each  $u$  and  $w$ , group vertices between them into layers using BFS, direct edges between layers to form a DAG, and count paths from  $u$  to  $w$  using DAG path counting algorithm
- ▶  $\#$  paths through  $v = (\# \text{ paths } u \text{ to } v)(\# \text{ paths } v \text{ to } w)$

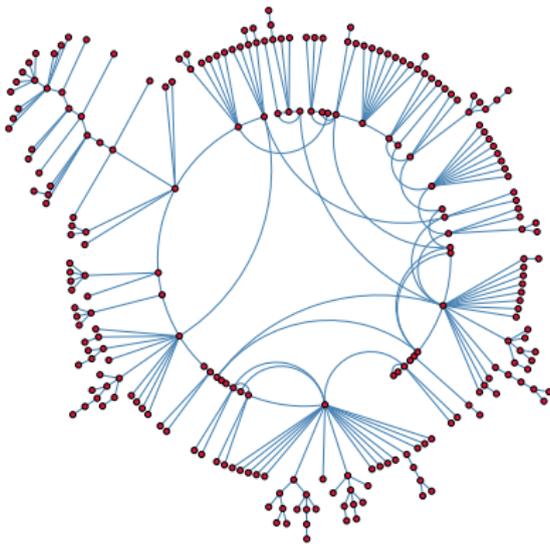
Faster algorithm:  $O(mn)$  [Brandes 2001]

Main idea: Rearrange calculation so it can be accumulated as BFS progresses rather than finishing all BFSs before combining results

**Cores**

# Structure in disease contact graphs

That HIV contact graph again:



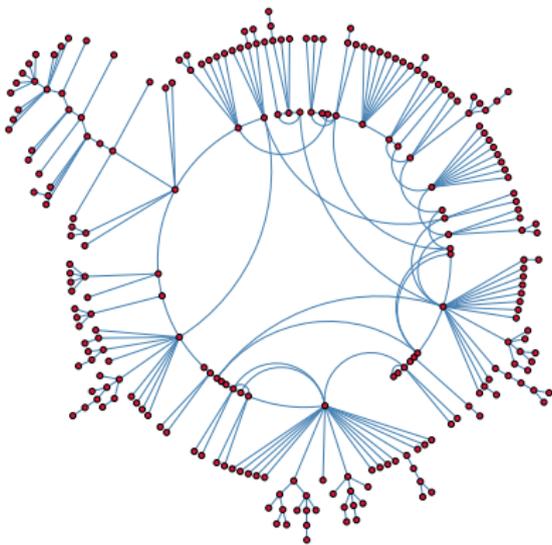
In the outer tree-like parts, lines of disease transmission are clear  
In the inner “core”, multiple lines of connection mean we still need  
more information to understand how the disease spread

How can we make this distinction automatically?

## Cores

For an integer parameter  $k$ , define the  $k$ -core of a graph to be the largest subgraph in which all vertices have degree  $\geq k$

Found by repeatedly deleting vertices of degree  $< k$  until none left



Center circle = 2-core = result of removing all degree-1 vertices

## Computing all cores simultaneously

Core number  $\text{core}(v)$  = the largest  $k$  so that  $v$  belongs to a  $k$ -core

```
def cores(G):
    core = new dictionary
    current_core = 0
    while G is non-empty:
        v = any minimum-degree vertex in G
        current_core = max(current_core, degree(v))
        core[v] = current_core
        delete v from G
    return core
```

Removing a minimum-degree vertex  $\Rightarrow$  whatever  $k$  we want, we remove vertices of degree  $< k$  before the first higher-degree vertex

When forced to remove higher-degree, we update `current_core`

The  $k$ -core is the set of all  $v$  with  $\text{core}(v) \geq k$ .

## How to make it efficient

Don't actually modify  $G$ ; just decorate vertices with  $N[v] = \#$  unprocessed neighbors of  $v$  (initially,  $\text{degree}(v)$ )

Maintain an array of collections of vertices  $D[i] =$  the set of unprocessed vertices whose number of unprocessed neighbors is  $i$

When processing a vertex  $v$ , subtract one from  $N[w]$  for each neighbor  $w$  and move  $w$  to the correct new collection  $D[N[w]]$

To find minimum degree unprocessed vertex, look in  $D[0], D[1], \dots$  until finding the first nonempty collection

Time to find  $v = O(\text{degree}(v))$ , sums to  $O(m)$  overall

[Matula and Beck 1983]

# Degeneracy

# Degeneracy, also called coloring number

Several equivalent definitions:

- ▶ Largest  $k$  for which a nonempty  $k$ -core exists
- ▶ Largest degree encountered by the  $k$ -core algorithm
- ▶ Smallest  $k$  so all subgraphs have a vertex of degree  $\leq k$
- ▶ Smallest  $k$  so that the vertices have an ordering in which each vertex has  $\leq k$  later neighbors (the ordering produced by the  $k$ -core algorithm)
- ▶ Smallest  $k$  so that the edges can be directed to form a DAG with out-degree  $\leq k$  (the DAG produced by directing edges from earlier to later in the ordering)

# Graph classes with small degeneracy

Many classes of graphs that automatically have low degeneracy.

- ▶  $G$  is an independent set  $\iff$  degeneracy = 0
- ▶  $G$  is a forest  $\iff$  degeneracy  $\leq 1$
- ▶ Barabasi–Albert graphs with parameter  $k \implies$  degeneracy  $\leq k$

# Real-world networks often have small degeneracy

In these tables,  $d$  = degeneracy,  $\Delta$  = max degree

Table I. Experimental Results for Mark Newman's Datasets

graph	$n$	$m$	$d$	$\Delta$	$\mu$	TTT-classic	TTT-lists	ELS-bare	ELS-array
zachary	34	78	4	17					
dolphins	62	159	4	12					
power	4,941	6,594	5	19					
polbooks	105	441	6	25					
adjnoun	112	425	6	49					
football	115	613	8	12					
lesmis	77	254	9	36					
celegensneural	297	2,148	9	134					
netscience	1,589	2,742	19	34					
internet	22,963	48,436	25	2,390	3				
condmat-2005	40,421	175,693	29	278	3				
polblogs	1,490	16,715	36	351	4				
astro-ph	16,706	121,251	56	360	1				

Table II. Experimental Results for BioGRID Datasets (PPI Networks)

graph	$n$	$m$	$d$	$\Delta$	$\mu$	TTT-classic	TTT-lists
mouse	1,455	1,636	6	111	1,523	0.01	<0.01
worm	3,518	3,518	10	523	5,652	0.14	0.01
plant	1,745	3,098	12	71	2,302	0.02	<0.01
fruitfly	7,282	24,894	12	176	21,995	0.62	0.03
human	9,527	31,182	12	308	23,863	1.06	0.03
fission-yeast	2,031	12,637	34	439	28,520	0.06	0.12
yeast	6,008	156,945	64	2557	738,613	1.74	11.37

Table III. Experimental Results for Pajek Datasets

graph	$n$	$m$	$d$	$\Delta$	$\mu$	TTT-classic	TTT-lists
foldoc	13,356	91,471	12	728	39,590	2.16	0.11
patents	240,547	560,943	24	212	482,538	*	0.56
eatRS	23,219	304,937	34	1,090	298,164	7.62	1.52
hep-th	27,240	341,923	37	2,411	446,823	12.56	3.40
days-all	13,308	148,035	73	2,265	2,173,772	5.83	62.86
ND-www	325,729	1,090,108	155	10,721	495,947	*	1.80

$d$  is generally smaller than  $\Delta$  and much smaller than  $n$

[Eppstein et al. 2013]

## Sneak peak at future topics

We will see that degeneracy is useful for:

Finding closely connected clusters (“cliques”)

Coloring graph vertices

(why degeneracy is also called coloring number)

## Morals of the story

There are many different ways of quantifying which nodes in a network are the important ones

We can take advantage of network structure (such as the small world property) to speed up their computation

Even when networks have some vertices with high degree (such as power law degree distributions) their degeneracy may be small

Degeneracy orderings can be constructed in linear time and are also useful for other problems such as graph coloring

## References

- Ulrik Brandes. A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25(2):163–177, 2001. doi:10.1080/0022250x.2001.9990249.
- David Eppstein and Joseph Y. Wang. Fast approximation of centrality. *J. Graph Algorithms & Applications*, 8(1):39–45, 2004. doi:10.7155/jgaa.00081.
- David Eppstein, Darren Strash, and Maarten Löffler. Listing all maximal cliques in large sparse real-world graphs in near-optimal time. *J. Experimental Algorithmics*, 18(3):3.1, 2013. doi:10.1145/2543629.
- David W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983. doi:10.1145/2402.322385.