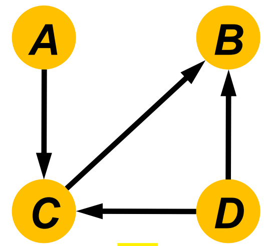
Network Algorithms

Michael Goodrich

Some slides adapted from:
Networked Life (NETS) 112, Univ. of Penn., 2018, Prof. Michael Kearns
Determining the Diameter of Small World Networks, Frank W. Takes & Walter A. Kosters, Leiden University, The Netherlands
Structure and models of real-world graphs and networks, Jure Leskovec, Carnegie Mellon University
Complex (Biological) Networks, by Elhanan Borenstein, Roded Sharan, and Tomer Shlomi

Computational Representation of Networks



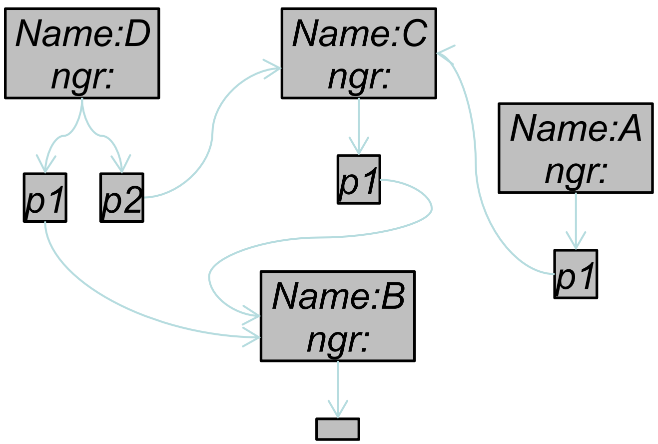
List or hash table of edges: (ordered) pairs of nodes

{ (A,C) , (C,B) , (D,B) , (D,C) }

Adjacency Matrix

	A	B	C	D
A	0	0	1	0
B	0	0	0	0
C	0	1	0	0
D	0	1	1	0

Adjacency List



- **Which is the most useful representation?**
- **Should you use them in combination?**

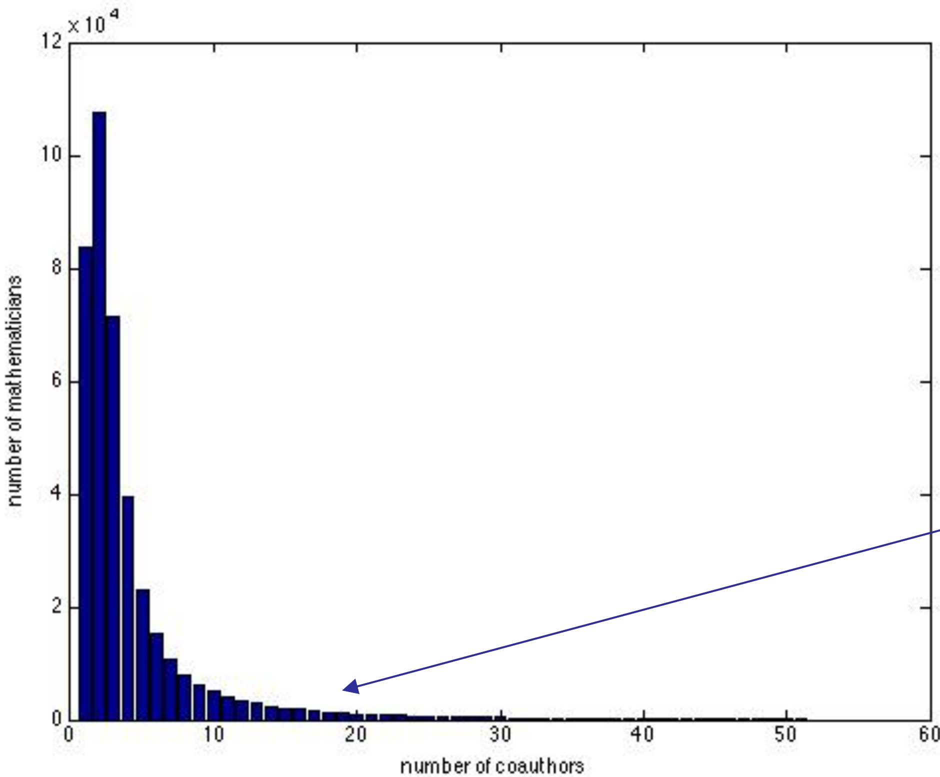
Network Structures

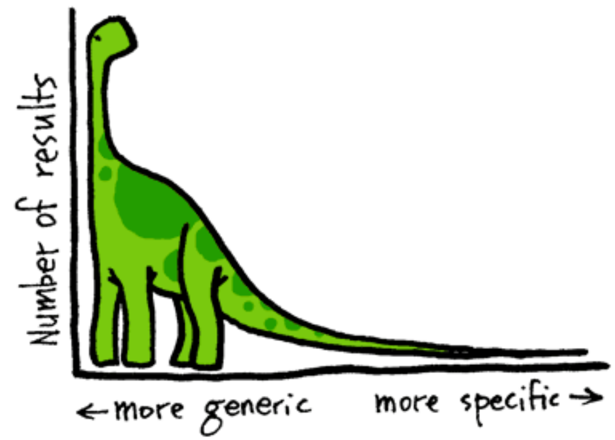
- Network structures characterize how networks “look”:
  - Large or small diameter?
  - Number of edges: sparse or dense?
  - Degree distributions: heavy/long tail with a power law?
  - Clustering coefficient: high or low?
- These are empirical phenomena
- How do you compute them?

Image from *https://matrix.berkeley.edu/research/social-networks-history*

Degree Distribution

- x axis: number of neighbors (degree)
- y axis: number of vertices with that degree





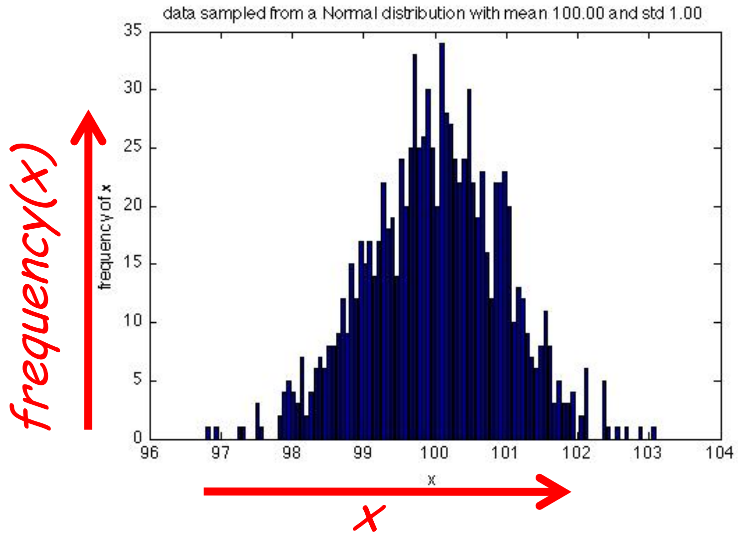
A long tail
(also known as a "heavy tail")

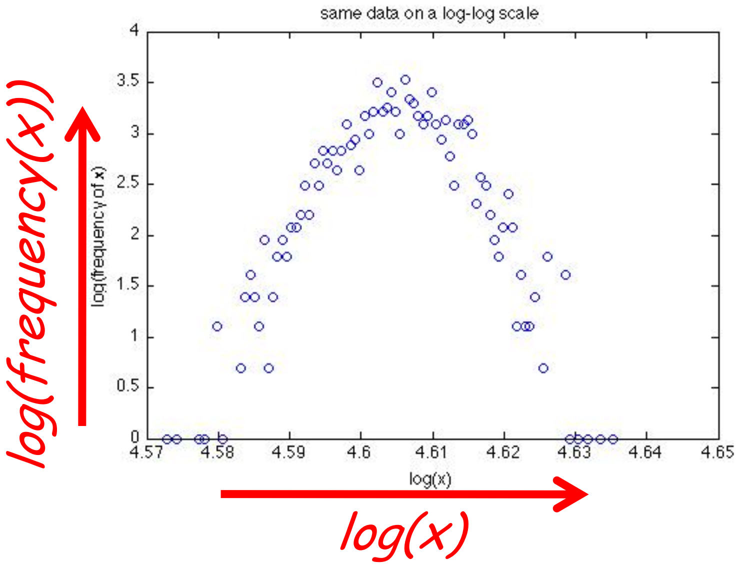
Degree Distribution Algorithm

1. Compute the degree, deg(v), of each vertex, v.
   – If G is represented as an adjacency list, count the number of elements in v's list.
2. Create a histogram count array, H, of size n, and initialize each $H[i] = 0$.
3. For each vertex, v, increment $H[\deg(v)]$.
4. Plot the values of H from 0 to n-1 on a regular and log-log scale
5. If the values on the log-log plot form a straight line, determine its slope to find the exponent of the power law degree distribution

Example 1

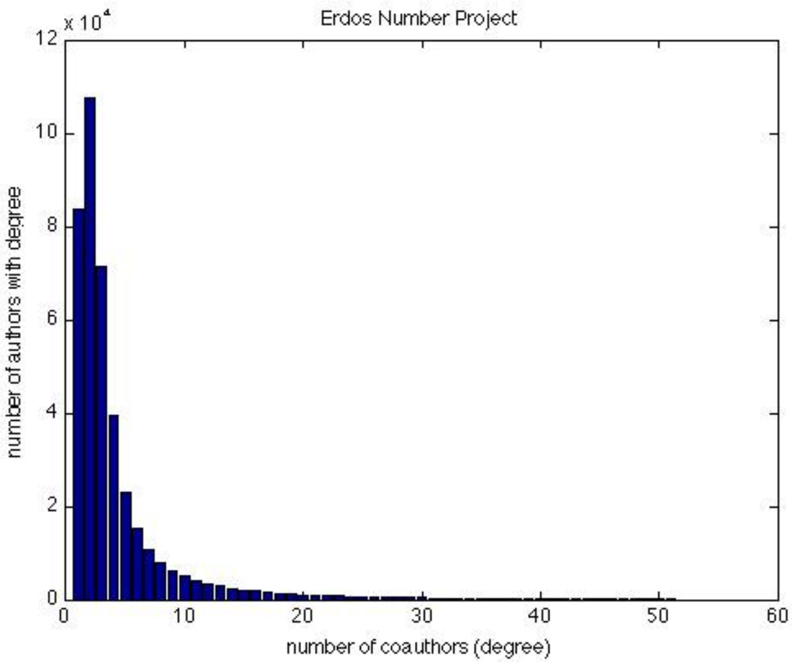
- Degree distribution without a long/heavy tail .
- Does not exhibit a power law.

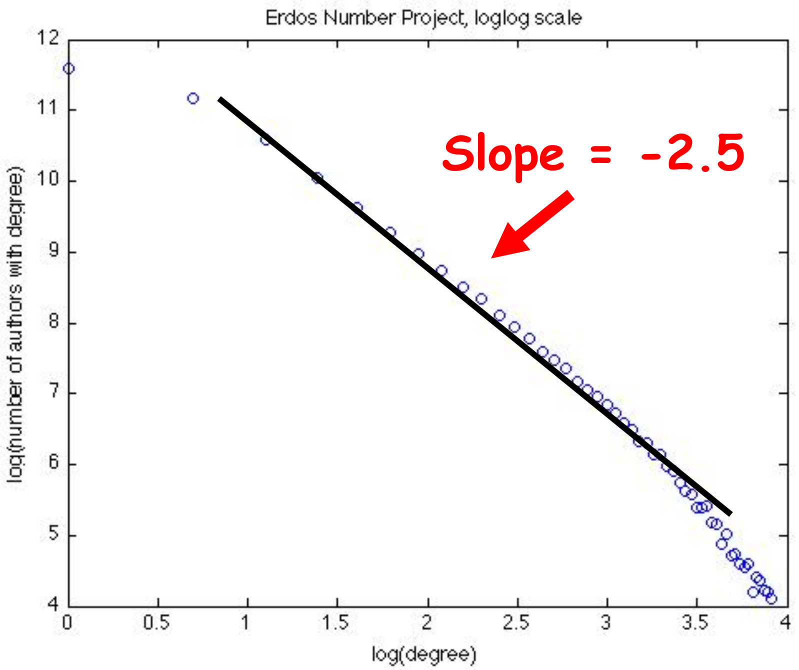




Example 2

- Degree distribution with a long/heavy tail .
- Does exhibit a power law, with exponent -2.5.





Distance

- The **distance** between two vertices is the length of the shortest path connecting them.
  - This assumes the network has only a single connected component
  - If two vertices are in different components, their distance is infinite

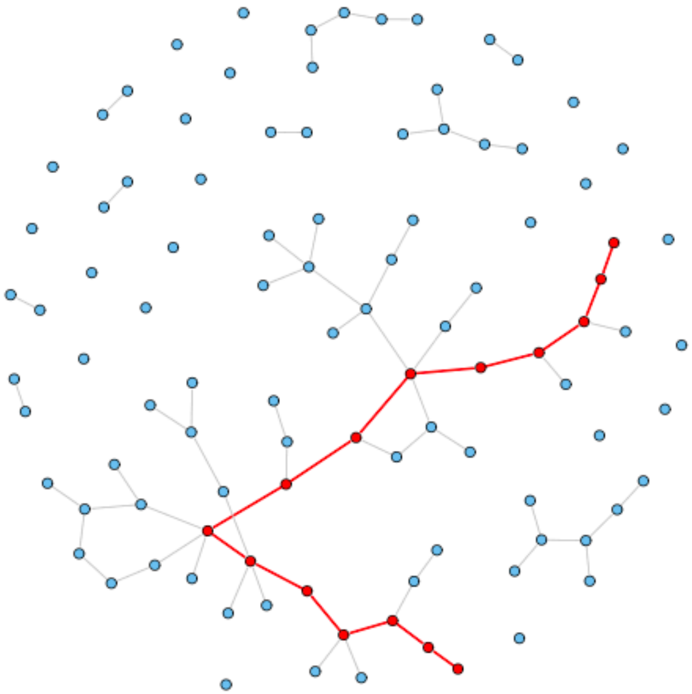
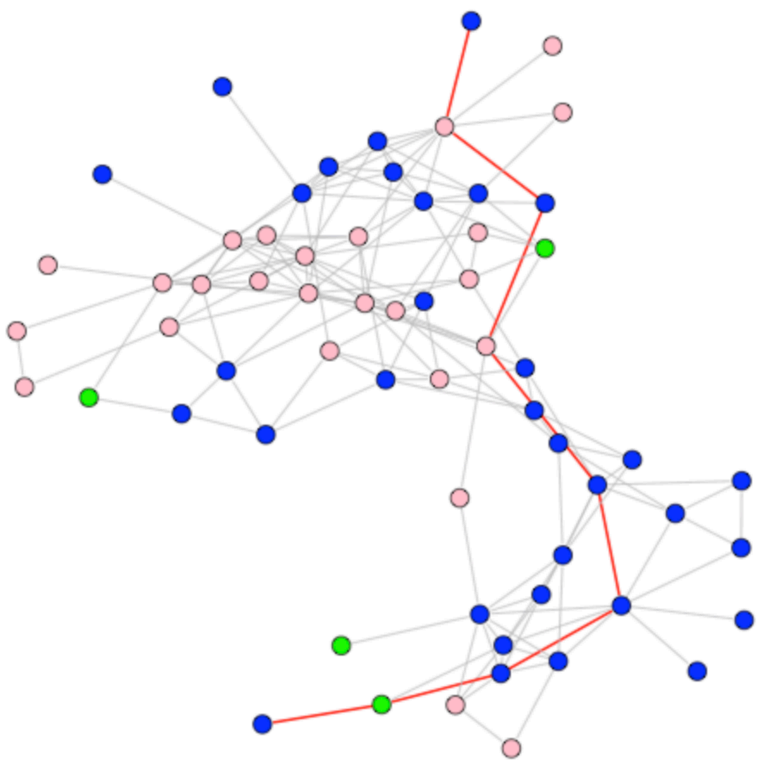


Image from https://www.sci.unich.it/~francesc/teaching/network/geodesic.html

Diameter

- The **diameter** of a network is the maximum distance between a pair of vertices in the network
  - It measures how near or far typical individuals are from each other



The dolphin network with the diameter (the longest shortest path) highlighted in red. The diameter is 8 edges long.

[From https://users.dimi.uniud.it/~massimo.franceschet/bottlenose/bottlenose.html](https://users.dimi.uniud.it/~massimo.franceschet/bottlenose/bottlenose.html)

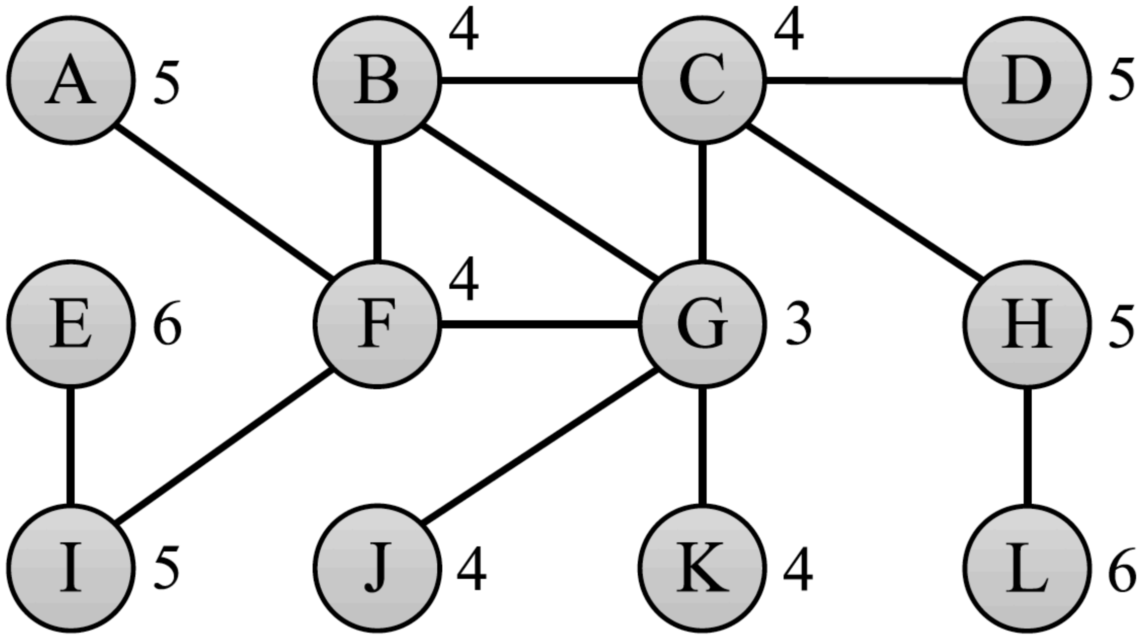
Definitions

- Consider a connected undirected graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges
- **Distance** $d(v, w)$: length of shortest path between nodes $v, w \in V$
- **Diameter** $D(G)$: maximal distance (longest shortest path length) over all node pairs: $\max_{v,w \in V} d(v, w)$
- **Eccentricity** $e(v)$: length of a longest shortest path from v: $e(v) = \max_{w \in V} d(v, w)$
- **Diameter** $D(G)$ (alternative definition): maximal eccentricity over all nodes: $\max_{v \in V} e(v)$
- Eccentricity distribution: (relative) frequency $f(x)$ of each eccentricity value x

$$f(x) = \frac{|\{u \in V \mid e(u) = x\}|}{n}$$

Example

- A graph with diameter 6
- Numbers next to nodes denote eccentricity values



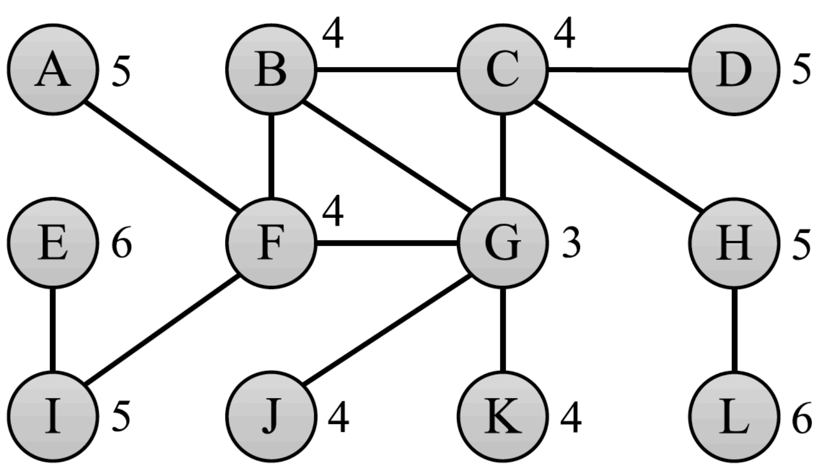
Naïve Algorithm

- **Diameter** is equal to the largest value returned by an All Pairs Shortest Path (APSP) algorithm
- Brute-force: for each vertex v, execute a Breadth First Search (BFS) from v in O(m) time to find v’s **eccentricity**. Return the largest value found.
- Time complexity O(nm)
- Problematic if n = 8 million and m = 1 billion.
  - If one BFS takes 6 seconds on a 3.4GHz machine, this brute-force algorithm takes 1.5 years to compute the diameter . . .

Heuristic Idea 1

- If we can find one of the nodes in a diameter pair, we can compute the diameter with one more BFS.

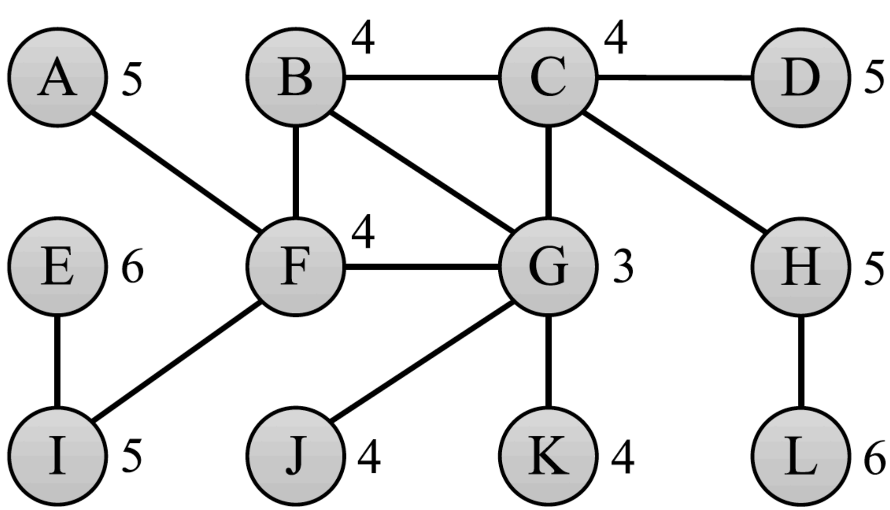
1. Perform a BFS from a random sample of nodes, recording nodes with maximum found distance, d.
2. Perform a BFS from all the far nodes (if small) or a random sample of this set (if large).



Heuristic Idea 2

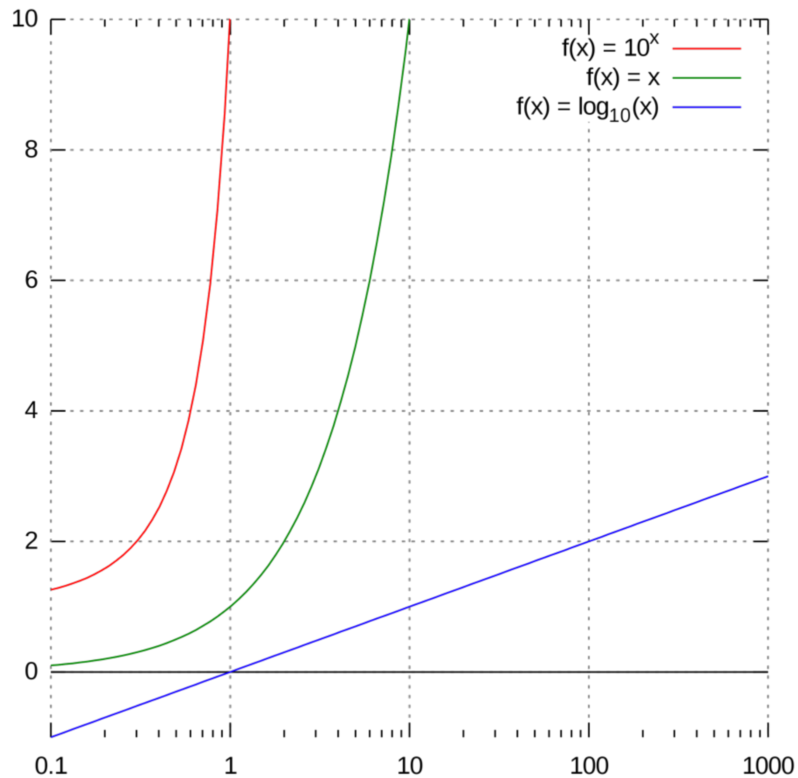
1. Let r be a random vertex and set $D_{max} = 0$.
2. Perform a BFS from r.
3. Select the farthest node, w, in this BFS.

- If the distance from r to w is larger than D_{max}, set D_{max} to this distance, let r = w, and repeat the above two steps.



Plot Results as a Function of n

- If the networks exhibit the **small world** phenomenon, then diameters are small.
- So plot diameters as a function of n on a lin-log scale:



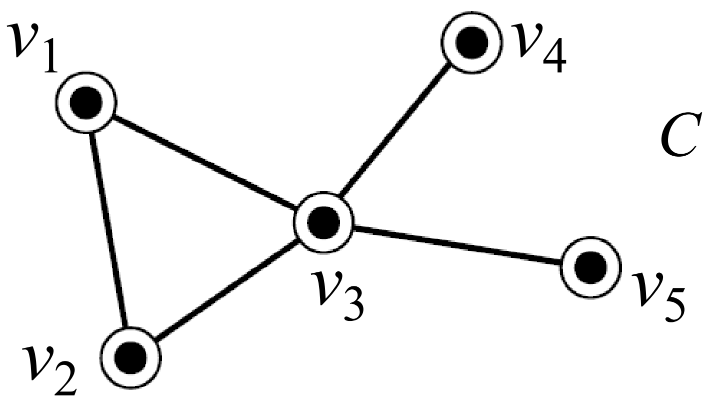
The log n function looks like a straight line

LinLogScale.png: davidfg derivative work: Autopilot [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0/)]

Clustering Coefficient

- "friend of a friend is a friend"
- If a connects to b, and b to c,
  then with high probability
  a connects to c.
- Clustering coefficient C:

  C = 3*number of triangles / number of 2-edge paths



$C = 3*(1)/(1+1+6+0+0) = 3/8 = 0.375$

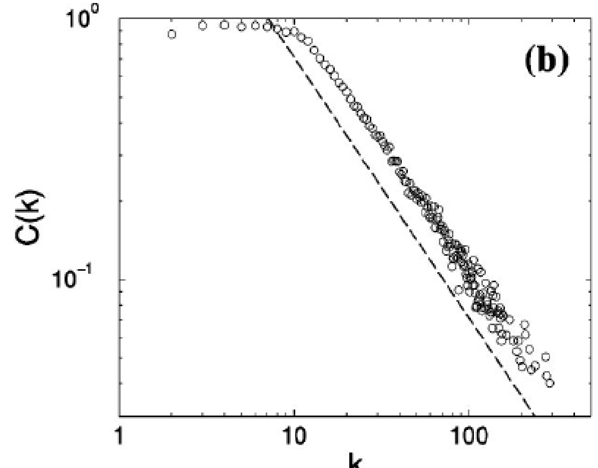
Clustering Coefficient (2)

- Clustering coefficient might have a power law:

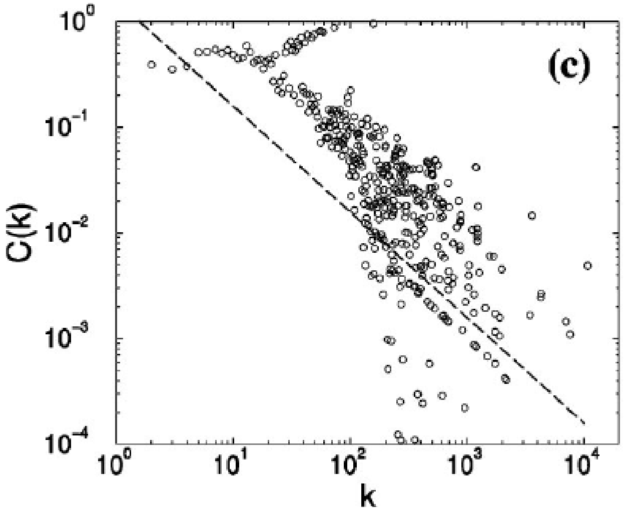
$$C(k) \sim k^{-1}$$

- It is speculated that in real networks:

$$C = O(1) \text{ as } n \to \infty$$



Synonyms network

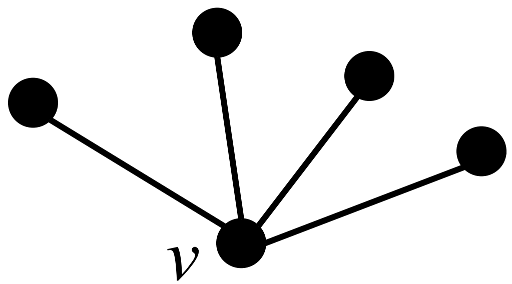


World Wide Web

Clustering Coefficient Algorithm

- Clustering coefficient C:

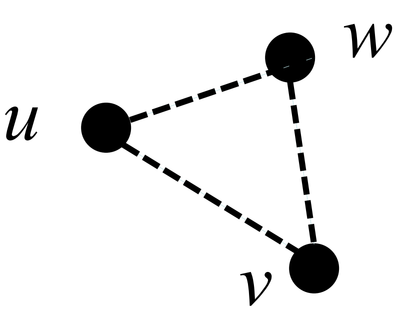
  C = 3*number of triangles / number of 2-edge paths
- Computing the **denominator** is **easy**:
  - For each vertex v, let deg(v) denote its degree.
  - The number of paths of length 2 with v in the middle is deg(v) choose 2 = deg(v)(deg(v)-1)/2.
  - So, to get the denominator for C, sum up deg(v)(deg(v)-1)/2 for all vertices, v, in G.



Number of 2-edge paths with v in the middle is 4(3)/2 = 6.

Counting Triangles

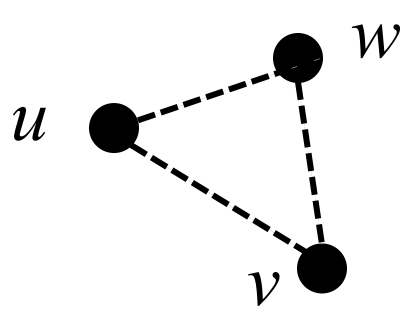
- To get the numerator for C, we need to count the number of triangles in the graph, G.
- Naïve algorithm:
  - For every triple, u, v, w in G, see if they form a triangle. If so, add 1 to a running count.
  - Running time is O(n^4) if G is represented with an adjacency list.



This is bad.

Counting Triangles: Slight Improvement

- Put every edge, (v,w), into a hash table, T, so we can do a lookup to see if an edge exists in O(1) expected time, i.e., with a get((v,w)).
- Slightly better naïve algorithm:
  - For every triple, u, v, w in G, see if they form a triangle. If so, add 1 to a running count.
  - Running time is $O(n^3)$ expected if edges in G are stored in a hash table.

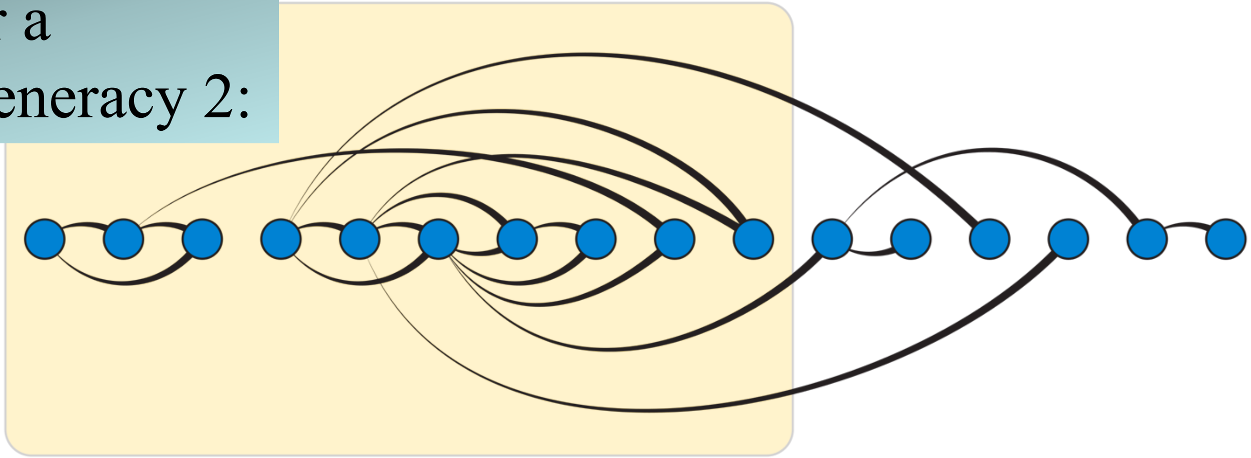


This is still bad.

Graph Degeneracy

- The **degeneracy** of a graph is the smallest value of d for which every subgraph has a vertex of degree at most d.
- If a graph has degeneracy d, then there exists an ordering of the vertices of G in which each vertex has at most d neighbors that are earlier in the ordering.

An ordering for a graph with degeneracy 2:



Public domain image by David Eppstein

Real-World Graphs

- Real-world graphs tend to have small degeneracy, d.

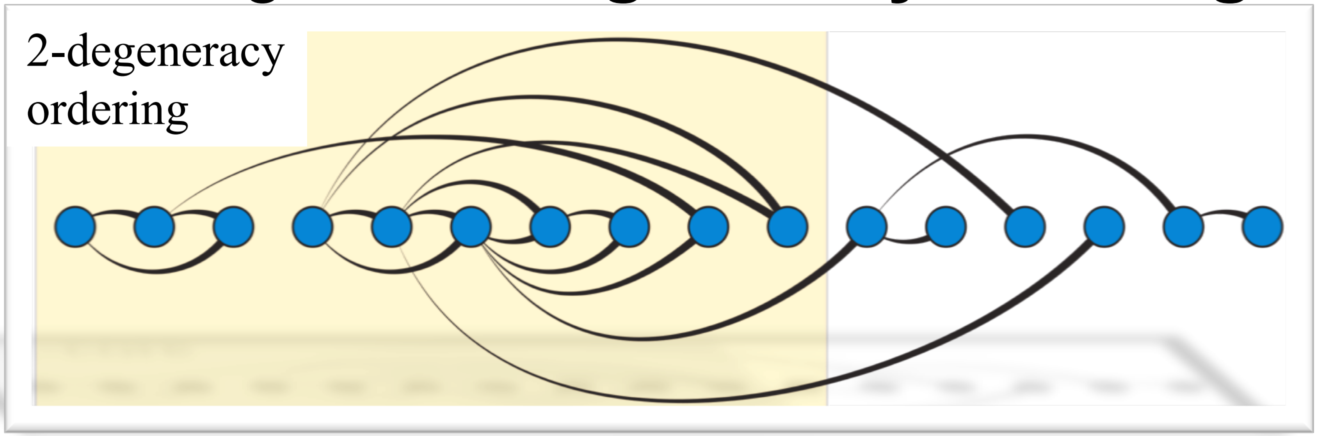
graph	n	m	d
zachary [48]	34	78	4
dolphins [35]	62	159	4
power [47]	4,941	6,594	5
polbooks [28]	105	441	6
adjnoun [29]	112	425	6
football [15]	115	613	8
lesmis [25]	77	254	9
celegensneural [47]	297	1,248	9
netscience [39]	1,589	2,742	19
internet [40]	22,963	48,421	25
condmat-2005 [38]	40,421	175,693	29
polblogs [4]	1,490	16,715	36
astro-ph [38]	16,706	121,251	56

graph	n	m	d
roadNet-CA [34]	1,965,206	2,766,607	3
roadNet-PA [34]	1,088,092	1,541,898	3
roadNet-TX [34]	1,379,917	1,921,660	3
amazon0601 [30]	403,394	2,443,408	10
email-EuAll [31]	265,214	364,481	37
email-Enron [24]	36,692	183,831	43
web-Google [2]	875,713	4,322,051	44
soc-wiki-Vote [33]	7,115	100,762	53
soc-slashdot0902 [34]	82,168	504,230	55
cit-Patents [18]	3,774,768	16,518,947	64
soc-Epinions1 [42]	75,888	405,740	67
soc-wiki-Talk [33]	2,394,385	4,659,565	131
web-berkstan [34]	685,231	6,649,470	201

Data from "Listing All Maximal Cliques in Large Sparse Real-World Graphs," by David Eppstein and Darren Strash

Degeneracy Ordering Algorithm

- Degeneracy can be computed by a simple **greedy algorithm**:
  - Repeatedly find and remove the vertex of smallest degree, adding it to the end of the list.
  - The degeneracy is then the highest degree, d, of any vertex at the moment it is removed.
  - The ordering is a **d-degeneracy ordering**.



Public domain image by David Eppstein

Linear-time Implementation

1. Initialize an output list, L, to be empty.
2. Compute a number, d_v, for each vertex v in G, which is the number of neighbors of v that are not already in L. Initially, d_v *is* just the degrees of v.
3. Initialize an array D such that $D[i]$ contains a list of the vertices v that are not already in L for which $d_v = i$.
4. Let N_v be a list of the neighbors of v that come before v in L. Initially, N_v is empty for every vertex v.
5. Initialize k to 0.
6. Repeat n times:
   – Let i be the smallest index such that $D[i]$ is nonempty.
   – Set k to max(k, i).
   – Select a vertex v from $D[i]$. Add v to the beginning of L and remove it from $D[i]$. Mark v as being in L (e.g., using a hash table, H_L).
   – For each neighbor w of v not already in L (you can check this using H_L):
     - Subtract one from d_w
     - Move w to the cell of D corresponding to the new value of d_w, i.e., $D[d_w]$
     - Add w to N_v

Triangle Counting Algorithm

- Compute a d-degeneracy ordering of the vertices, e.g., using the algorithm of the previous slide.
- Process the vertices according to this ordering, L:

  For each vertex, v:

  For each pair of vertices, u and w, adjacent to v and earlier in the ordering, i.e., u and v are in the list N_v from the degeneracy algorithm:

  If (u,w) is an edge in the graph, then add one to the triangle count.
- Running time is $O(d^2n) = O(dm)$ expected, assuming edges are stored in a hash table.