

# **Stanford CS224W:** **Generative Models for Graphs**

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

<http://cs224w.stanford.edu>



# Motivation for Graph Generation

- So far, we have been learning from graphs
  - We assume the graphs are given



Image credit: [Medium](#)

Social Networks

Economic Networks

Communication Networks

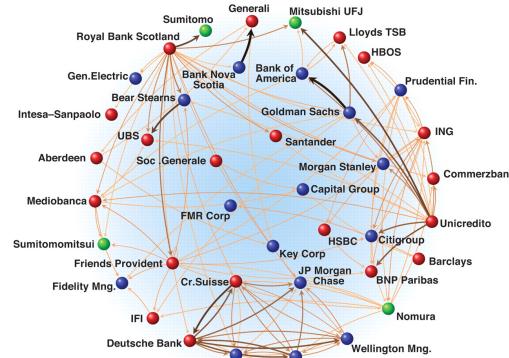


Image credit: [Science](#)



Image credit: [Lumen Learning](#)

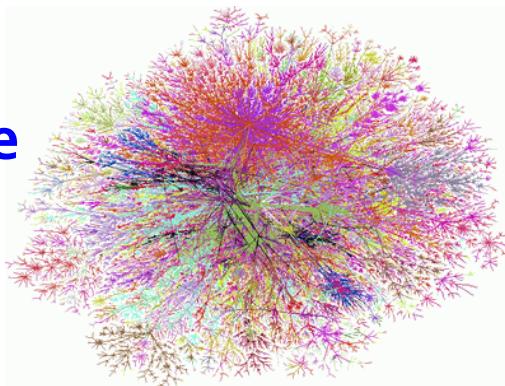
- But how are these graphs generated?

# The Problem: Graph Generation

- We want to generate realistic graphs, using **graph generative models**

Graph  
Generative  
Model

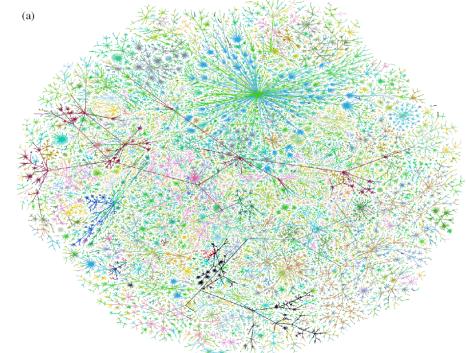
Generate  
→



Synthetic graph

which is  
similar to

~



Real graph

# Why Do We Study Graph Generation

- **Insights** – We can understand the formulation of graphs
- **Predictions** – We can predict how will the graph further evolve
- **Simulations** – We can use the same process to generate novel graph instances
- **Anomaly detection** - We can decide if a graph is normal / abnormal

# Road Map of Graph Generation

- **Step 1: Properties of real-world graphs**
  - A successful graph generative model should fit these properties
- **Step 2: Traditional graph generative models**
  - Each come with different assumptions on the graph formulation process
- **Step 3: Deep graph generative models**
  - Learn the graph formulation process from the data
  - Cover in the next lecture

# Stanford CS224W: Properties of Real-world Graphs

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

<http://cs224w.stanford.edu>



# Plan: Key Network Properties

We will characterize graphs by:

Degree distribution:  $P(k)$

Clustering coefficient:  $C$

Connected components:  $S$

Path length:  $h$

We have introduced these notions in Lecture 1&2. Here we give a quick recap.

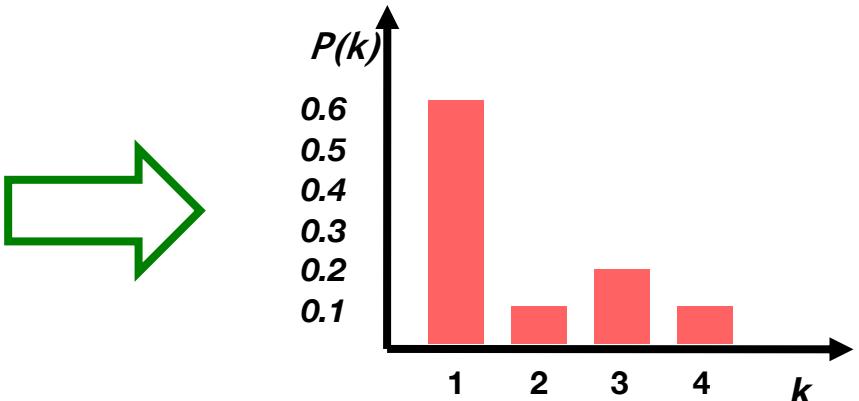
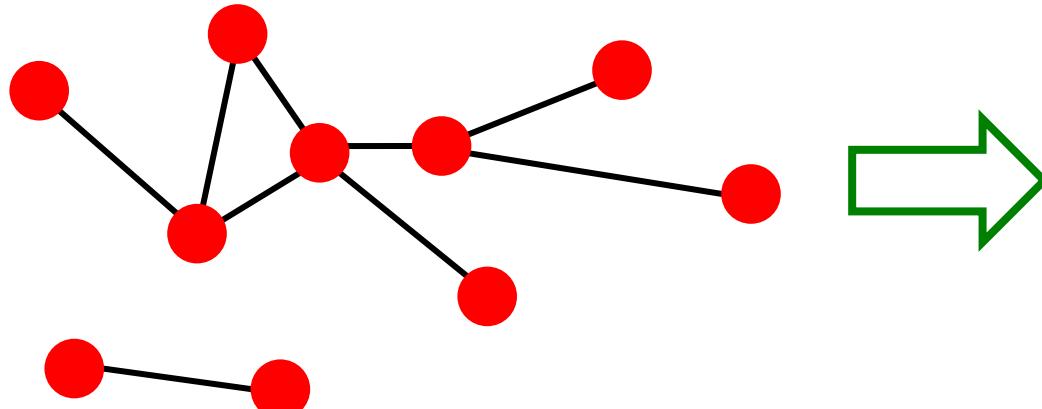
# (1) Degree Distribution

- **Degree distribution  $P(k)$ :** Probability that a randomly chosen node has degree  $k$

$$N_k = \# \text{ nodes with degree } k$$

- Normalized histogram:

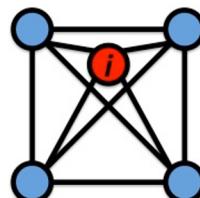
$$P(k) = N_k / N \rightarrow \text{plot}$$



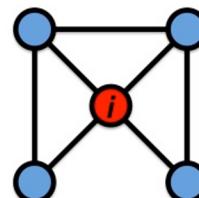
# (2) Clustering Coefficient

## ■ Clustering coefficient:

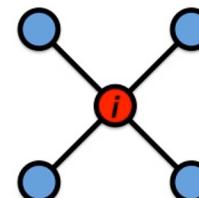
- How connected are  $i$ 's neighbors to each other?
- Node  $i$  with degree  $k_i$
- $C_i = \frac{2e_i}{k_i(k_i - 1)}$  where  $e_i$  is the number of edges between the neighbors of node  $i$



$$C_i = 1$$



$$C_i = 1/2$$



$$C_i = 0$$

$$C_i \in [0,1]$$

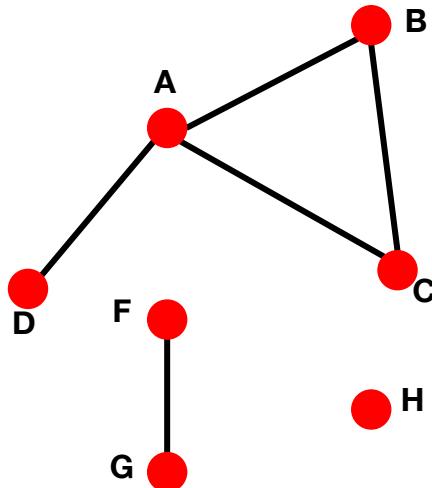
## ■ Graph clustering coefficient:

- Take average over all the nodes

$$C = \frac{1}{N} \sum_i^N C_i$$

# (3) Connectivity

- **Size of the largest connected component**
  - Largest set where any two vertices can be joined by a path
- **Largest component = Giant component**



## How to find connected components:

- Start from random node and perform Breadth First Search (BFS)
- Label the nodes that BFS visits
- If all nodes are visited, the network is connected
- Otherwise find an unvisited node and repeat BFS

# (4) Path Length

- **Diameter:** The maximum (shortest path) distance between any pair of nodes in a graph
- **Average path length** for a connected graph or a strongly connected directed graph

$$\bar{h} = \frac{1}{2E_{\max}} \sum_{i, j \neq i} h_{ij}$$

- $h_{ij}$  is the distance from node  $i$  to node  $j$
- $E_{\max}$  is the max number of edges (total number of node pairs) =  $n(n-1)/2$

- Many times we compute the average only over the connected pairs of nodes (that is, we ignore “infinite” length paths)

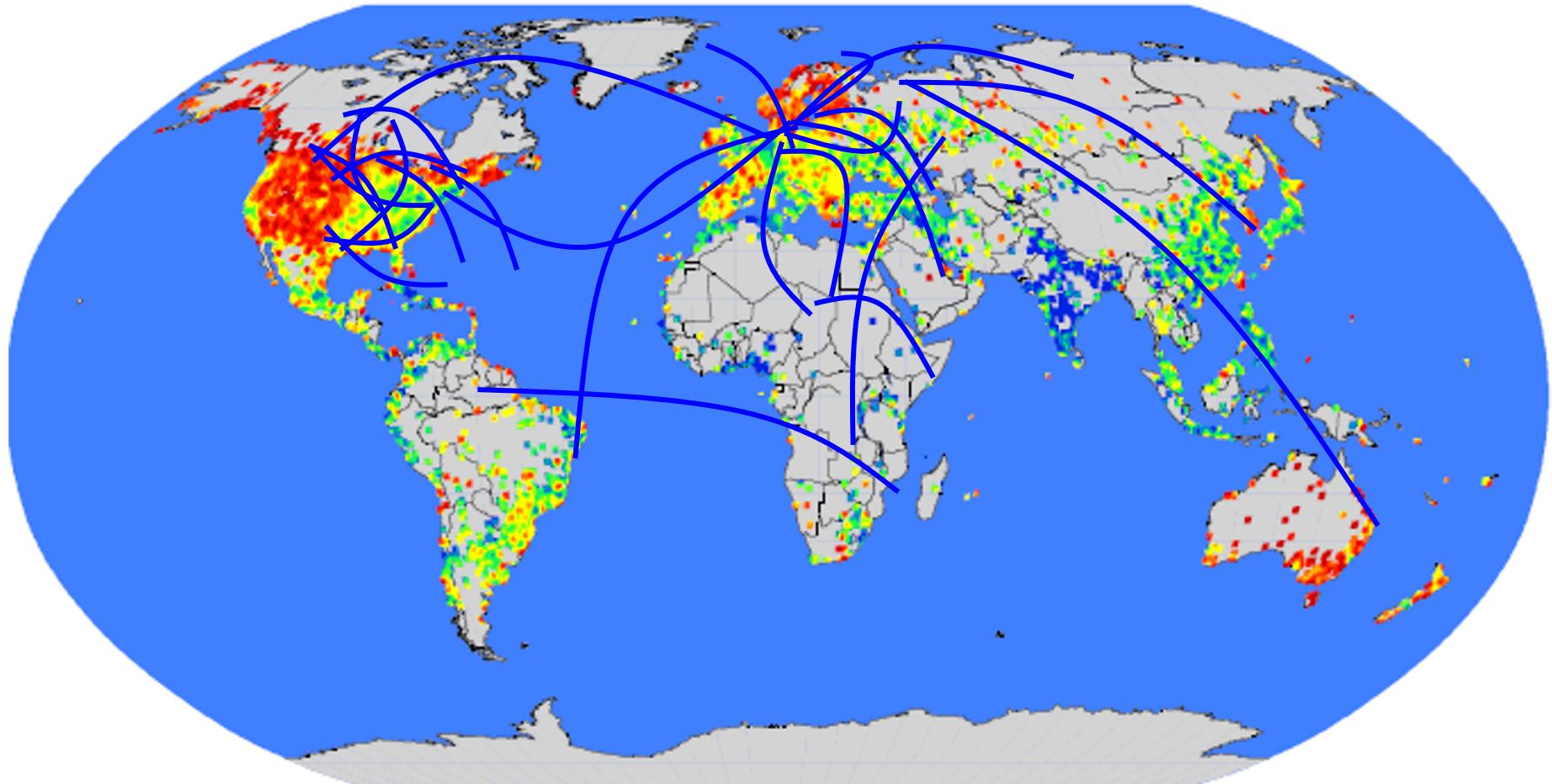
# Case Study: MSN Graph



- **MSN Messenger:**
- **1 month of activity**

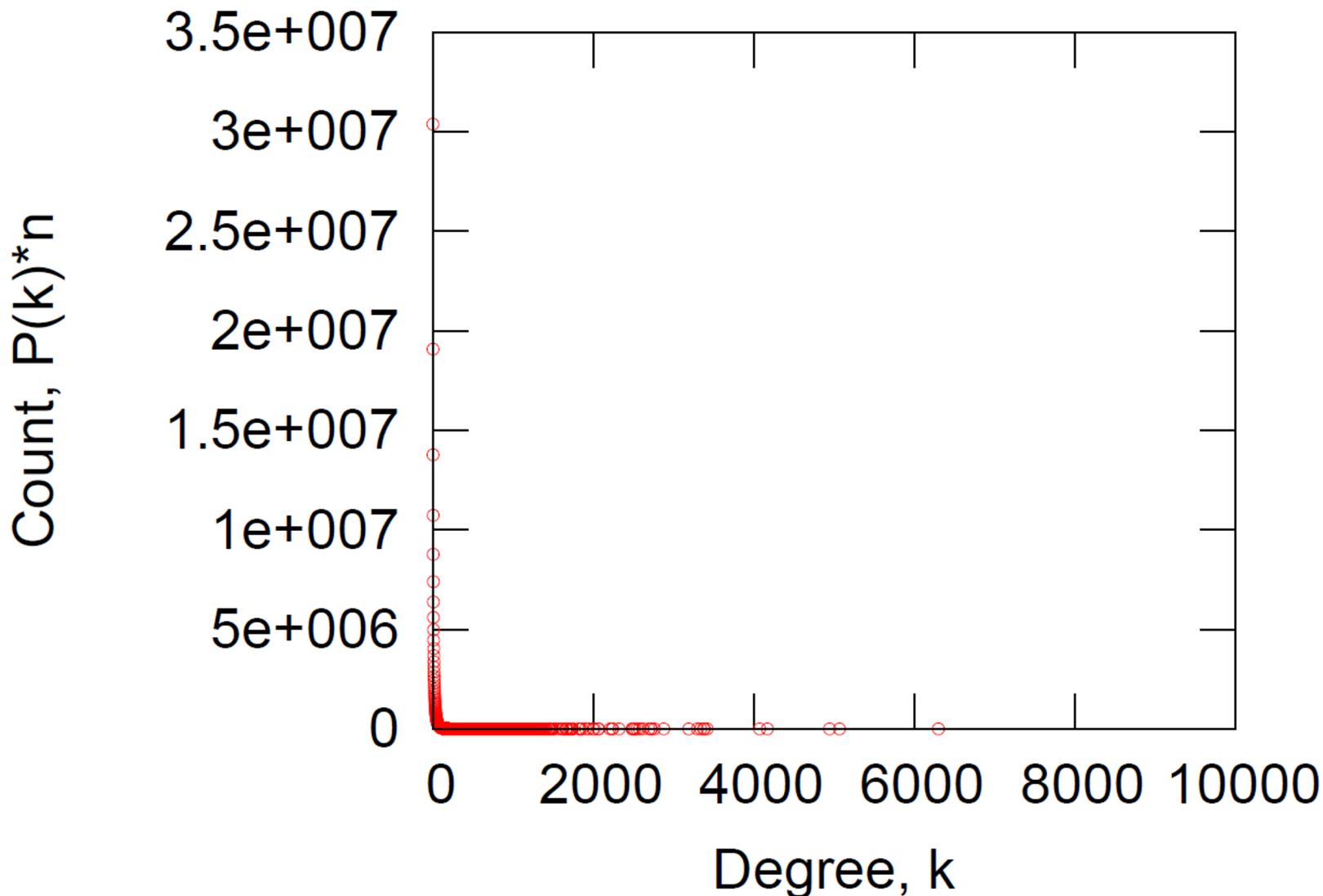
- 245 million users logged in
- 180 million users engaged in conversations
- More than 30 billion conversations
- More than 255 billion exchanged messages

# Communication Network

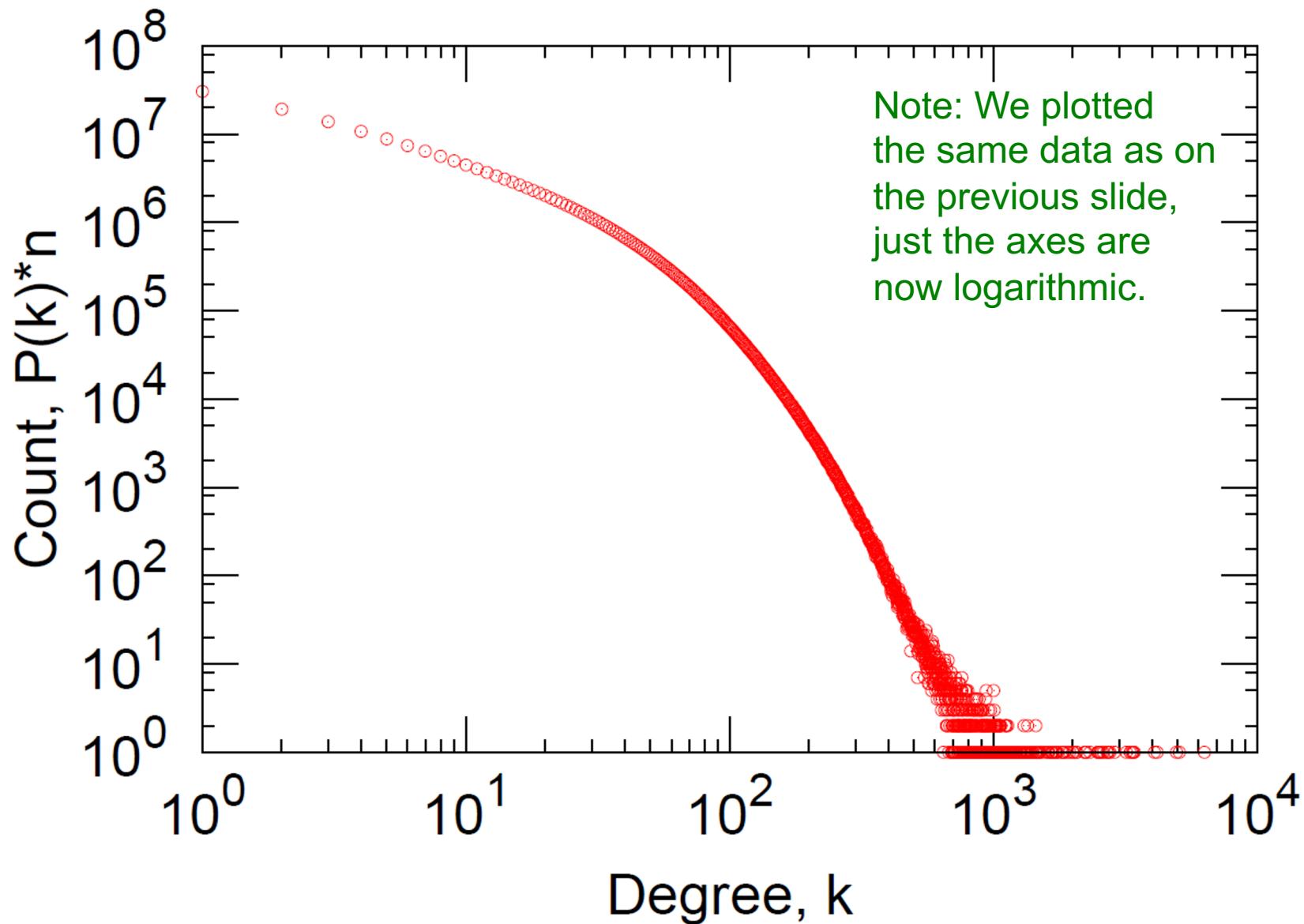


**Network:** 180M people, 1.3B edges

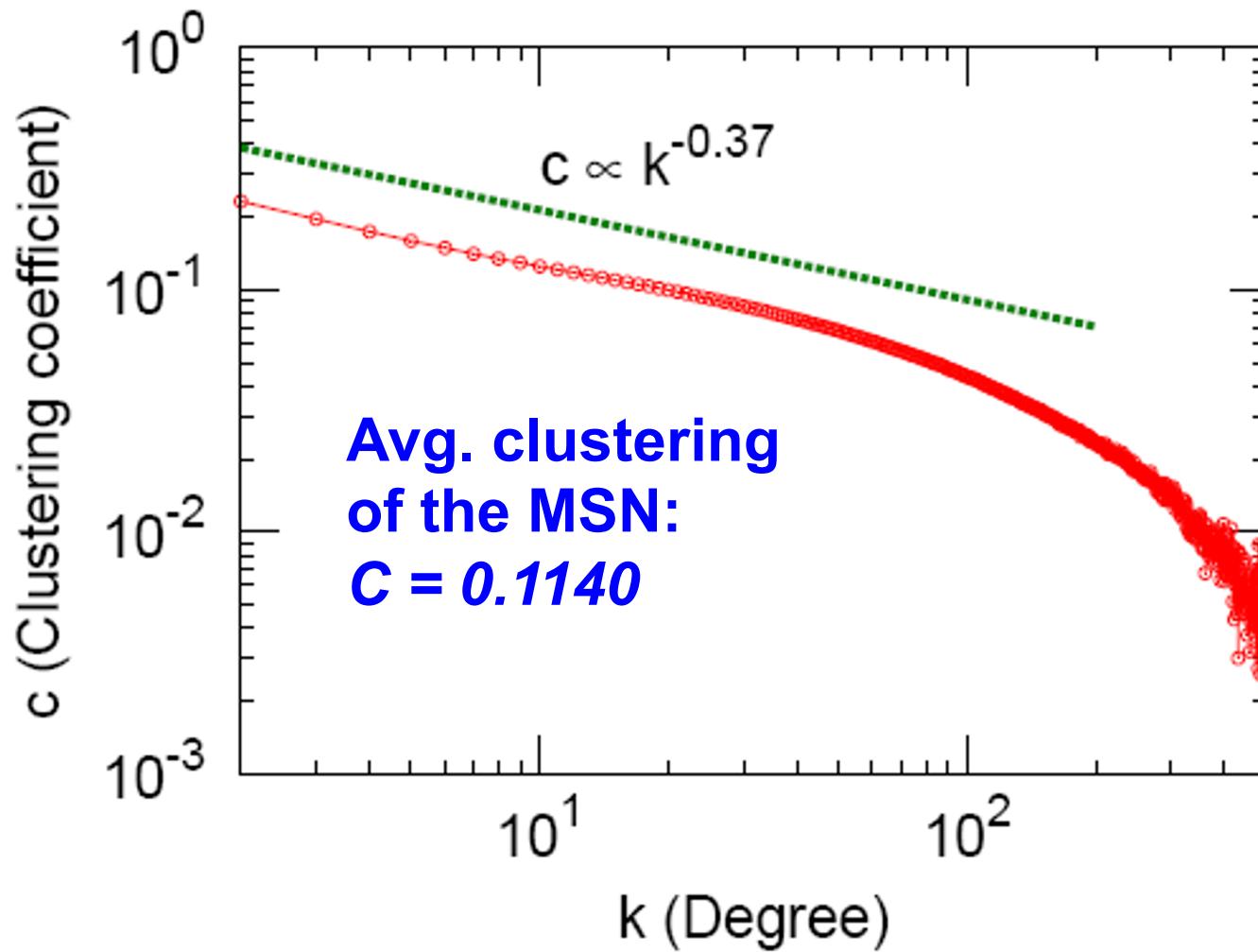
# MSN: (1) Degree Distribution



# MSN: Log-Log Degree Distribution

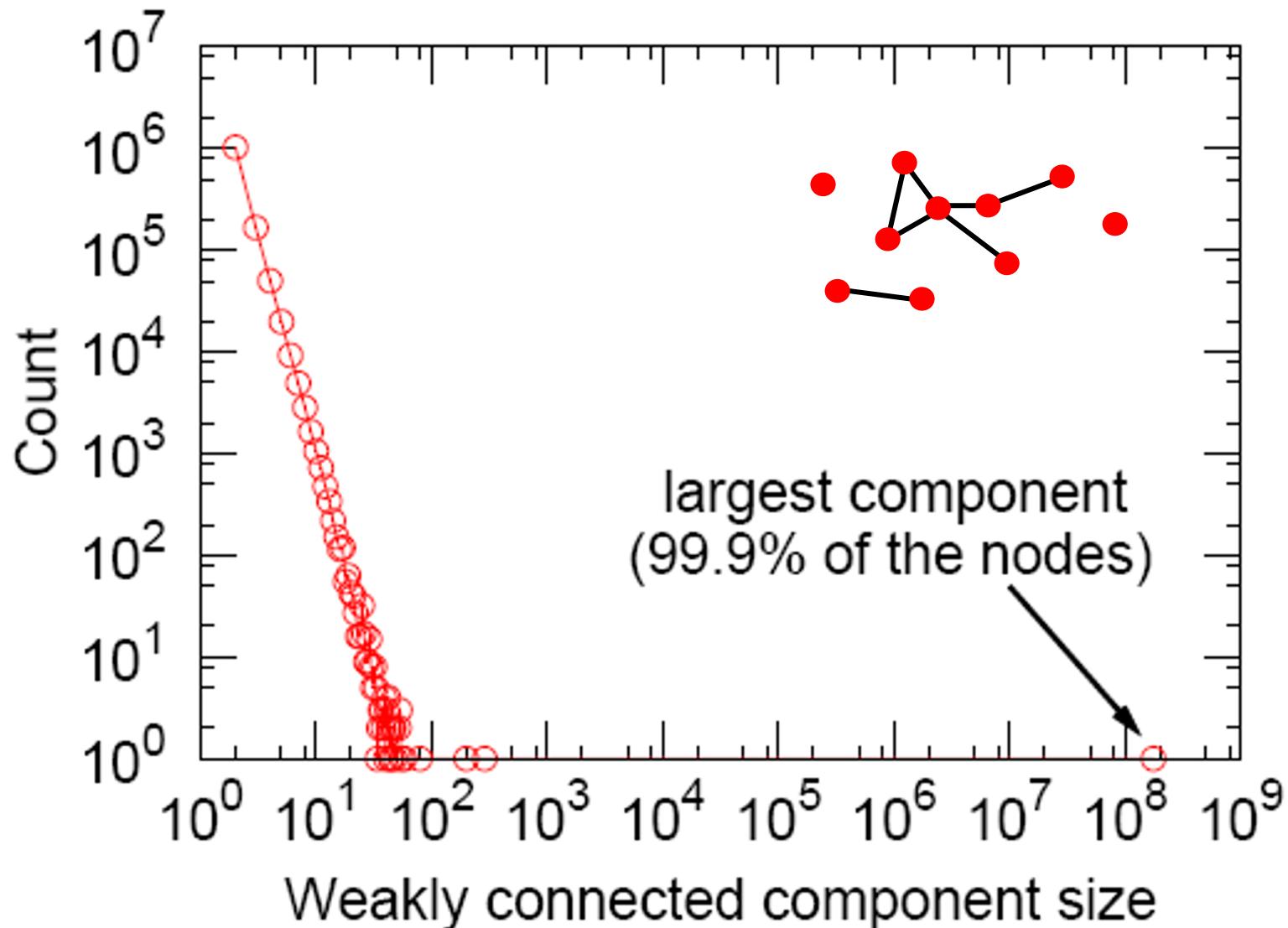


# MSN: (2) Clustering

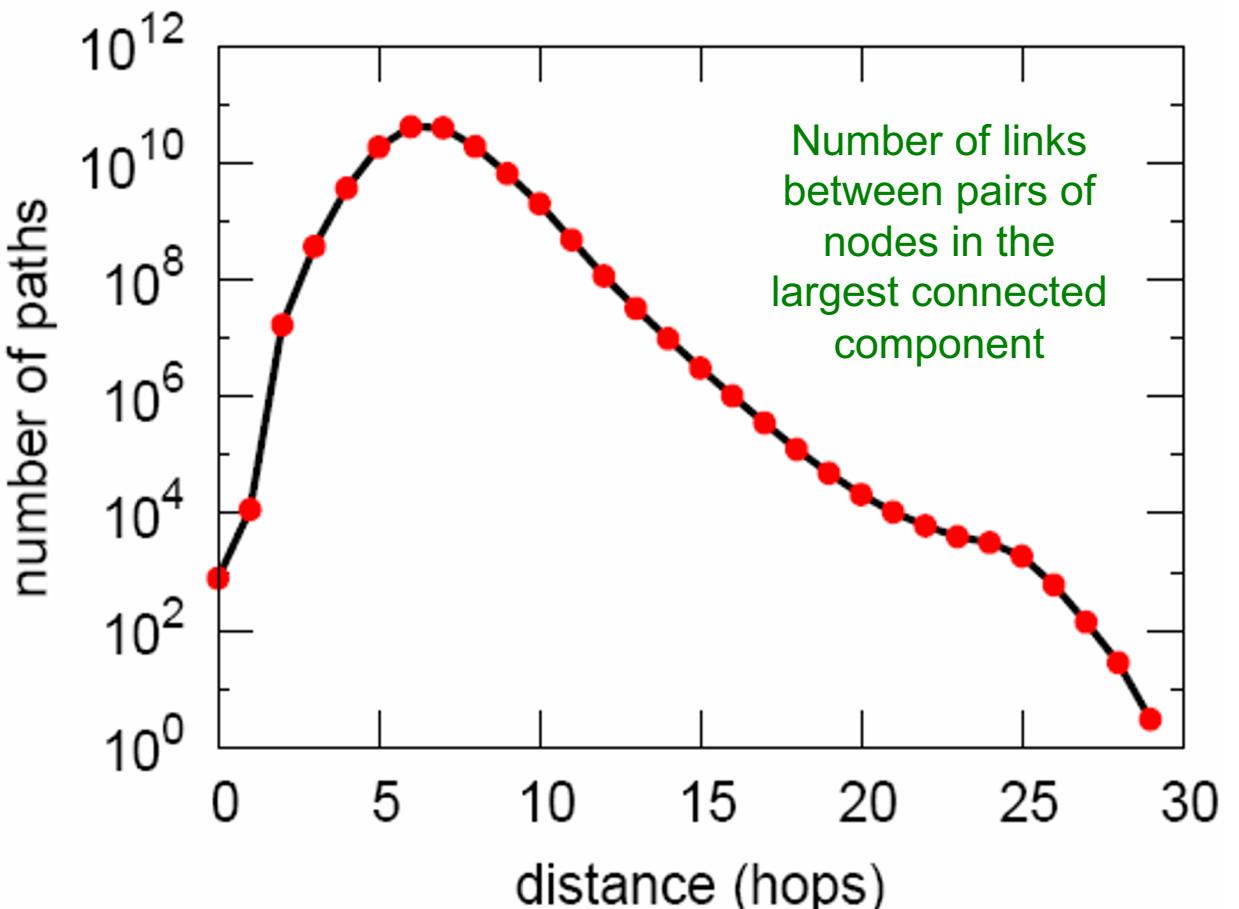


$C_k$ : average  $C_i$  of nodes  $i$  of degree  $k$ :  $C_k = \frac{1}{N_k} \sum_{i:k_i=k} C_i$

# MSN: (3) Connected Components



# MSN: (4) Path Length



Avg. path length 6.6

90% of the nodes can be reached in < 8 hops

Steps	#Nodes
0	1
1	10
2	78
3	3,96
4	8,648
5	3,299,252
6	28,395,849
7	79,059,497
8	52,995,778
9	10,321,008
10	1,955,007
11	518,410
12	149,945
13	44,616
14	13,740
15	4,476
16	1,542
17	536
18	167
19	71
20	29
21	16
22	10
23	3
24	2
25	3

# MSN: Key Network Properties

**Degree distribution:**

*Heavily skewed;  
avg. degree = 14.4*

**Clustering coefficient:**

*0.11*

**Connectivity:**

*giant component*

**Path length:**

*6.6*

**Are these values “expected”?**

**Are they “surprising”?**

**To answer this, we need a model!**

# **Stanford CS224W: Erdös-Renyi Random Graphs**

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

<http://cs224w.stanford.edu>



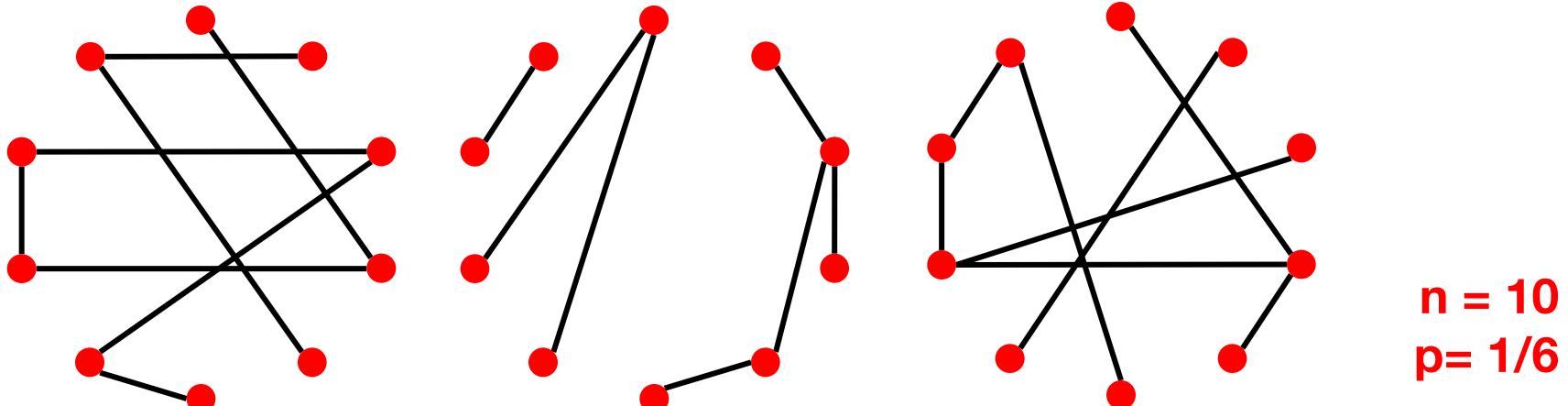
# Simplest Model of Graphs

- **Erdös-Renyi Random Graphs** [Erdös-Renyi, '60]
- **Two variants:**
  - $G_{np}$ : undirected graph on  $n$  nodes where each edge  $(u,v)$  appears i.i.d. with probability  $p$
  - $G_{nm}$ : undirected graph with  $n$  nodes, and  $m$  edges picked uniformly at random

What kind of networks do such models produce?

# Random Graph Model $G_{np}$

- **$n$  and  $p$  do not uniquely determine the graph!**
  - The graph is a result of a random process
- We can have many different realizations given the same  $n$  and  $p$



# Properties of $G_{np}$

Degree distribution:  $P(k)$

Clustering coefficient:  $C$

Path length:  $h$

What are the values of  
these properties for  $G_{np}$ ?

# (1) Degree Distribution of $G_{np}$

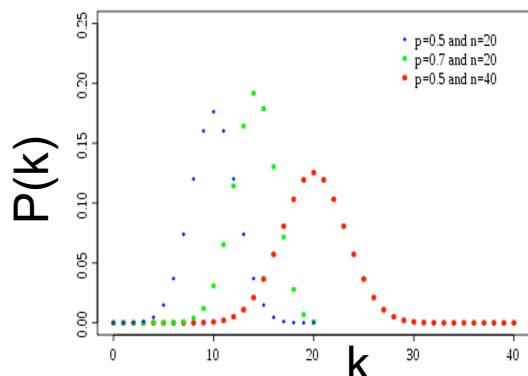
- Fact: Degree distribution of  $G_{np}$  is binomial.
- Let  $P(k)$  denote the fraction of nodes with degree  $k$ :

$$P(k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}$$

Select  $k$  nodes out of  $n-1$

Probability of having  $k$  edges

Probability of missing the rest of the  $n-1-k$  edges



Mean, variance of a binomial distribution

$$\bar{k} = p(n-1)$$

$$\sigma^2 = p(1-p)(n-1)$$

## (2) Clustering Coefficient of $G_{np}$

**Remember:**  $C_i = \frac{2e_i}{k_i(k_i - 1)}$

Where  $e_i$  is the number  
of edges between i's  
neighbors

Edges in  $G_{np}$  appear i.i.d. with prob.  $p$

So, expected  $E[e_i]$  is:  $= p \frac{k_i(k_i - 1)}{2}$

Each pair is connected  
with prob.  $p$

Number of distinct pairs of  
neighbors of node  $i$  of degree  $k_i$

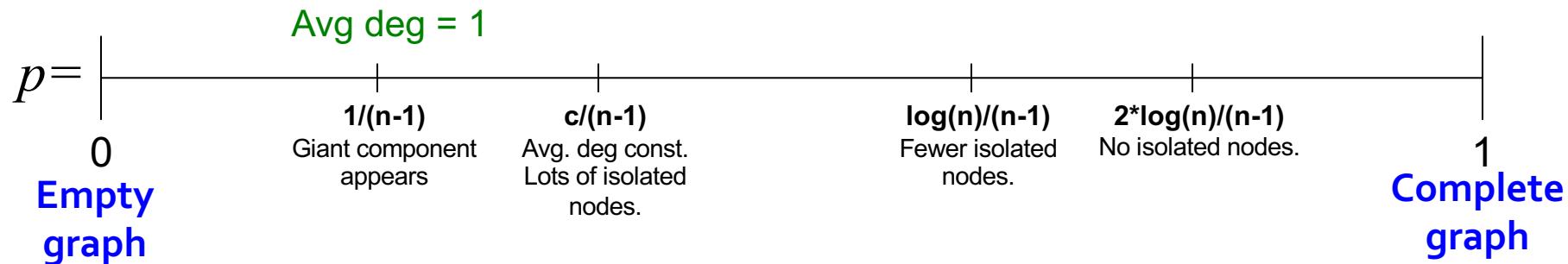
Then  $E[C_i]$ :  $= \frac{p \cdot k_i(k_i - 1)}{k_i(k_i - 1)} = p = \frac{\bar{k}}{n-1} \approx \frac{\bar{k}}{n}$

Clustering coefficient of a random graph is small.

If we generate bigger and bigger graphs with fixed avg. degree  $k$  (that is we set  $p = k \cdot 1/n$ ), then  $C$  decreases with the graph size  $n$ .

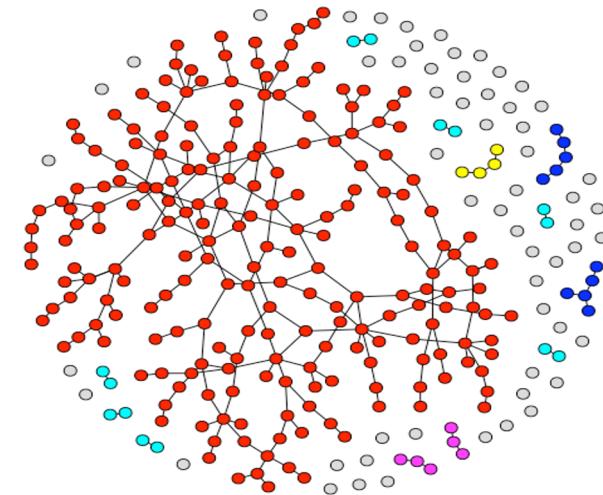
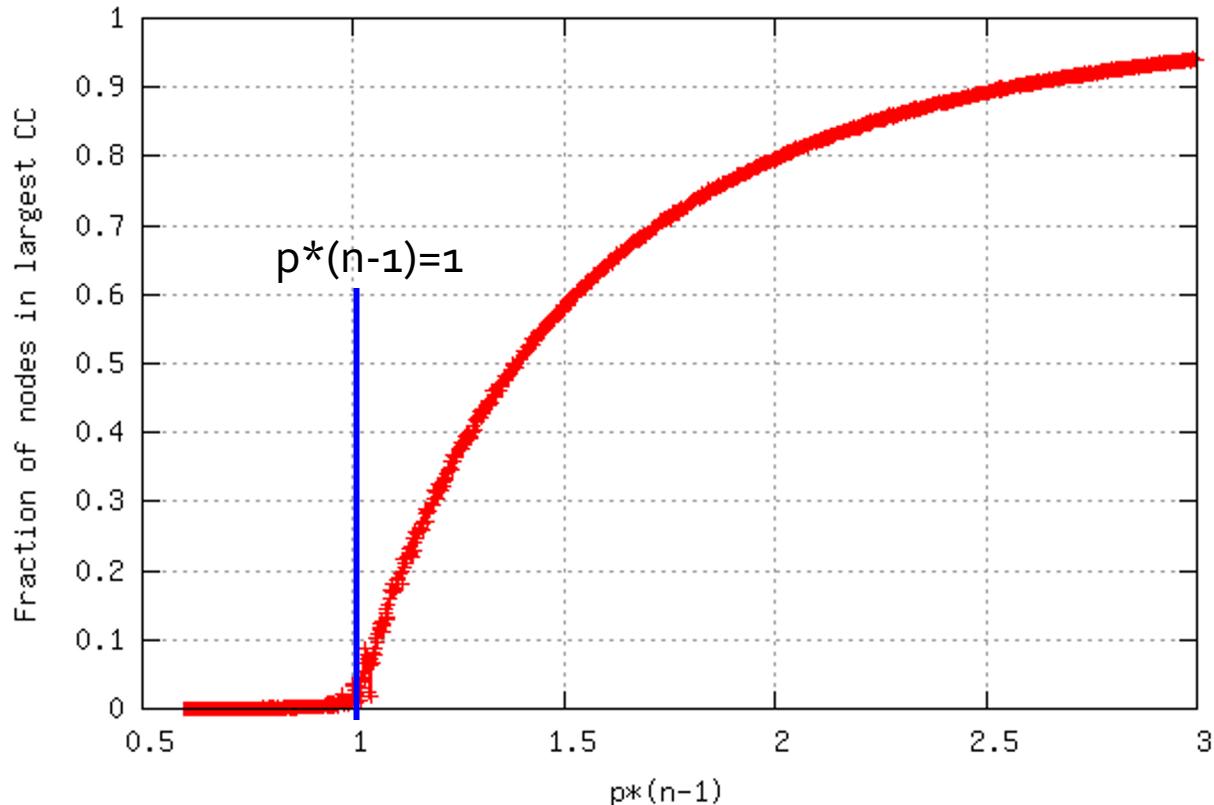
# (3) Connected Components of $G_{np}$

- Graph structure of  $G_{np}$  as  $p$  changes:



- Emergence of a giant component:
  - avg. degree  $k=2E/n$  or  $p=k/(n-1)$ 
    - Degree  $k=1-\varepsilon$ : all components are of size  $\Omega(\log n)$
    - Degree  $k=1+\varepsilon$ : 1 component of size  $\Omega(n)$ , others have size  $\Omega(\log n)$
    - Each node has at least one edge in expectation

# $G_{np}$ Simulation Experiment



Fraction of nodes in the largest component

$G_{np}, n=100,000, k=p(n-1) = 0.5 \dots 3$

# Network Properties of $G_{np}$

**Degree distribution:**

$$P(k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}$$

**Clustering coefficient:**

$$C = p = \bar{k}/n$$

**Connectivity:**

GCC exists  
when  $k > 1$ .

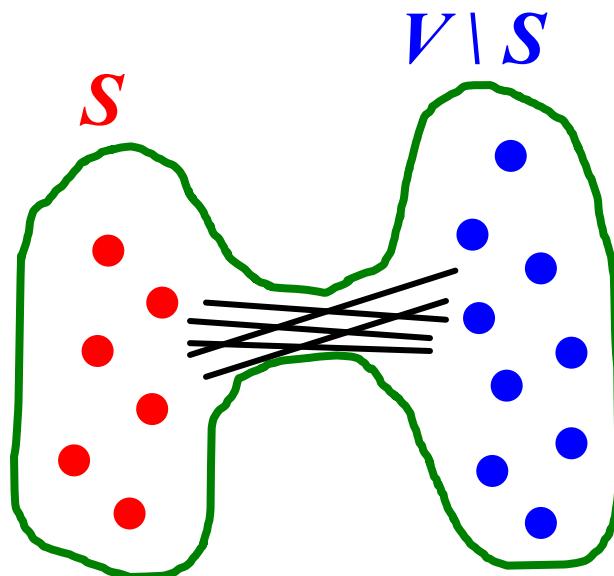
**Path length:**

*next!*

# Def: Expansion

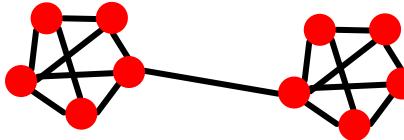
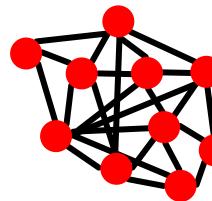
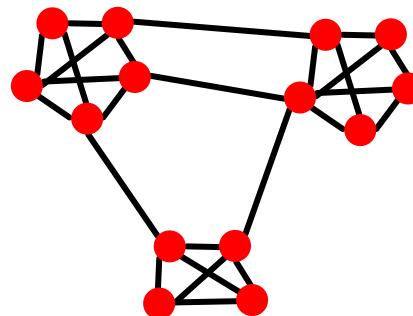
- Graph  $G(V, E)$  has **expansion  $\alpha$** : if  $\forall S \subseteq V$ :  
# of edges leaving  $S \geq \alpha \cdot \min(|S|, |V \setminus S|)$
- **Or equivalently:**

$$\alpha = \min_{S \subseteq V} \frac{\#\text{edges leaving } S}{\min(|S|, |V \setminus S|)}$$



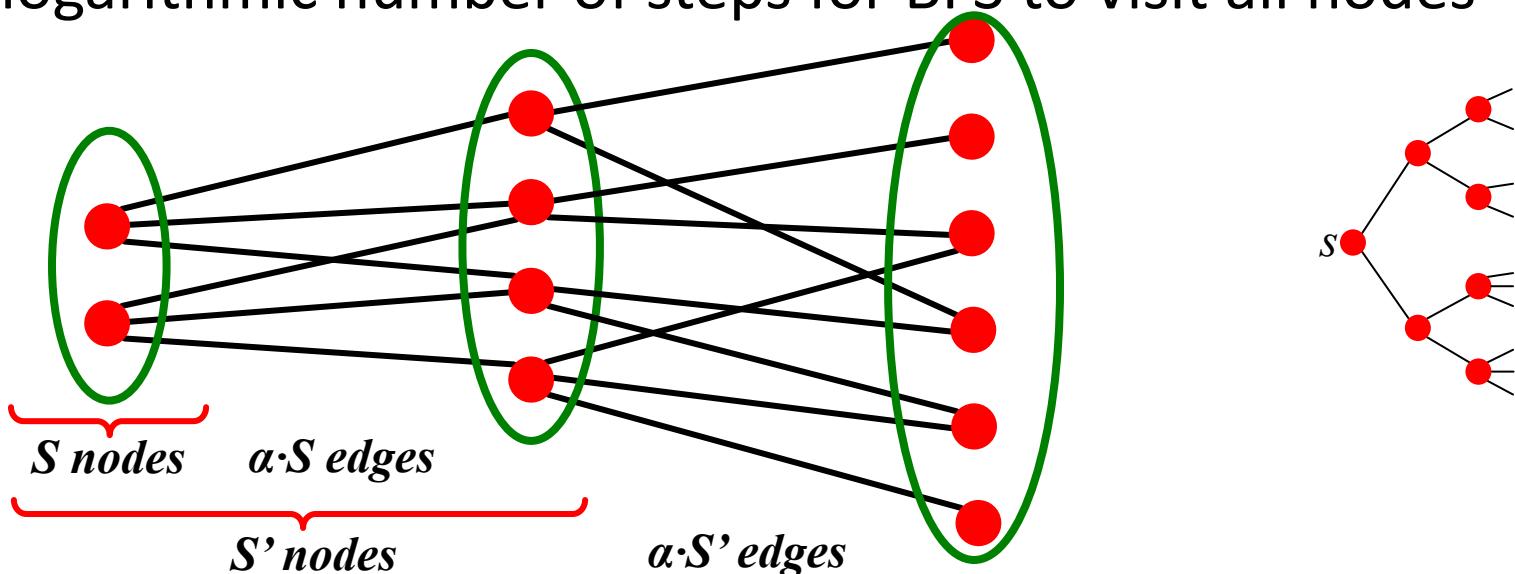
# Expansion: Measures Robustness

$$\alpha = \min_{S \subseteq V} \frac{\#\text{edges leaving } S}{\min(|S|, |V \setminus S|)}$$

- Expansion is **measure of robustness**:
  - To disconnect  $l$  nodes, we need to cut  $\geq \alpha \cdot l$  edges
- Low expansion:
- High expansion:
- Social networks:
  - “Communities”

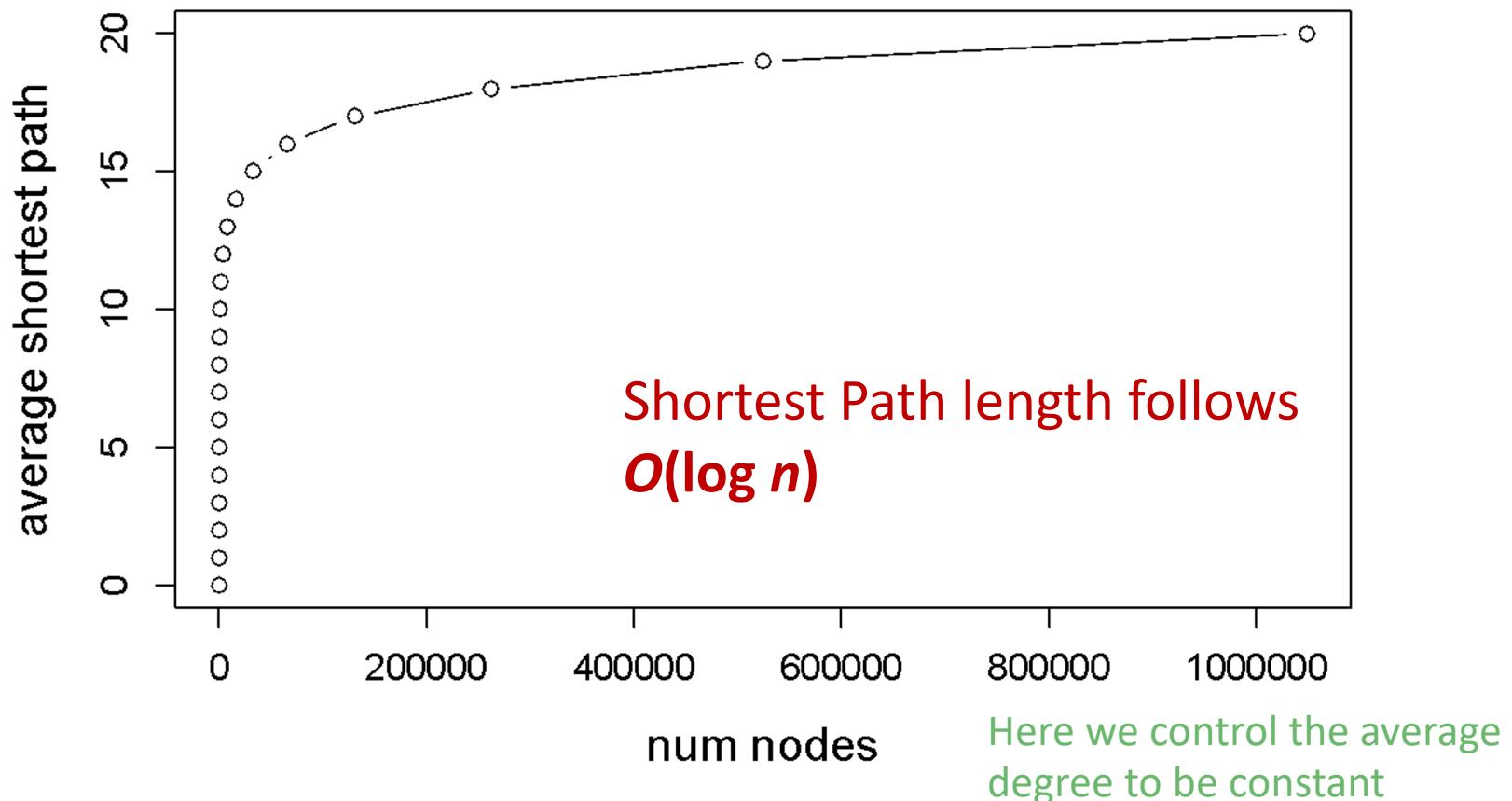
# Expansion: Random Graphs

- **Fact:** In a graph on  $n$  nodes with expansion  $\alpha$  for all pairs of nodes there is a path of length  $O((\log n)/\alpha)$ .
- **Random graph  $G_{np}$ :**  
For  $\log n > np > c$ ,  $\text{diam}(G_{np}) = O(\log n / \log(np))$ 
  - Random graphs have good expansion so it takes a logarithmic number of steps for BFS to visit all nodes



# (4) Shortest Path of $G_{np}$

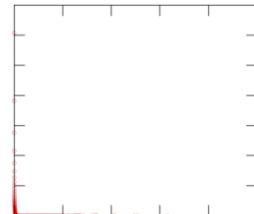
Erdös-Renyi Random Graph can grow very large but nodes will be just a few hops apart



# Back to MSN vs. $G_{np}$

Degree distribution:

MSN



$G_{np}$

$n=180M$



Avg. path length:

6.6

$O(\log n)$



Avg. clustering coef.: 0.11

$\bar{k} / n$



Largest Conn. Comp.: 99%

= Not random graph !

GCC exists  
when  $\bar{k} > 1$ .

$\bar{k} \approx 14$ .



# Real Networks vs. $G_{np}$

- **Are real networks like random graphs?**
  - Giant connected component: 😊
  - Average path length: 😊
  - Clustering Coefficient: 😞
  - Degree Distribution: 😞
- **Problems with the random networks model:**
  - Degree distribution differs from that of real networks
  - Giant component in most real networks does NOT emerge through a phase transition
  - No local structure – clustering coefficient is too low
- **Most important: Are real networks random?**
  - The answer is simply: **NO!**

# **Stanford CS224W:** **The Small-World Model**

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

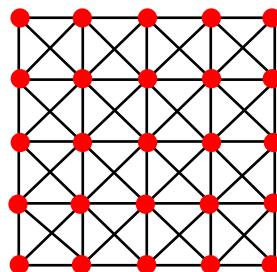
<http://cs224w.stanford.edu>



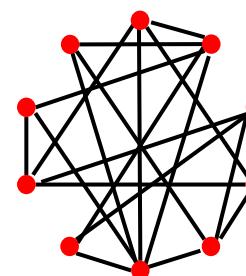
# Motivation for Small-World

	MSN	$G_{np}$	$n=180M$
Avg. path length:	6.6	$O(\log n)$ $h \approx 8.2$	<input checked="" type="checkbox"/>
Avg. clustering coef.:	0.11	$\bar{k} / n$ $C \approx 8 \cdot 10^{-8}$	<input type="checkbox"/>

Can we have **high clustering** while also having **short paths**?



Vs.



Regular lattice graph:  
High clustering coefficient  
High diameter

$G_{np}$  random graph:  
Low clustering coefficient  
Low diameter

# Clustering Implies Edge Locality

- Real networks have high clustering:
  - MSN network has 7 orders of magnitude larger clustering than the corresponding  $G_{np}$ !
- Other examples:

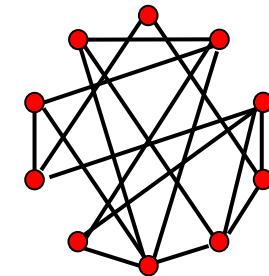
Network	Nodes	Degree	$h_{\text{actual}}$	$h_{\text{random}}$	Avg. path length	Clustering coefficients
			$C_{\text{actual}}$	$C_{\text{random}}$		
Film actor collaborations	225,226	61.00	3.65	2.99	0.79	0.00027
Power Grid	4,941	2.67	18.70	12.40	0.080	0.005
C. elegans	282	14.00	2.65	2.25	0.28	0.05

“actual” ... real network

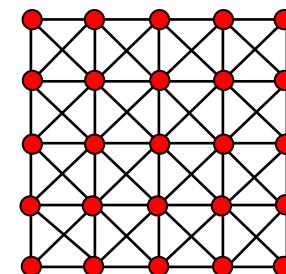
“random” ... random graph with same avg. degree

# The “Controversy”

- Consequence of expansion:
  - Short paths:  $O(\log n)$ 
    - This is the smallest diameter we can get if we keep the degree constant.
  - But clustering is low! → *no local structure.*
- But networks have “local” structure:
  - Triadic closure:  
Friend of a friend is my friend
  - High clustering but diameter is also high
- How can we have both?



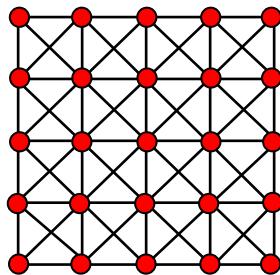
$G_{np}$  random graph:  
Low clustering coefficient  
Low diameter



Regular lattice graph:  
High clustering coefficient  
High diameter

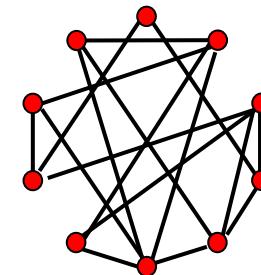
# Small-world Graphs: Idea

- Idea: Interpolate between regular lattice graphs and  $G_{np}$  random graph



Regular lattice graph:  
High clustering coefficient  
High diameter

Interpolate



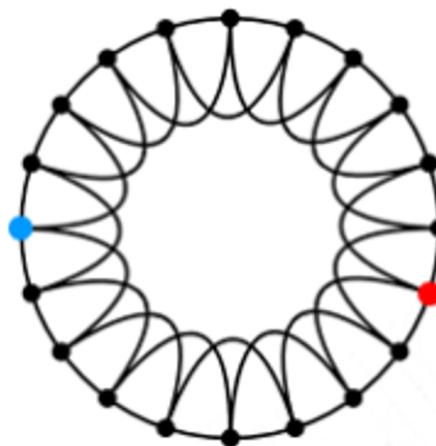
Small-world graph:  
High clustering coefficient  
Low diameter

$G_{np}$  random graph:  
Low clustering coefficient  
Low diameter

- How do we interpolate between these two graphs?

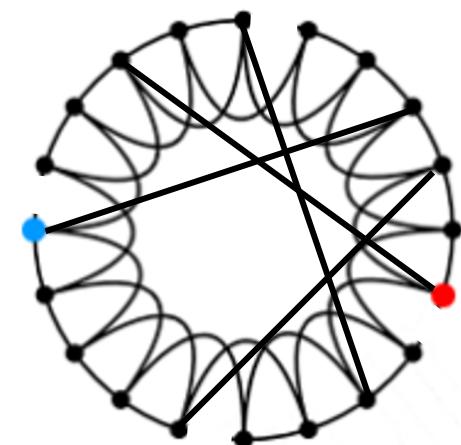
# Solution: The Small-World Model

- **Small-World Model**
- Two components to the model:
- **(1) Start with a low-dimensional regular lattice**
  - (In our case we are using a ring as a lattice)
  - Has high clustering coefficient

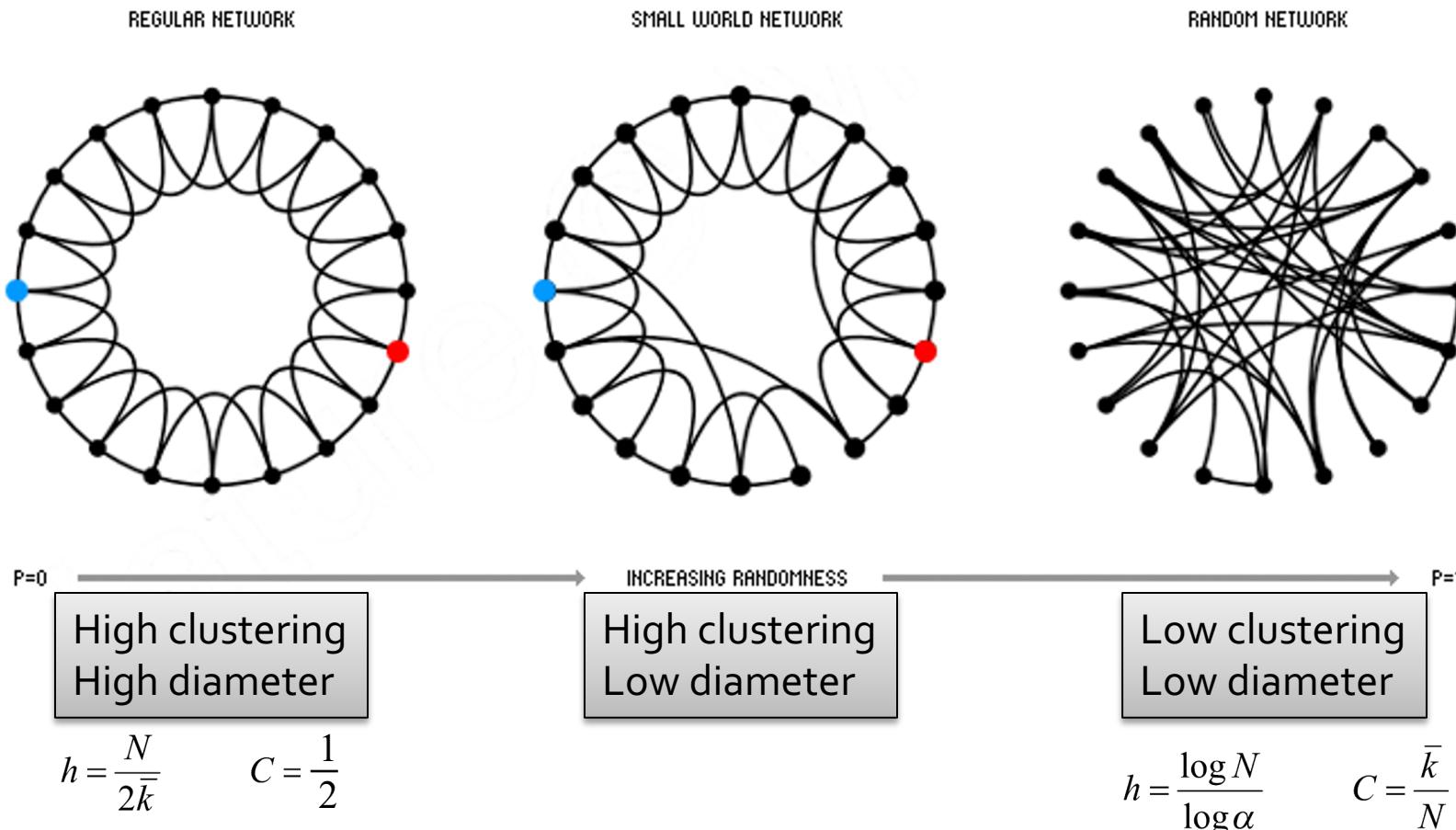


# Solution: The Small-World Model

- **Small-World Model**
- Two components to the model:
- **(2) Rewire: Introduce randomness (“shortcuts”)**
  - Add/remove edges to create shortcuts to join remote parts of the lattice
  - For each edge, with prob.  $p$ , move the other endpoint to a random node

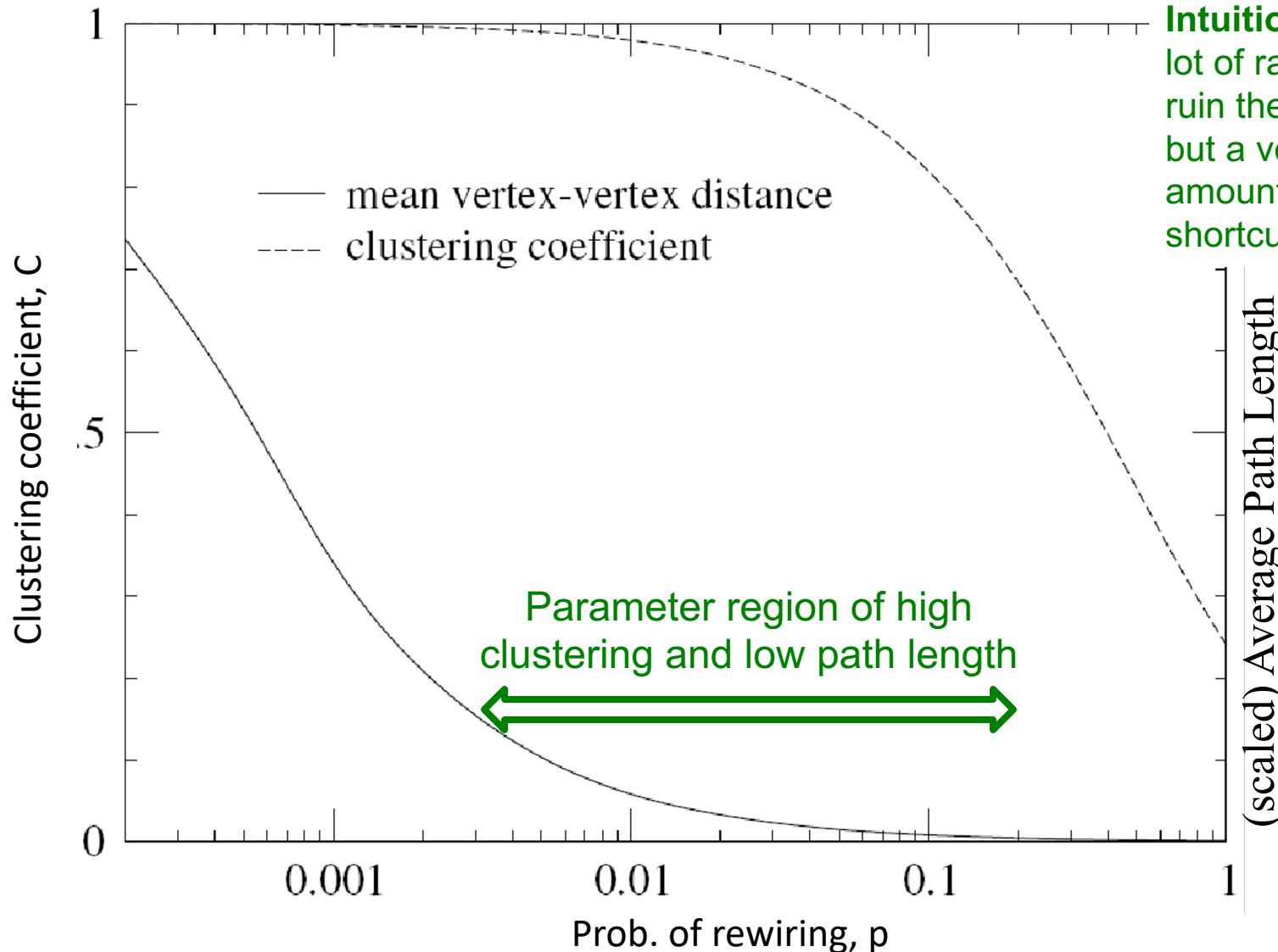


# The Small-World Model



Rewiring allows us to “interpolate” between a regular lattice and a random graph

# The Small-World Model



**Intuition:** It takes a lot of randomness to ruin the clustering, but a very small amount to create shortcuts.

# Small-World: Summary

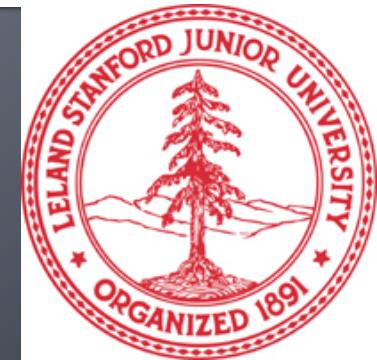
- Could a network with high clustering be at the same time a small world?
  - Yes! You don't need more than a few random links
- **The Small-World Model:**
  - Provides insight on the interplay between clustering and the small-world
  - Captures the structure of many realistic networks
  - Accounts for the high clustering of real networks
  - Does not lead to the correct degree distribution

# Stanford CS224W: Kronecker Graph Model

CS224W: Machine Learning with Graphs

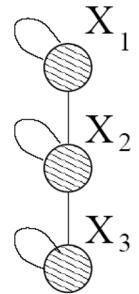
Jure Leskovec, Stanford University

<http://cs224w.stanford.edu>

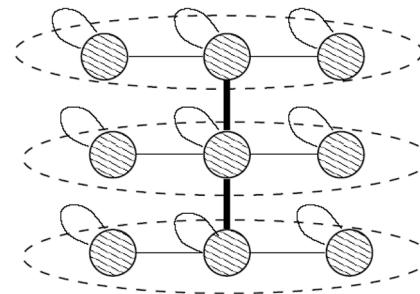


# Idea: Recursive Graph Generation

- How can we think of network structure recursively? Intuition: Self-similarity
  - Object is similar to a part of itself: the whole has the same shape as one or more of the parts
- Mimic recursive graph/community growth:



Initial graph



Recursive expansion

- Kronecker product is a way of generating self-similar matrices

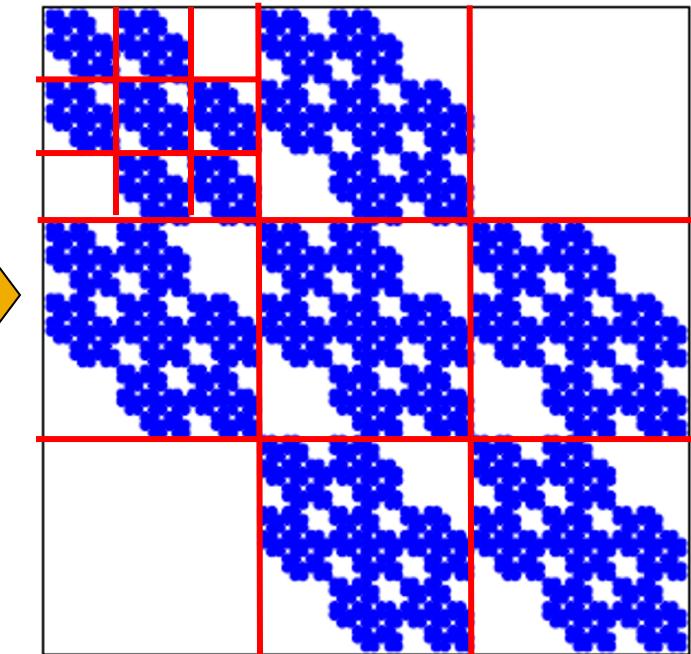
# Kronecker Graph

- Kronecker graphs:
  - A recursive model of network structure

1	1	0
1	1	1
0	1	1

 $K_1$  $3 \times 3$ 

$$\begin{array}{ccc} K_1 & K_1 & 0 \\ K_1 & K_1 & K_1 \\ 0 & K_1 & K_1 \end{array}$$
$$K_2 = K_1 \otimes K_1$$

 $9 \times 9$  $81 \times 81$  adjacency matrix

# Kronecker Product: Definition

- Kronecker product of matrices  $A$  and  $B$  is given by

$$\mathbf{C} = \mathbf{A} \otimes \mathbf{B} \doteq \begin{pmatrix} a_{1,1}\mathbf{B} & a_{1,2}\mathbf{B} & \dots & a_{1,m}\mathbf{B} \\ a_{2,1}\mathbf{B} & a_{2,2}\mathbf{B} & \dots & a_{2,m}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1}\mathbf{B} & a_{n,2}\mathbf{B} & \dots & a_{n,m}\mathbf{B} \end{pmatrix}_{N^*K \times M^*L}$$

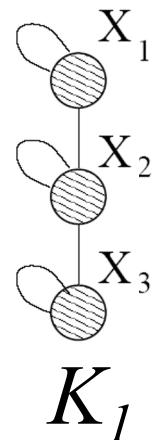
- Define a Kronecker product of two graphs as a Kronecker product of their **adjacency matrices**

# Kronecker Graphs

- Kronecker graph is obtained by growing sequence of graphs by iterating the Kronecker product over the initiator matrix  $K_1$ :

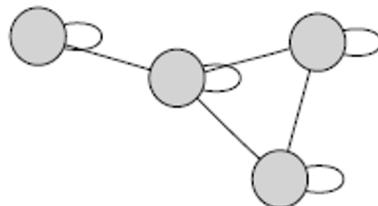
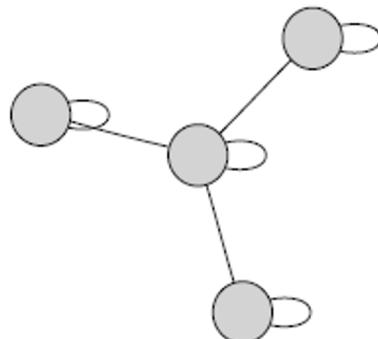
$$K_1^{[m]} = K_m = \underbrace{K_1 \otimes K_1 \otimes \dots \otimes K_1}_{m \text{ times}} = K_{m-1} \otimes K_1$$

1	1	0
1	1	1
0	1	1



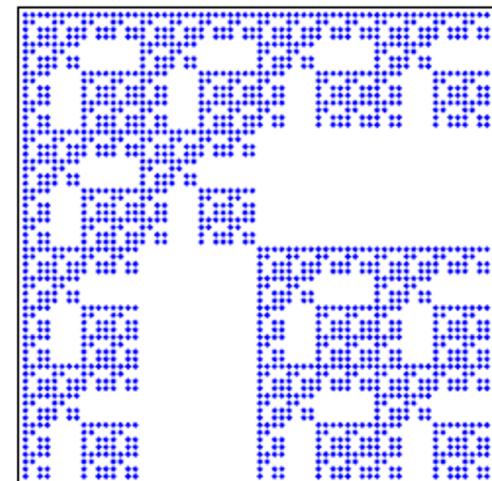
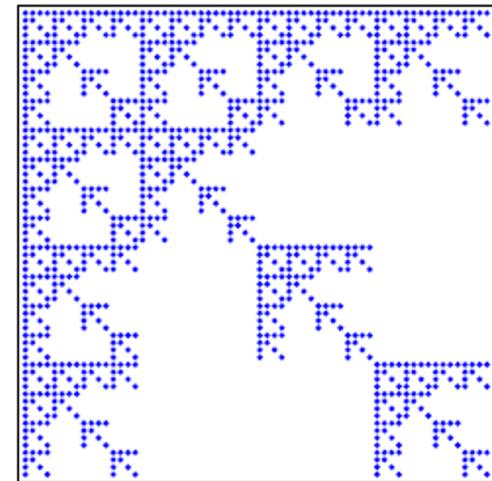
- Note: One can easily use multiple initiator matrices ( $K_1'$ ,  $K_1''$ ,  $K_1'''$ ) (even of different sizes)

# Kronecker Initiator Matrices

Initiator  $K_1$ 

1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	1

1	1	1	1
1	1	0	0
1	0	1	1
1	0	1	1

 $K_1$  adjacency matrix $K_3$  adjacency matrix

# Stochastic Kronecker Graphs

- **Step 1:** Create  $N_1 \times N_1$  probability matrix  $\Theta_1$
  - **Step 2:** Compute the  $k^{th}$  Kronecker power  $\Theta_k$
  - **Step 3:** For each entry  $p_{uv}$  of  $\Theta_k$  include an edge  $(u, v)$  in  $K_k$  with probability  $p_{uv}$

→ 청탁금지법 제1항 위반

0.5	0.2
0.1	0.3

Θ

# Kronecker multiplication

0.25	0.10	0.10	0.04
0.05	0.15	0.02	0.06
0.05	0.02	0.15	0.06
0.01	0.03	0.03	0.09

$$\Theta_2 = \Theta_1 \otimes \Theta_1$$

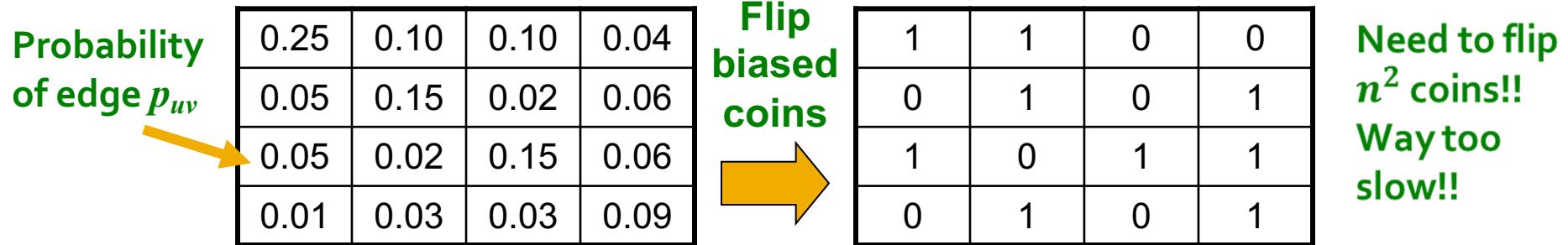
## Probability of edge $p_{uv}$

# Instance matrix $K_2$

# Flip biased coins

# Generation of Kronecker Graphs

- How do we generate an instance of a (Directed) stochastic Kronecker graph?



- Is there a faster way? YES!
- Idea: Exploit the recursive structure of Kronecker graphs
  - “Drop” edges onto the graph one by one

# Generation of Kronecker Graphs

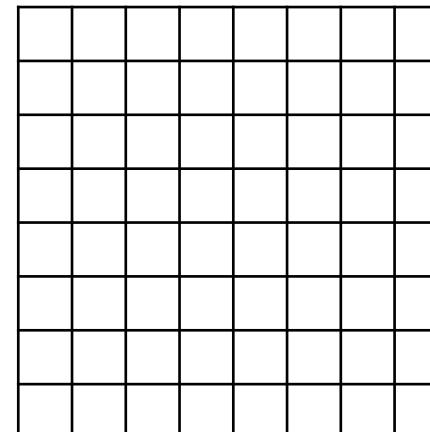
- A faster way to generate Kronecker graph

$$\Theta = \begin{array}{|c|c|} \hline a & b \\ \hline c & d \\ \hline \end{array} \rightarrow \Theta \otimes \Theta = \begin{array}{|c|c|c|c|} \hline v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & a \cdot a & a \cdot b & b \cdot a & b \cdot b \\ \hline v_2 & a \cdot c & a \cdot d & b \cdot c & b \cdot d \\ \hline v_3 & c \cdot a & c \cdot b & d \cdot a & d \cdot b \\ \hline v_4 & c \cdot c & c \cdot d & d \cdot c & d \cdot d \\ \hline \end{array}$$

$\Theta \otimes \Theta$

$$= \begin{array}{|c|c|c|c|} \hline v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & a & b & a & b \\ \hline v_2 & \mathbf{a} & c & d & c \\ \hline v_3 & c & d & c & d \\ \hline v_4 & \mathbf{c} & \mathbf{d} & \mathbf{c} & \mathbf{d} \\ \hline \end{array}$$

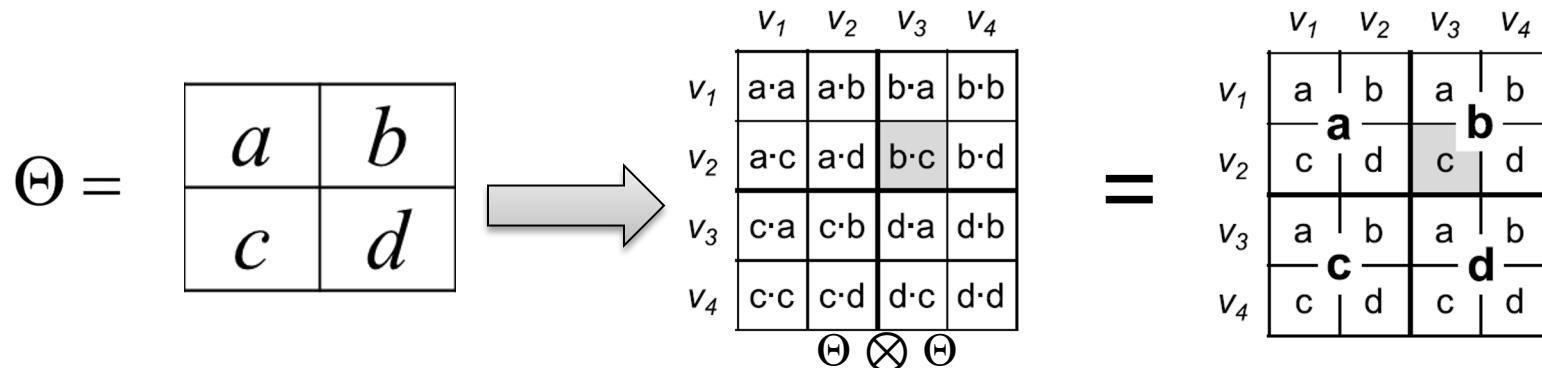
- How to “drop” an edge into a graph  $G$  on  $n=2^m$  nodes



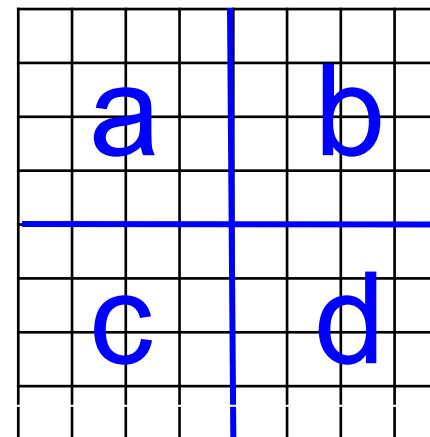
Adjacency matrix  $G$

# Generation of Kronecker Graphs

- A faster way to generate Kronecker graph



- How to “drop” an edge into a graph G on  $n=2^m$  nodes



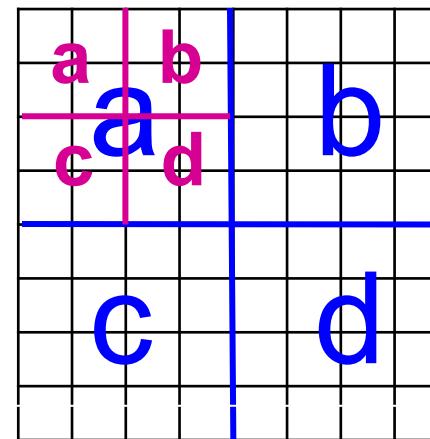
Adjacency matrix  $G$

# Generation of Kronecker Graphs

- A faster way to generate Kronecker graph

$$\Theta = \begin{array}{|c|c|} \hline a & b \\ \hline c & d \\ \hline \end{array} \rightarrow \Theta \otimes \Theta = \begin{array}{|c|c|c|c|} \hline v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & a \cdot a & a \cdot b & b \cdot a & b \cdot b \\ \hline v_2 & a \cdot c & a \cdot d & b \cdot c & b \cdot d \\ \hline v_3 & c \cdot a & c \cdot b & d \cdot a & d \cdot b \\ \hline v_4 & c \cdot c & c \cdot d & d \cdot c & d \cdot d \\ \hline \end{array}$$

- How to “drop” an edge into a graph  $G$  on  $n=2^m$  nodes



Adjacency matrix  $G$

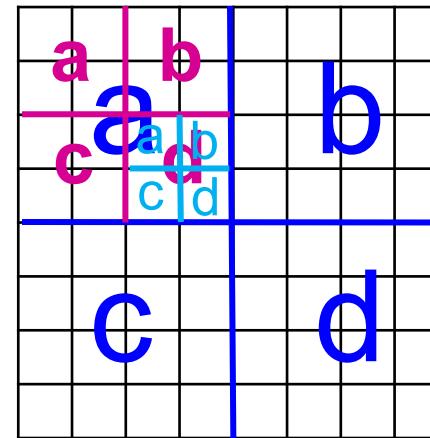
# Generation of Kronecker Graphs

- A faster way to generate Kronecker graph

$$\Theta = \begin{array}{|c|c|} \hline a & b \\ \hline c & d \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|c|} \hline v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & a \cdot a & a \cdot b & b \cdot a & b \cdot b \\ \hline v_2 & a \cdot c & a \cdot d & b \cdot c & b \cdot d \\ \hline v_3 & c \cdot a & c \cdot b & d \cdot a & d \cdot b \\ \hline v_4 & c \cdot c & c \cdot d & d \cdot c & d \cdot d \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & a & b & a & b \\ \hline v_2 & \textcolor{red}{a} & d & c & \textcolor{red}{d} \\ \hline v_3 & c & b & a & b \\ \hline v_4 & c & d & c & d \\ \hline \end{array}$$

$\Theta \otimes \Theta$

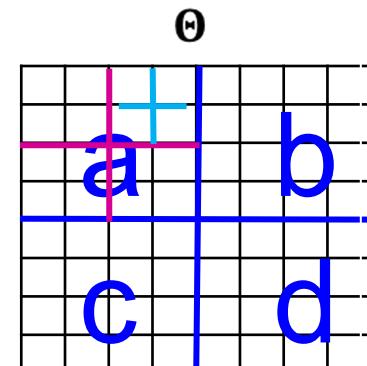
- How to “drop” an edge into a graph G on  $n=2^m$  nodes:
- We may get a few edges colliding. We simply reinsert them.



Adjacency matrix  $G$

# Generation of Kronecker Graphs

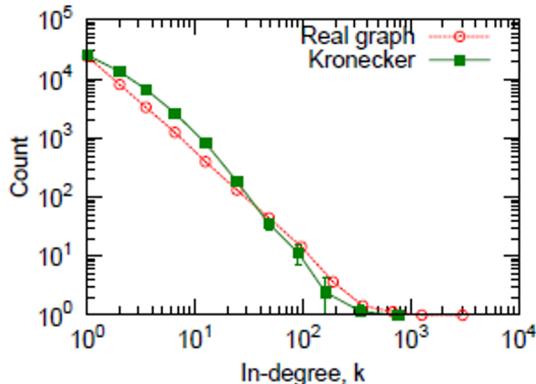
- Fast Kronecker generator algorithm:
  - For generating **directed graphs**
- **Insert 1 edge on graph  $G$  on  $n = 2^m$  nodes:**
  - Create normalized matrix  $L_{uv} = \Theta_{uv}/(\sum_{op} \Theta_{op})$
  - **For  $i = 1 \dots m$** 
    - Start with  $x = 0, y = 0$
    - Pick a row/column  $(u, v)$  with prob.  $L_{uv}$
    - Descend into quadrant  $(u, v)$  at level  $i$  of  $G$ 
      - This means:  $x += u \cdot 2^{m-i}, y += v \cdot 2^{m-i}$
  - Add an edge  $(x, y)$  to  $G$



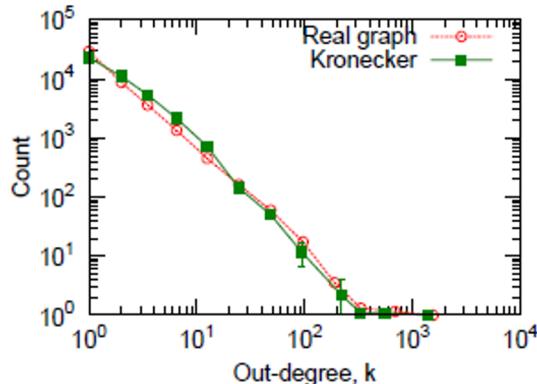
# Estimation: Epinions (n=76k, m=510k)

- Real and Kronecker are very close:

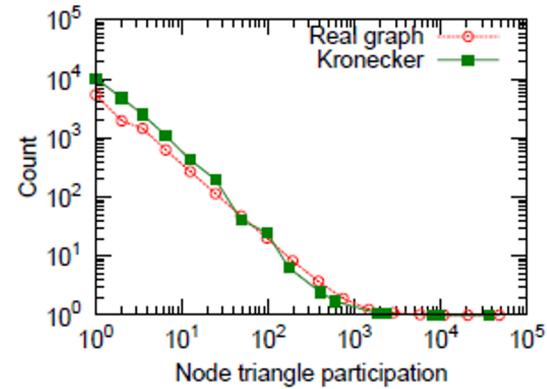
$$\Theta_1 = \begin{bmatrix} 0.99 & 0.54 \\ 0.49 & 0.13 \end{bmatrix}$$



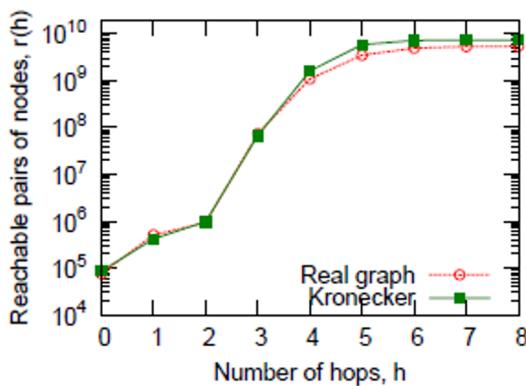
(a) In-Degree



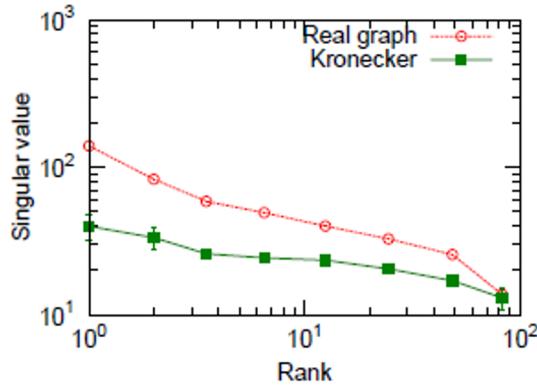
(b) Out-degree



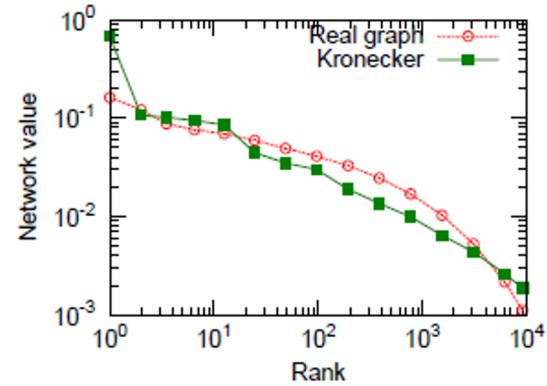
(c) Triangle participation



(d) Hop plot



(e) Scree plot



(f) "Network" value

# Summary of the Lecture

- **Today: Traditional graph generative models**
  - Erdös-Renyi graphs
  - Small-world graphs
  - Kronecker graphs
  - All these models have **prior assumption of the graph generation processes**
- **Next: Deep graph generative models**
  - **Learn** the graph generation process **from raw data**