Social and Technological Networks

University of Edinburgh - Dr. Rik Sarkar

Lecture Notes

Autumn 2018

Josh Spicer

Primary Resources

***http://www.inf.ed.ac.uk/teaching/courses/stn/* (Dr. Rik Sarkar)**
http://www.cs.cornell.edu/home/kleinber/networks-book/
http://www-bcf.usc.edu/~dkempe/teaching/structure-dynamics.pdf
https://www.cs.cornell.edu/jeh/book.pdf

[2] Random Graphs

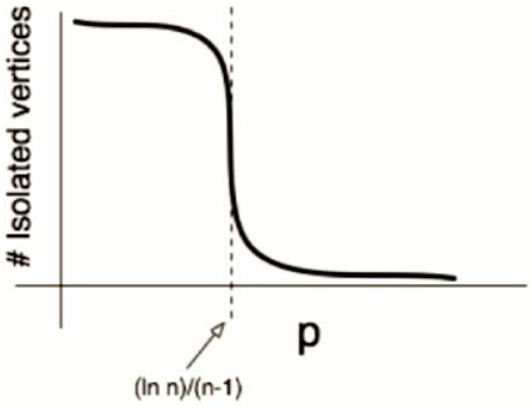
Erdos – Renyi Random graphs

$$\mathcal{G}(n,p)$$

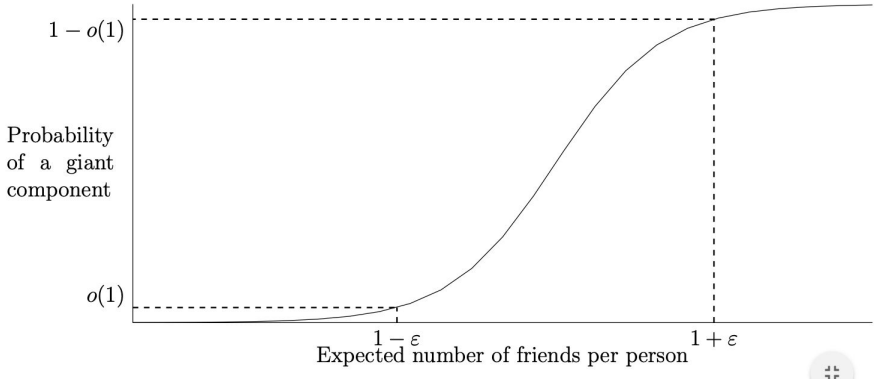
- n: number of vertices
- p: probability that any particular edge exists
  - Take V with n vertices
  - Consider each possible edge. Add it to E with probability p
- Expected total number of edges $\binom{n}{2}p$
- Expected number of edges at any vertex $(n-1)p$

- Often **p** is a function of **n,** such as **p = d/n**, for some constant **d**.
- In this case, expected degree of a vertex is approx. **d.**
- *conceptually it is helpful to think of **n** as both the total number of vertices and as the number of potential neighbors of any given node*

- Isolated vertices are
- Likely when: $p < \frac{\ln n}{n}$
- Unlikely when: $p > \frac{\ln n}{n}$



Union bound: Pr[A or B or C ...] ≤ Pr[A] + Pr[B] + Pr[C] + …



For small **p**, with **p = d/n, d < 1**, each connected component in the graph is small.
For **d > 1**, there is a giant component consisting of a constant fraction of the vertices.
In addition, there is a rapid transition at the threshold d = 1. Below the threshold, the probability of a giant component is very small, and above the threshold, the probability is almost one.

The degree distribution of $G(n,p)$ for general p is also binomial. Since p is the probability of an edge being present, the expected degree of a vertex is $p(n-1) \approx pn$. The degree distribution is given by

$$\text{Prob}(\text{vertex has degree } k) = \binom{n-1}{k} p^k (1-p)^{n-k-1} \approx \binom{n}{k} p^k (1-p)^{n-k}.$$

The quantity **(n choose k)** is the number of ways of choosing **k** edges, out of the possible **n** edges, and $\mathbf{p^k(1-p)^{n-k}}$ is the probability that the **k** selected edges are present and the remaining **n-k** edges are not.

Existence of Triangles
As the number of vertices increases, the probability of an edge between two specific vertices decreases linearly with n. (Even though the # of triples grows with n^3).
Thus the probability of all three edges between the pairs of vertices in a triple of vertices being present goes down as n^-3, exactly canceling the rate of growth of triples.

A random graph with n vertices and edge probability d/n has an **expected number of triangles** that is INDEPENDENT of n, namely it is $\mathbf{d^3/6}$.

$$E(x) = E\left(\sum_{ijk} \Delta_{ijk}\right) = \sum_{ijk} E(\Delta_{ijk}) = \binom{n}{3}\left(\frac{d}{n}\right)^3 \approx \frac{d^3}{6}.$$

Lower Bound of Triangles

Even though we can expect on average to have $d^3/6$ triangles, there are still instances where a graph may not a triangle. For instance, we haven't said whether that average was generated due to half of all graphs having probability ($d^3/3$) triangles and the other half having none ($d^3/3$ + 0 / 2 == our expected value….lame). It can be shown that when d > ~1.8, the random graph has a triangle with nonzero probability.

Q 1. *Show that a connected graph has $\Omega(n)$ triads (counting both open and closed).*

Answer. As per the definition of $\Omega(n)$, we need to show that for some constant $c > 0$, number of triad in a connected graph $T > c.n$ for $n > n_0$.

We would prove this by induction. In a minimally connected graph, a tree with three vertices, the number of triad is one. This constitutes the base case with $c = 1/3$.

For induction, we are assuming there is a graph G with n nodes and T triads. In G, $T \geq \frac{n}{3}$. Now, we'll show that after adding nodes and edges to G, it still holds this property.

- Adding an edge to G: Adding an edge to G only increases the number of triads. Thus, the number of triads in the new graph $T' \geq T \geq \frac{n}{3}$.
- Adding an vertex to G: As the new graph is to be connected, there should be at least an edge connecting the newly added vertex (i) to one of the vertices (j) in G. As, G was connected there was at least an edge jk. Thus, ijk is a triad in the new graph. Thus, $T' \geq T + 1 \geq \frac{n}{3} + 1 > \frac{1}{3}(n + 1)$.

Therefore, for $n \geq 3$, $T \geq c.n$ where $c = \frac{1}{3}$.

In class we saw that a triangle or closed triad is is three vertices a, b, c with all edges ab, bc, ca between them. The number of possible triangles or triads is clearly $\binom{n}{3}$, which is $\Theta(n^3)$. The probability that a particular triangle exists is p^3.

Clustering Coefficient:
Measures how tight the friend neighborhoods are: frequency of closed triads
cc(A) = fractions of pairs of A's neighbors that are friends

Triad: If **A** and **C** have common friend **B,** then **ABC** forms a triad
- If Edge AC also exists, it is a **closed triad**

Global cc: # closed triads / # all triads
Global CC in ER Graphs:
When p is almost 0 (very small): cc is almost 0
When p is almost 1 (very large): cc is almost 1

[**] Major CC takeaway:
- **Global cc in ER graphs is vanishingly small.**
- In the limit → tends toward 0
- There is no constant c s.t cc(ER-graph) > c
- At tipping point

Giant component

- When p = **(1 - eps) / n**
    - No Giant Component, components of size O(log(n))
- When p = **(1 + eps) / n**
    - Giant Component exists
- When p = **1 / n**
    - Largest component has size $n^{2/3}$

END [2]

[3] Cascades

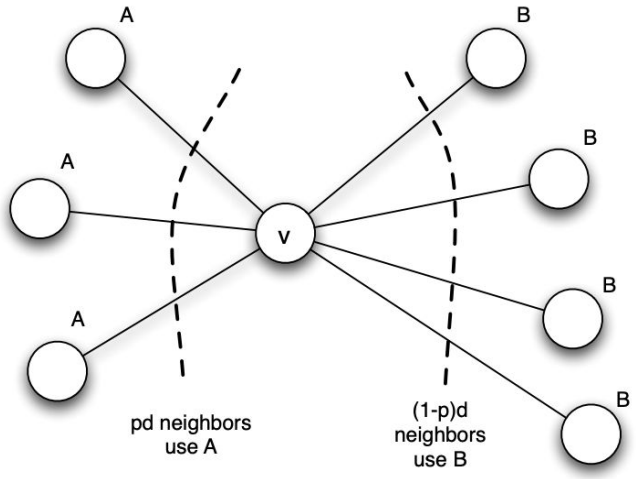
Things that spread (diffuse) along network edges.

Basic idea: Your benefit of adopting a new behavior increases as more of your friends adopt it.

V had d edges
P fraction use A
(1-p) use B
V's benefit is per A-edge
V's benefit is per B-edge

A is a better choice IF:

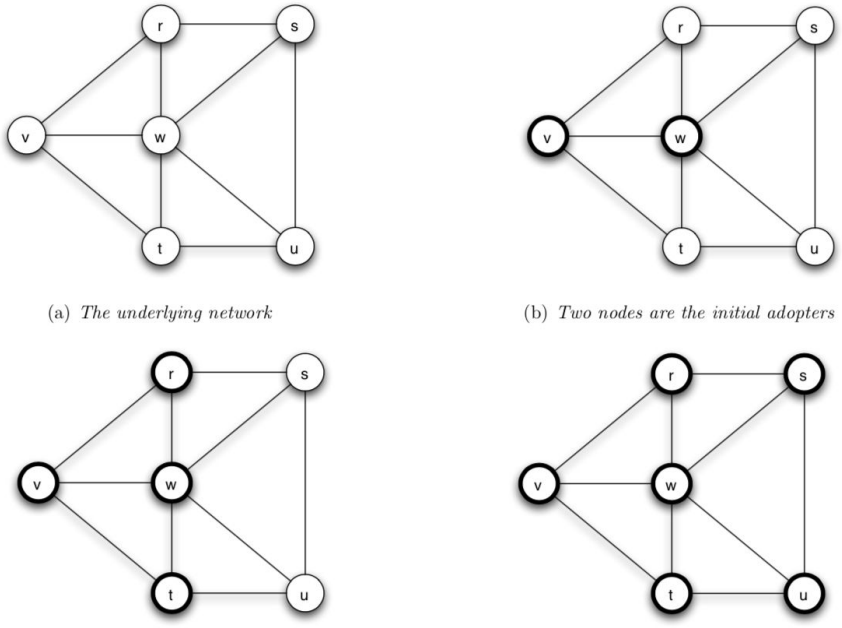
$$pda \geq (1-p)db,$$



$$p \geq \frac{b}{a+b}.$$

The Contagion Threshold

- Let us write threshold $q = b/(a+b)$
- If q is small, that means b is small relative to a
  - Therefore A is useful even if only a small fraction of neighbors are using it
- If q is large, that means the opposite is true, and B is a better choice



(a) *The underlying network*

(b) *Two nodes are the initial adopters*

(c) *After one step, two more nodes have adopted*

(d) *After a second step, everyone has adopted*

Figure 19.3: Starting with v and w as the initial adopters, and payoffs $a = 3$ and $b = 2$, the new behavior A spreads to all nodes in two steps. Nodes adopting A in a given step are drawn with dark borders; nodes adopting B are drawn with light borders.

Takeaways*

- **Tightly-knit communities** can work to **hinder the spread of an innovation**.
- In other words, clusters are the natural obstacles to cascades.
- Weak links are good for information transmission, but **not** for behavior transmission!

More detail:

Claim: Consider a set of initial adopters of behavior A, with a threshold of q for nodes in the remaining network to adopt behavior A.

(i) If the remaining network contains a cluster of density greater than $1 - q$, then the set of initial adopters will not cause a complete cascade.

(ii) Moreover, whenever a set of initial adopters does not cause a complete cascade with threshold q, the remaining network must contain a cluster of density greater than $1 - q$.

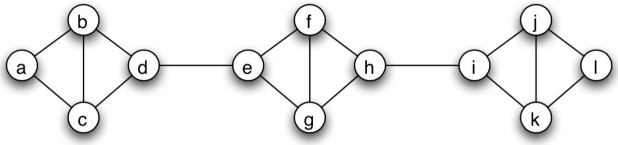
This wraps up our analysis of cascades and clusters; the punch-line is that in this model, a set of initial adopters can cause a complete cascade at threshold q if and only if the remaining network contains no cluster of density greater than $(1 - q)$. So in this sense, cascades and clusters truly are natural opposites: clusters block the spread of cascades, and whenever a cascade comes to a stop, there's a cluster that can be used to explain why.

Alpha-Strong communities (dense communities)

- The set of nodes form an alpha-strong community if for each node in subset S, the degree of each node is greater than alpha*that degree.
- That is, at least alpha fraction of neighbors of each node is within the community!
- For large alpha, this represents a tight knit community
- THM: A cascade with contagion threshold q cannot penetrate an alpha-dense community with alpha > 1 - q
  - Therefore, for a cascade with threshold q, and set X of initial adopters of A:
    1. If the rest of the network contains a cluster of density > 1-q, then the cascade from X does not result in a complete cascade
    2. If the cascade is not complete, then the rest of the network must contain a cluster of density > 1-q
- Proof is by contradiction: The first node in the cluster that converts, cannot convert (not enough “a” neighbors).

Density

- We say that a cluster of density p is a set of nodes such that each node in the set has at least a p fraction of its network neighbors in the set



- Figure 19.6: A collection of four-node clusters, each of density 2/3.

- *If you have two clusters of density p, then the union of these two cluster (ie set of nodes that lie in at least one of them) is also a cluster of density p.*

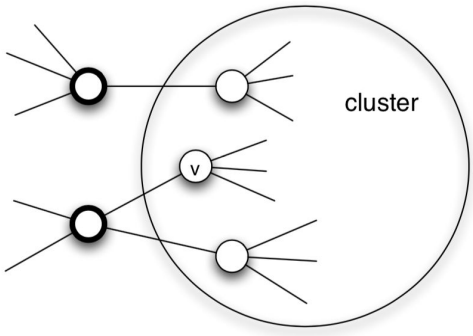
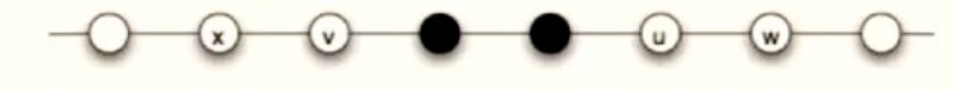


Figure 19.8: The spread of a new behavior, when nodes have threshold q, stops when it reaches a cluster of density greater than $(1 - q)$.

Cascade Capacity

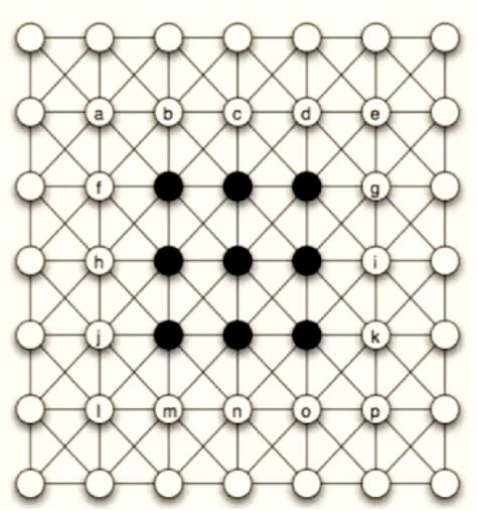
- Up to what threshold q can a small set of early adopters cause a full cascade?

  1-D grid: capacity = ½



  (Threshold must be less than half!)

  2-D grid with 8 neighbors: capacity: ⅜



  Thm: No infinite network has cascade capacity > ½

  ⇒ *An inferior technology cannot win an infinite network!*
  -or-
  In a large network, inferior technology cannot win with small starting resources

  Causing large spread of cascade!

  - Which k nodes do you convert to get as large a cascade as possible?
  - Suppose each node has a “sphere of influence” - other nearby nodes it can affect
  - Which k nodes do you select to cover most nodes within their sphere?
  - …….brings us to influence maximization!

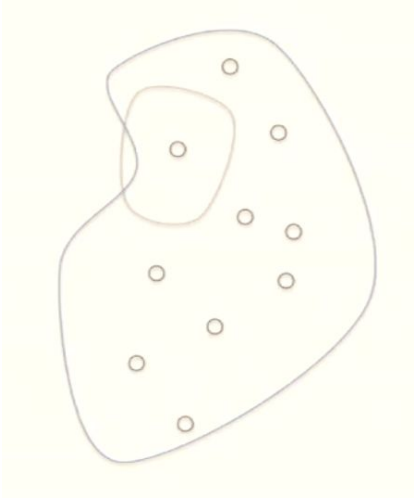
END [3]

[4] Influence Maximizations (Submodular opt)

- Final size of cascade depend on initial choice of k nodes!
- We don't know what OPT is!
    - We can't know, it's np-hard.
- Lets approximate!
- For maximizing sphere of influence problem, there is a simple algorithm that gives an approximation of (1 - 1/e)
    - To prove, you can use a property called **submodularity**

Example: camera coverage

- Suppose you placing sensors to monitor region
- N possible camera locations
- Each camera can "see" a region
- With budget of k cameras, we want to cover largest possible area



== Marginal Gains

- Marginal coverage depend on other sensors selected.
- Same problem as social sphere of influence.

Submodular functions (diminishing returns)

- Suppose function f(x) rep. Total benefit of selecting x
    - And f(S) the benefit of selecting set S
    - Function f is submodular if:

$$S \subseteq T \implies$$

$$f(S \cup \{x\}) - f(S) \geq f(T \cup \{x\}) - f(T)$$

- A selection of x gives smaller benefits if many other elements have been selected!
- <u>There's a greedy approx. algorithm</u>
    - Sensor with most area, then most marginal area, ...repeat k times
    - Will have that (1-1/e) submodular approx!

Idea: A node can influence its neighbors (same problem as above!)

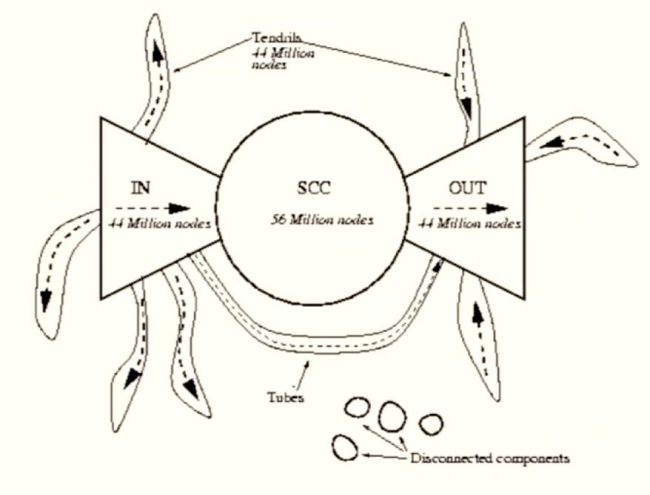
END [4]

[5] Web graphs and Ranking Pages

Structure and Analysis of WWW

Bow-tie structure of the web:

- The web has one GIANT strongly connected component
- Largely due to:
    - Many topic related to each other. (eg. wikipedia)
    - Many search sites have links to important sites



Analyzing the web to measure importance:

IDEA: In-links constitute a vote for importance

- If many people are linking to it, then likely the page is valuable to many other people as well.
- Simple idea: Count the number of in-edges
- Enhanced idea: not all links imply equal importance, links from important pages are more important
- Recursive idea: Importance depends on importance of in-edges

Algorithms that capture this idea:

1. **The HITS Algorithm**
    a. Not all pages similar
    b. Authorities:
        i. Some are important for the information they contain (e.g. course pages)
    c. Hubs:
        i. Some are important for the links they contain (e.g. list of courses)
        ii. ** they guide you to the right authority)
    d. Rank separately, but dependent on each other
        i. A hub linking to good authorities is likely good
        ii. An authority linked by good hubs is likely good

For each page p, estimate its score both as:
-- A hub: hub(p)
-- An authority: auth(p)

Repeatedly apply in each round

Update Rules:

1. Start with all hub and auth == 1
2. Apply authority update to all nodes:
    a. auth(p) = sum of all hub(q) where q → p is a link
3. Apply Hub update to all nodes:
    a. hub(p) = sum of all auth(r) where p → r is a link
4. Repeat for k rounds
5. Normalize:
    a. We need only relative values
    b. Invariant: Total hub value is 1, total auth value is 1
    c. Do by:
        i. Dividing each auth(p) by sum of all auth scores
        ii. Divide each hub(p) by the sum of all hub scores

2. PageRank

IDEA: Not all pages have good classification as hubs and authorities

- Sometimes authorities link directly to each other
    - E.g. wikipedia
- We want something with one value for every page.

Basic PageRank Algorithm:

- Overall “value” in system is conserved = 1
- Start: Assign “value” 1/n to each node.
- In each round:
    - Each node divides equal portion of its pagerank value to its out-going links
    - Updates its own value to be the sum of values it receives
    -

Difficulties of PageRank:

- Acyclic Graph
- Some nodes can get all the values
    - Lakes/seas at the local minima
- Some nodes can end without any value
    - Mountains or peaks (maxima)

Solve with Scaled pagerank!

- In every round:
    - Divide **s** fraction of your PageRank equally among neighbors
    - Divide **(1-s)** fraction equally among all nodes in the network

Random-walk Interpretation

- Suppose users start at random web pages
    - Then click links on them randomly
- Sometimes with (Pr = 1-s) they decide to leave the page and jump to a random page in the web
- **Pagerank ===> Probability of a user being at that page at any particular time**

Other improvement:

- Use textual information
- Use usage data: which links people click
- Use other contextual data (location, history, etc.)
- Adjustment to SEO
- Adaptation to the fast changing web...

PROPERTIES:

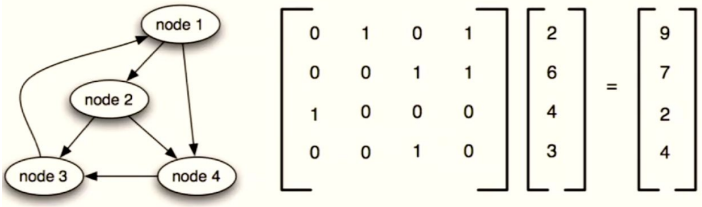
- **HITS converges**
- **Pagerank converges**
    - **They give clear values for each node**
- **** Pagerank is equivalent to a random walk! ****

~~~~~ Spectral Analysis of HITS~~~~~

HITS Analysis

- Adjacency matrices (0 or 1)
- Hubs and authority scores can be written as vectors **h** and **a**
- Updates rules are matrix multiplications

$$h \leftarrow Ma$$



-
- Hub rule for i (sum of a-values of nodes that I points to)
  - H ← Ma
- Authority rule for i (sum of h-values of nodes that point to i)
  - A ← M^T h
- Iterations:

• After one round:

$$a^{\langle 1 \rangle} = M^T h^{\langle 0 \rangle}$$

$$h^{\langle 1 \rangle} = Ma^{\langle 1 \rangle} = MM^T h^{\langle 0 \rangle}$$

• Over k rounds:

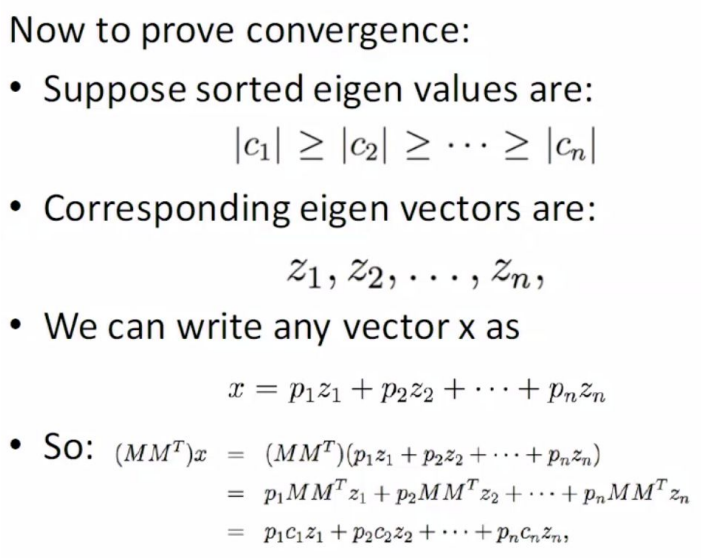
$$h^{\langle k \rangle} = (MM^T)^k h^{\langle 0 \rangle}$$

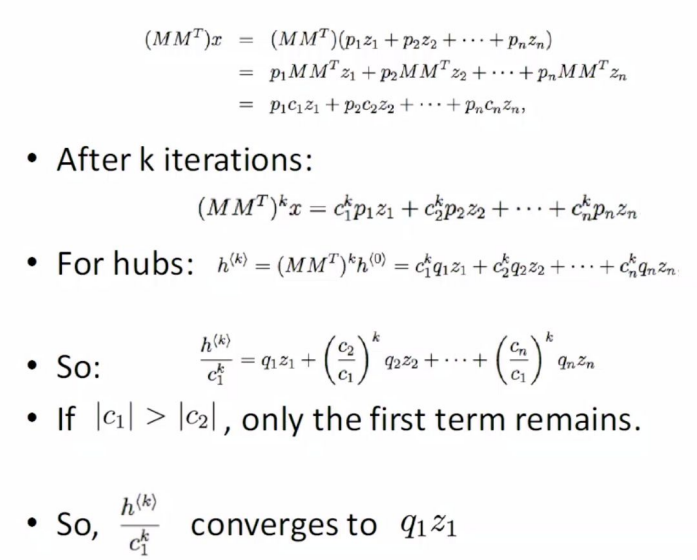
Convergence:

- Remember that **h** keeps increasing
- We want to show that the normalized value $h^{<k>}$ / C^k
- Converges to a vector of finite real numbers as k goes to infinity
- If convergence happens then, there is a CONSTANT **c** s.t:
- This is due to our normalization step on each iteration
  - $$(MM^T)h^{\langle * \rangle} = ch^{\langle * \rangle}$$
- Implies that for the matrix (MM^T), c is an EIGENVALUE, with h<*> being the corresponding EIGENVECTOR!

- Proof of convergence to eigenvectors
    - Useful linear algebra theorems:
        - As symmetric matrix has orthogonal eigenvector. *(MM^T) is symmetric*
            - They form a basis of n-D space
            - Any vector can be written as a linear combination
        - For matrix P with all positive values, Perron's theorem says:
            - A unique pos. Real valued largest eigenvalue c exists
            - Corresponding eigenvector y in unique AND has positive real coordinates.
            - If c = 1, the P^k*x converges to y.

The rest of proof:



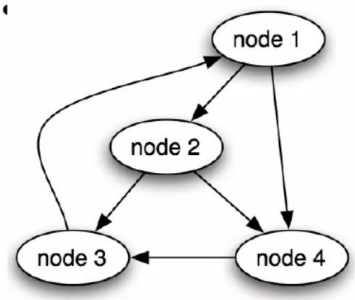


Properties

- The vector **q1z1** is a simple multiple of **z1**.
    - A vector essentially similar to the first eigenvector (same direction)
    - Therefore independent of starting values of **h**
- **Q1** can be shown to be non-zero always - so the scores are not zero.
- Interesting: No matter what **h0** you start with (whatever dist of hub and authority score), the resulting eigenvector will always be the same. Is property of matrices, not h0….

~~~ Spectral Analysis of PageRank ~~~~

Pagerank Update rule as a matrix derived from adjacency



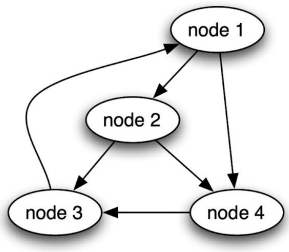
$$\begin{bmatrix} 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1/2 & 1/2 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$r \leftarrow N^T r$$

Entries of half are because it represents the distribution of "flow" from the starting node to all its children.

Over several iterations:

- Scaled pagerank:
$$r \leftarrow \tilde{N}^T r$$
- Over k iterations:
$$r^{\langle k \rangle} = (\tilde{N}^T)^k r^{\langle 0 \rangle}$$
- Pagerank does not need normalization.
$$\tilde{N}^T r^{\langle * \rangle} = r^{\langle * \rangle}$$
- We are looking for an eigen vector with eigen value=1



$$\begin{bmatrix} .05 & .45 & .05 & .45 \\ .05 & .05 & .45 & .45 \\ .85 & .05 & .05 & .05 \\ .05 & .05 & .85 & .05 \end{bmatrix}$$

Not surprising, Pagerank lets some fluid diffuse until it arrives at some steady state.

Figure 14.14: The flow of PageRank under the Scaled PageRank Update Rule can also be represented using a matrix derived from the adjacency matrix M (shown here with scaling factor $s = 0.8$). We denote this matrix by $\tilde{N}$; the entry $\tilde{N}_{ij}$ specifies the portion of i's PageRank that should be passed to j in one update step.

Rule converges to a limiting vector $r^{\langle * \rangle}$, this limit should satisfy $\tilde{N}^T r^{\langle * \rangle} = r^{\langle * \rangle}$ —
should expect $r^{\langle * \rangle}$ to be an eigenvector of $\tilde{N}^T$ with corresponding eigenvalue 1. S

PageRank and Random Walks

- A random walker is moving along random directed edges
- Suppose vector **b** shows the probabilities of walker currently being at different nodes
- Then vector $\mathbf{N}^{\mathbf{T}}\mathbf{b}$ gives the probabilities for the next step.
- Thus, pagerank values of nodes after k iterations is equal to:
    - Probability of the walker being at the node after k steps
- The final values given by the eigenvector are the steady state probabilities
    - Note that these depend only on the network and are independent of the starting points

END [5]

[6] Metrics and Network Construction

A metric is a measure **d** on a set **X**. It satisfies some intuitive property (e.g. triangle inequality).

Metrics are important:

- Metrics are used to construct networks
- Networks have metrics that determine their properties

Euclidean metrics

1-D: Straight line
2-D: Plane

Distance measure in dimension d:

$$d(u,v) = \sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2 + \cdots + (u_d - v_d)^2}$$

Non-Euclidean Metrics

Examples:

- Jungle River
- Epsilon binary distance
- Manhattan distance
- Curved sphere (earth)
- Other realistic shapes
- Hyperbolic plane (negative curvature)

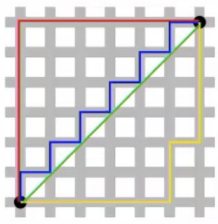
Lp Metrics

- L_p metrics

$$d(u,v) = \sqrt[p]{(u_1 - v_1)^p + (u_2 - v_2)^p + \cdots + (u_d - v_d)^p}$$

L1 metric (Manhattan Distance)

$$d(u,v) = |u_x - v_x| + |u_y - v_y|$$



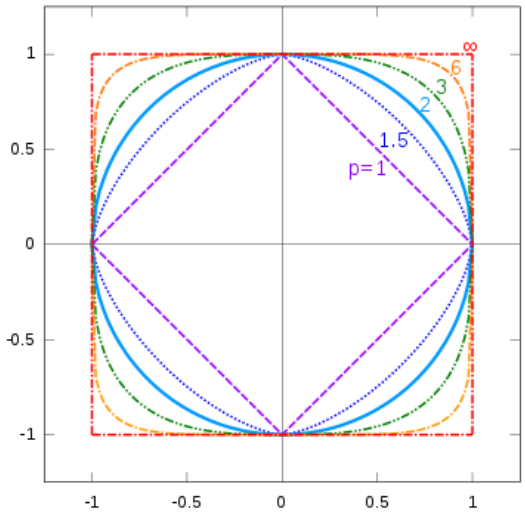
L(infinity) Metric

- Largest component over dimensions

$$d(u,v) = \lim_{p\to\infty} \sqrt[p]{(u_x - v_x)^p + (u_y - v_y)^p}$$

$$d(u,v) = \max(|u_x - v_x|, |u_y - v_y|)$$

It is the equal to max…why?
Raising the largest number to the p power, and then taking the p root will essentially just make the larger of the two terms the only significant one



illustrations of unit circles(see also superellipse) in different p-norms (every vector from the origin to the unit circle has a length of one, the length being calculated with length-formula of the corresponding p).

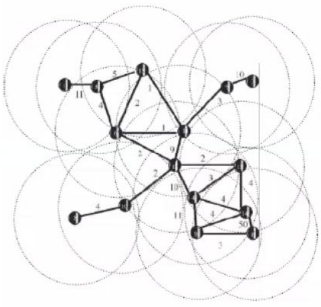
** **Note:** Things like the `undirected graph` is a form of a metric, too!

Graph Embedding

- Map the vertices V to points in the plane (or some other space)

- Different Distances
    - What's the distance between u and v in graph?
        - POSSIBILITIES:
            1. Embedding or extrinsic distance:
                a. Distance in the embedded space
                    i. E.g. Euclidean distance
            2. Intrinsic Distance
                a. Distance in the graph
                    i. Length of shortest path
            3. Intrinsic Distance
                a. Weighted distance in the graph
                    i. Weight of least weight path

Making networks from metrics



Technique 1: **Unit disk graph**
- Consider vertices in the plane (like wireless nodes)
- Connect 2 vertices by edge if they are within distance <= 1
- Applies generally to higher dimensions (unit ball graphs)
- Connect two nodes if they are within a given distance

Network Metric: Shape of the Data!
Technique 2: **k-NN Graphs**
- For each vertex, find k nearest neighbors
- Connect edges to all k nearest neighbors
- Variants:
    - Connect all k-NN edges
    - Connect only if both vertices are k-NN of each other.

Finding the distance between two nodes in a graph:
- BFS
- Dijkstra's
- Etc…

END [6]

[7] Growth, Expansion and Doubling Dim. of Metrics

Suppose your dataset doesn't have obvious distance yet, such as……..

Computing distances for **categorical data**

- E.g: We are given a list of clubs people belong to // songs they like // etc..

- Can define the **COSINE DISTANCE:**
  - Represent the list as 0-1 vectors A, B, …
  - Formulas:
    - Find cosine similarity $S_c = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2}$
    - Cosine distance $d_c = 1 - S_c$
  - Normalized dot product between two lists.
  - E.g. Two people like/don't like songs. Assign 0 or 1.
  - Given the similarity can define the cosine distance, as above ^^^
  - **Whenever you can define a similarity, you can define a metric by taking a suitable inverse!**

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

- Or we could define the **JACCARD SIMILARITY:**
  - Treat the vectors as sets A, B..
  - And compute $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$:
  - Distance $J_d = 1\text{-}J$
  - E.g: Set A contains songs person 1 likes. Set B contains songs person 2 likes. The Jaccard similarity is the ratio of songs they both like over the union of both sets (total options).
  - Again, can convert into distance by taking suitable inverse!

- or the **MIN CATEGORY DISTANCE**

- Take the size of the smallest club with both A and B as the distance
- Intuition: If two people belong to a huge club (football club), the membership doesn't signify any particular similarity between two people. Now, if two people are members of a small club (chess club), it signifies a greater similarity.

- ……..or make up your own way!

BALL

- A **ball** of radius r at point v:
  - The set of **all points** within distance r from v
  - Called a disk in 2D
- Usually written as B(v,r) or $B_r(v)$
- **Sphere** $S_r(v)$: set of points at distance **exactly** r from v
  - The boundary of the ball
  - 1-D sphere: boundary of a 2D ball
  - 2-D sphere: boundary of a 3D ball, etc…..
- The "measure" in a suitable dimension
  - Area in 2D
  - Volume in 3D
  - Length in 1D
- **GROWTH** of a metric
  - How does the size of a ball B(v,r) grow with radius?
    - 1D: O(r)
    - 2D: O(r^2)
    - 3D: O(r^3) (...etc…)

** Growth can be used to detect dimension **

- E.g. Long strips

  - Growth is linear: O(r)
  - Compared to size, this is 1-D

MORE GROWTH

Balanced Binary Tree (with height **h**)

- There are 2^h - 1 nodes in a tree. This is how it grows

Diameter of graph:

- **Largest distance between two farthest nodes in given graph.**
- *Put another way:* The **diameter of a graph** is the maximum length of the shortest path between a pair of nodes.

Sphere is number of nodes at exactly r.
2D grid. Ball grows as r^2. Sphere grows as O(r)

Doubling Dimension

"The notion people use to define dimension of a metric space"

- A metric space X is said to have constant doubling dimension if:
  - Any ball Bx(v,r) can be covered by:
    - At most M balls of radius r/2
    - For some constant M
  - **Lg(M) is called the doubling dimension of X**

- *Allows for measure of dimension in arbitrary structures, like graphs*
- This definition applies to any metric space.

Expanders / Edge Expansion

Informally, a graph is a good expander if it has low degree and high expansion parameters.

IDEA: How fast the "boundary" expands relative to "volume" or "size" of a subset.

The expansion of a graph is the minimum "surface-to-volume" ratio of any set of nodes. More precisely, if we use |S| to denote the size of a set of nodes S, use S_bar to denote the complement of a set of nodes S, and use $e_{out}(S)$ to denote the set of edges with exactly one end in S. Expansion can then be defined as:

- Expansion:

$$\alpha = \min_{S \subseteq V} \frac{|e^{out}(S)|}{\min(|S|, |\overline{S}|)}$$

*So we look at the number of edges **crossing a cut from S to S_bar**, and we compare it to the size of the smaller side. The **worst such** bottleneck in the graph is the expansion.*

Expanders (formally) are the class of graphs with an expansion at least a constant **alpha >= c**

Random **d-regular** graph for d>3

- Example of Expander
- All nodes have degree d
- Nodes are connected at random
- Constructed with "half-edges" that are then joined

For each $d \geq 3$, there is a constant α depending only on d, such that a random d-regular graph (of any size) has expansion at least α.

Properties of Expanders

- Expansion implies ***short paths***
    - Expanders implies a “robustness” to the graph.
- **THEOREM:** Expanders have **small diameter.**
    - A graph with degrees <= d and expansion >= a
        - Has diameter: O(d/a * lg n)
        - Has diameter == is *connected by a path of at most that length*

- Expanders are well-connected
- Usually sparse (number of edges much smaller than n^2)
- **Diffusion** process **spread fast** in an expander
- Random walks mix fast (achieve steady state)

END [7]

[8] Clustering and community detection

What are the “communities”
– Closely connected groups of nodes
– Relatively few edges to outside the community

Community Detection by Clustering

1) Define metric between nodes
    a) Intrinsic metric (eg: pairs of shortest paths...Floyd-warshall algorithm)
    b) Embed nodes in Euclidean space, use some metric there..
2) Apply A Clustering Algorithm

Given a set of items

- Define some distance between them
- Determine a grouping (partitioning) that optimizes *some* function

K-means clustering

Find k-clusters $\mathcal{C} = \{C_1, \ldots, C_k\}$

– With centers $\mathbf{c}_1, \ldots, \mathbf{c}_k,$

– That minimize the sum of squared distances of nodes to their cluster centers (called the k-means cost)

$$\Phi_{kmeans}(\mathcal{C}) = \sum_{j=1}^{k} \sum_{\mathbf{a}_i \in C_j} d^2(\mathbf{a}_i, \mathbf{c}_j)$$

Idea: minimize the sum of every node is as close to to cluster center with (k-clusters).

- Lloyd’s algorithm (heuristic)
    - Iterate until convergence
    - Selects k centers, and then recomputes cluster center as mean location of all elements in the cluster

- Ward’s Algorithm (also heuristic)
    - Start with each node as its own cluster, at each round MERGE two clusters, reducing the k-means cost the most.
    - Repeat until k-clusters

K means: discussion

- Tries to minimise squared sum of distances of items to cluster centers
  - NP-hard. Computationally intractable
  - Algorithm gives local optimum
- Depends on initialisation (starting set of centers)
  - Can give poor results
  - Submodular optimisation can help
- The right 'k' may be unknown
  - Possible strategy: try different possibilities and take the best
- Can be improved by heuristics like choosing centers carefully
  - E.g. choosing centers to be as far apart as possible: choose one, choose point farthest to it, choose point farthest to both (maximise min distance to existing set etc)…
  - Try multiple times and take best result..

K-medoids

- Similar to k-means, but now each center must be one of the given items.
- Useful when no ambient space (extrinsic metric)

Other center based methods

- K-center: Minimize maximum distance to center:

$$\Phi_{kcenter}(\mathcal{C}) = \max_{j=1}^{k} \max_{\mathbf{a}_i \in C_j} d(\mathbf{a}_i, \mathbf{c}_j)$$

- K-median: Minimize sum of distances:

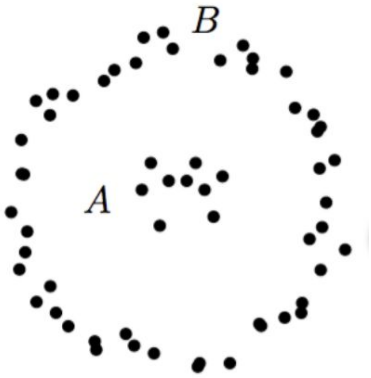
$$\Phi_{kmedian}(\mathcal{C}) = \sum_{j=1}^{k} \sum_{\mathbf{a}_i \in C_j} d(\mathbf{a}_i, \mathbf{c}_j)$$

Hierarchical clustering

- Top Down (divisive):
  - Start with everything in 1 cluster.
  - Make the best division, and repeat in each subcluster
- Bottom up (agglomerative):
  - Start with n different clusters
  - Merge two at a time by finding pairs that give the best improvement.

Density-based Clustering

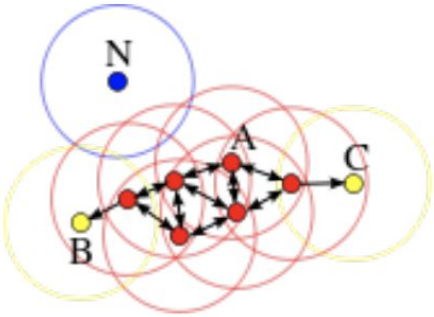
- Group dense regions together
- Better at NON-LINEAR separations
- Works with unknown numbers of cluster!



IDEA: Take unitless graph, connect nearby points, and get a network. The connected components are the clusters (the highly connected nodes!).

DBSCAN

- Density at a data point
  - Num of data points within radius Eps
- A core point
  - Point with density at least tau
- Border point
  - Density less than tau, but at least one core point within radius Eps
- Noise point
  - Neither core nor border; far from dense region



Algorithm:

- Construct undirected graph of core points
- Connected components of the graph give the cluster
- Assign border points to suitable cluster (eg: cluster to which it had most edges)

Dbscan discussion

- Useful in cases where it is clear which objects can be considered similar but **the number of clusters is not known**
- Works well in the real world!!!
- Useful case example:
    - A number of songs “seem” similar, but you have no idea how many groups we can divide everything into.

Dbscan problems

- Worst case: O(n^2)
- Requires knowledge of some suitable radius and density params (Eps and tau)
- Does not allow for possibility that different clusters may have different densities
    - Populations of Edi, Glasgow, and highlands would break it.

Other density based clustering algos

- Single linkage (Kruskal’s MST!!!)
    - Start with n clusters
    - Merge two clusters with the shortest bridging link
    - Repeat until k clusters

Communities

Other applications:

- Coarser representation of networks
- Meta-nodes for each communities
- Identify bridges/weak-links
- Structural holes (when bridges are absent)

Community detection (a simple strategy):

1. Choose a suitable distance measure based on available data
    a. (path lengths, distance based on inverse tie strength, etc.…)
2. Apply a standard clustering algorithm

Although clustering is not always suitable because……

- Small world networks have **small diameters**
- **High degree nodes** are common
    - Connect different communities
    - Hard to separate communities
- Edge **densities vary** across the network
    - Same threshold does not work well everywhere

Note: Finding dense subgraphs is HARD IN GENERAL

- Finding largest clique

- NP-hard
- Decision version is also np-complete

- We will look for approx :) :)

Dense subgraphs: Few preliminary definitions

- For S, T subgraphs of V
- e(S,T): Set of edges from S to T
  - e(S) = e(S,S): Edges within S
- $d_S(v)$: number of edges from v to S
- Edge density of S : |e(S)|/|S|
  - Largest for complete graphs or cliques

Dense subgraph problem

- Find the subgraph with largest edge density
- There also exists a decision version:
  - Is there a subgraph with edge density > alpha
  - Can be solved with max-flow algos
- Other versions: Find subgraphs size k or less

Variant: find densest S containing the given subset X
IDEA: knowing a few people, we can find the community they reside in

Example:

- Finding other STN students when you know there are four students already (X)

Q4. *How fast can the edge density of a subset S ⊂ V grow?*
Suppose we use notion n = |V | and x = |S|. Answer.
A4: *The number of edges in S can be as large as Θ(x^2), so the density can be Θ(x).*
Since x can be as large as Θ(n), the edge density can grow as Θ(n)

Betweenness and Graph Partitioning

IDEA: Identify the bridges and remove them.

- Bridges are central to the network.
  - They lie on the shortest paths!
  -
- Betweenness of edge(e)
  - Send one unit of traffic between every pair of nodes in the network!

- Measure what fraction passes through e (assume flow is split equally).
- IDEA: Bridges will carry a lot more! === Shortest paths!
- **"How many shortest paths pass through each edge ==== BETWEENNESS"**

Partitioning

- Algorithm: Girvan-newman algorithm
  - Find edge e of highest betweenness. Remove e
  - Produces hierarchical partitioning structure as the graph decomposes into smaller components.

Modularity

*The idea of the **modularity measure** is to capture **how many more** edges a partitioning explains **beyond what could be predicted merely from the degree distribution**. To capture what we mean by "from the degree distribution", we look at how many more edges are inside communities than would be in a random graph with the same degree distribution.*

Networks with high modularity have dense connections between the nodes within modules but sparse connections between nodes in different modules

- What is the "right" cut in e hierarchic clustering that represents good communities?
- IDEA: Maximize a quantity called modularity!
- Modularity of subset S:
  - Given a graph G

Modularity of subset S

- Given graph G
- Consider a random G' graph with same node degrees (remember configuration model)
  - Number of edges in S in G: $|e(S)|_G$
  - Expected number of edges in S in G': $E[|e(S)|_{G'}]$
  - Modularity of S: $|e(S)|_G - E[|e(S)|_{G'}]$
  - More coherent communities have more edges inside than would be expected in a random graph with same degrees
  - Note: modularity can be negative

IDEA: Modularity lets you evaluate how a good a partitioning is. Good split = modularity is high!

There is also modularity based clustering, but modularity is better when it's just used as a measure of quality. (eg. Louvain method).

$$q(\mathcal{P}) = \frac{1}{m}\sum_i |e(S_i)|_G - \frac{1}{4m}d(s_i)^2$$

Example!
For a complete graph, (I think) the modularity is 0. Using the configuration model, the only way to construct a random graph that mimics the complete graph, is to make a complete graph.

Correlation Clustering

- Some edges known to be similar (mark +) , and some dissimilar (mark -)
- Either max(num + edges inside clusters) OR min(- edges inside cluster)
- Applications:
    - Community detection based on similar people/users
    - Document clustering based on known similarity or dissimilarity between docs
- Features:
    - Clustering WITHOUT needing to know num. Clusters
    - Actually, Doesn't need *any* params!

*******NEW *****
Local detection of communities

- We often want to find smaller community that contains particular node or group
- Local Method like DBSCAN may work

Conductance: Measure of edges inside community vs outside

- Communities are likely to have **low conductance**

Personalised pagerank

- Given a seed set X
- Find the community S that contains X
- Pagerank style: Use random walks
- Algorithm
  - Set a limit k to number of steps in random walks
  - Repeat:
    - Select at random a start point from X
    - Take k random steps in the graph
  - Count how frequently each node occurs – pagerank
  - Nodes in the community have high pagerank

END [8]

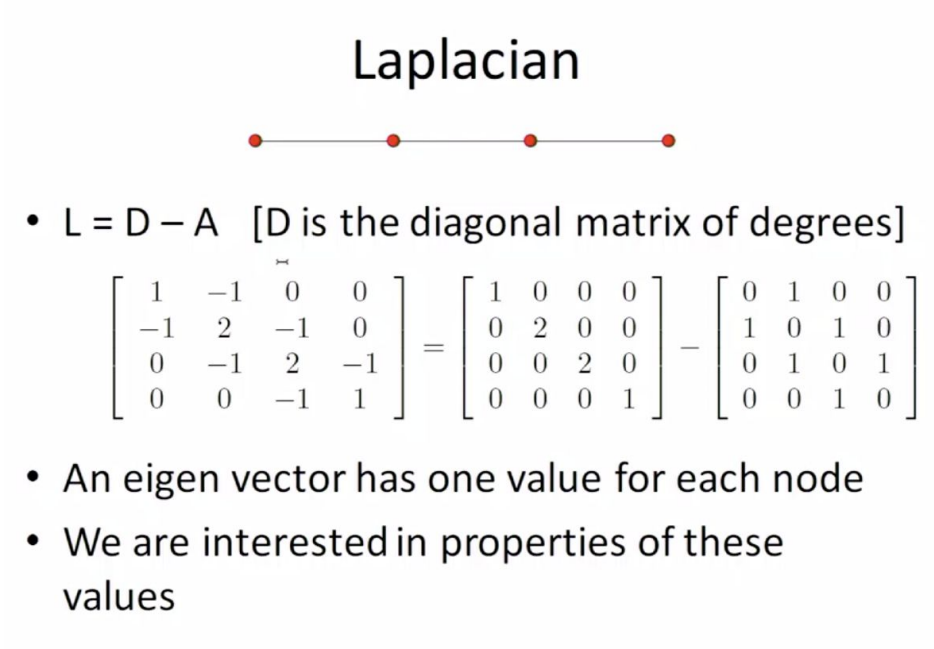
[9] Spectral Methods

Spectral: Understanding a graph using eigenvalues and eigenvectors of the matrix.

We already saw:

- Ranks of web pages: components of 1st eigenvector of suitable matrix
- Pagerank or HITS are algorithms designed to compute the eigenvectors
- Random walks and local pageranks help in understanding community structure

The **LAPLACIAN:**



- Eigenvector is going to be a vector of size n, with one value for each node in the graph.
- Matrix is symmetric => real eigenvalues.
- Positive semidefinite => Non-negative eigenvalues.

Laplacians and random walks

“Laplacians are a good representation of random walks.” “How does this change with time”

- Suppose we are doing a random walk on a graph.
- Le u(i) be the probability of the walk being at node i
    - E.g Initially it is at starting node s
    - After 10 steps, probability is higher near s, low at nodes far away.
    - Question: How does the probability change with time?
    - ***The probability diffuses with time. Like heat diffuses.***

Laplacian Matrix Intuition

- Imagine a small and different quantity of heat at each node (say, in a metal mesh)
- We write a function u: u(i) = heat @ i
- This heat will spread through the mesh/graph
- Q: How much heat will each node have after a small amount of time?

- Suppose nodes i and j are neighbors
  - **How much heat will flow from i to j?**
  - Depends on the difference u(i) and u(j)

- Proportional to the gradient: $(u(i) - u(j))*\Delta t$
  - Let us keep Δt fixed, and write just (u(i) - u(j))
- this is signed: negative means heat flows into i

Intuition:

- If i has neighbors j1, j2….
- Then heat flowing out of i is:
  = (u(i) - u(j1)) + (u(i) - u(j2)) + (u(i) - u(j3)) + …
  = degree(i)*u(i) - u(j1) - u(j2) - u(j3) - ….
- Hence L = D - A

$$\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

***** Gives you the net heat flow out of nodes at each step!**

The heat equation

$$\frac{\partial u}{\partial t} = L(u)$$

- The net heat flow out of nodes in a time step
- The change in heat distribution in a small time step
  - The rate of change of heat distribution

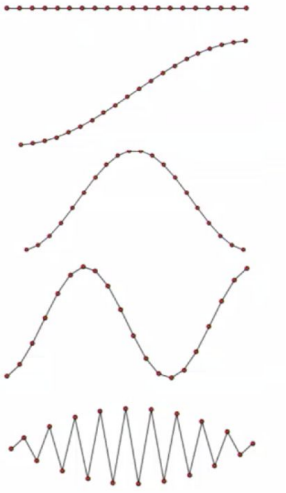
Heat Flow

- Will eventually converge to v[0]: the zeroth eigenvector, with eigenvalue $\lambda_0 = 0$.
- v[0] is a constant: no more flow!

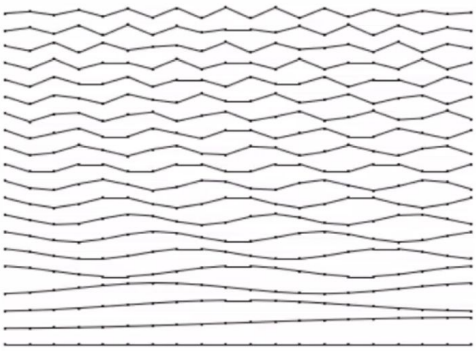
Eigenvalue Intuitions (the 1D case)

Observations

- j = 0
- j=1
- j=2
- j =3
- j = 19



- Low ones at bottom
- High ones at top
- Code on web page



Eigenvalue Observations

In Dim 1 Grid:

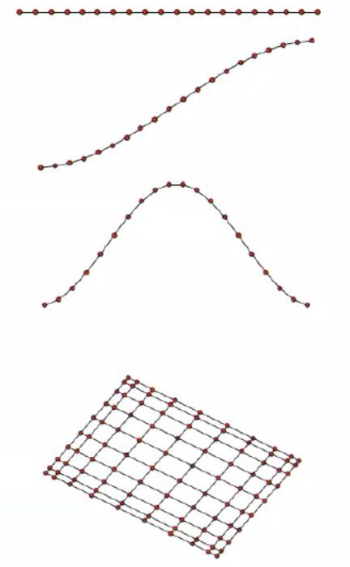
- V[1] is monotone
- V[2] is not monotone

In Dim 2 grid:

- Both v[1] and v[2] are monotone in suitable directions

For low values of j:

- Nearby nodes have similar values

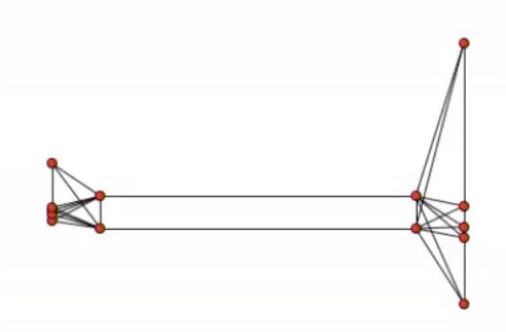


Applications

1. **Drawing a Graph (Embedding)**
    - Computer don't know how our graphs are supposed to look
    - We want to draw grids *nicely*
    - Using eigenvectors:
        - Suppose v[0], v[1], v[2] are eigenvectors
        - Plot graph using X=v[1] Y=v[2].

2. **Coloring**
    - Using the HIGH eigen-vectors for this one!
    - Assign colors to vertice, such that neighboring vertices do not have same color
    - E.g. Assignment of radio channels to wireless nodes. Good coloring reduces interference.
    - **IDEA: High eigen vectors give dissimilar values to nearby nodes.**

3. **Cuts/Segmentation/Clustering**
    - Find the smallest "cut"
    - A small set of edges whose removal disconnects the graph
    - Can also be used for clustering, community detection, etc…
    - Use v[1]. Tends to stretch out narrow connections: discriminates diff. Communities
    - Example Image Segmentation:
        - Assign a weight to each edge between pixels with a weight function that records difference.



4. **Isomorphism testing**
    - Eigenvalues are different implies graphs are different
    - Though not necessarily other way around

- Change implied by L on any input vector can be represented by sum of action of its eigenvectors (we saw this last time for M*M(transpose).
- V[0] is the slowest component of the change
    - With multiplier $\lambda_0 = 0$
- V[1] is slowest non-zero component
    - With multiplier λ_1

Spectral Gap

- **λ1 - λ0**
- Determines the overall speed of change
- If the slowest component v[1] changes fast
    - Then overall the values must be changing fast
    - Fast diffusion
- If the slowest component is slow
    - Convergence will be slow
- Examples:
    - Expanders have large spectral gaps
    - Grids and dumbbells have small gaps (~ 1/n)
- “Large” means some constant. “Small” means small in terms of size of graph.

Spectral Methods Conclusions

- Wide applicability inside and outside networks
- Related to many fundamental concepts
    - PCA/SVD
- Random walks, diffusion, heat equation…
- Results are good manny times, but not always
- Relatively hard to prove properties
- Inefficient eig. Computation costly on large matrix

END [9]

[10] Strong and weak ties

IDEA: Position of a node in a network determines its role/importance. Structure matters.

- Notation of strong ties (close friends) and weak ties (remote acquaintances)
    - How they influence network and spread of info
    - Eg. People more often find jobs through acquaintances (weak ties) than close friends (strong ties).
- "Weak ties are more critical: they can act as bridges across communities"

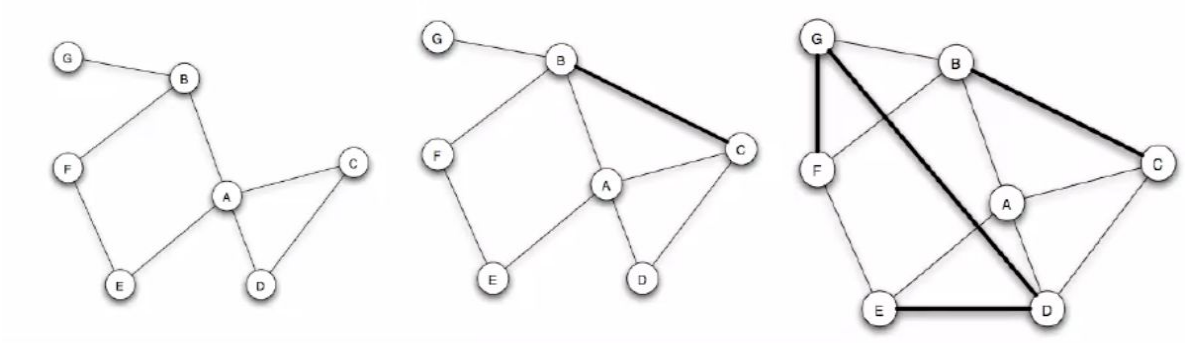
Homophily

- We are similar to our friends.
    - Not always explain by things intrinsic to the network like simple triadic closure
    - Could be external contexts like culture, hobbies, interests, influence networks

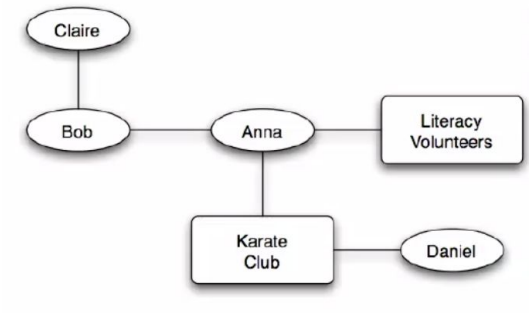
    - Suppose your network has 2 types of nodes (male/female), and fractions p and q.
        - Expected fraction of cross-gender edges: 2pq
        - A TEST for homophily:
            - Fraction of cross gender edges < 2pq
    - Example: "The obesity epidemic"

<u>Triadic Closure</u>: Friends of Friends

- If two people have a friend in common, they are more likely to become friends
- If B & C both know A:
    - They are likely to meet, may be for extended time
    - Likely to trust each other



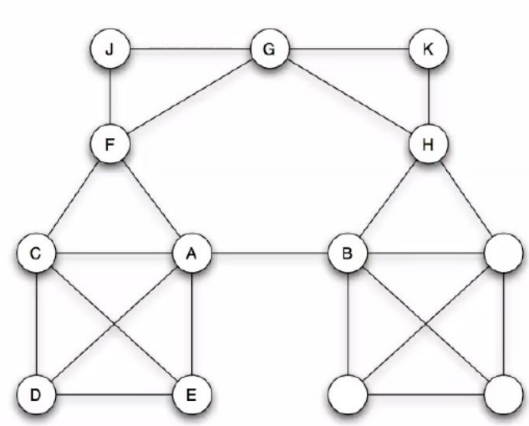
Affiliation Networks (showing Triadic Closure)



(i) Bob introduces Anna to Claire.
(ii) Karate introduces Anna to Daniel.
(iii) Anna introduces Bob to Karate.

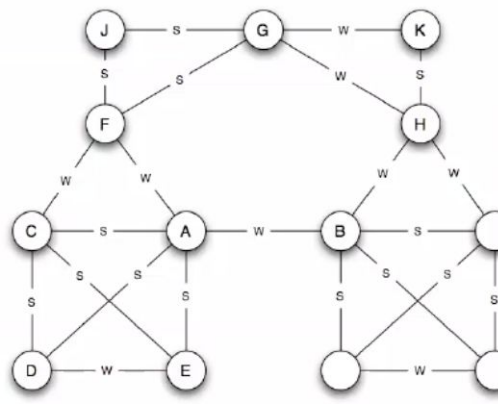
Bridges

- Remove a bridge will disconnect network
- Local bridge (A, B): If A, B have no friends in common
- Deleting (A,B) will increase distance to **d** > 2
- **d** is called the ***span*** of the bridge (A,B)



Strong Triadic Closure

- Suppose we know some ties to be strong, some to be weak
- STC: If **ab** and **bc** are strong, then edge **ac** exists (may be weak, but definitely there)
- **THM:** <u>If a network satisfies strong triadic closure and node A has >= 2 strong ties, then any bridge involving A must be a weak tie.</u>
- Absence of triadic closure generally implies poor relation between friends.
- TODO: Show proof of Strong Triadic Closure



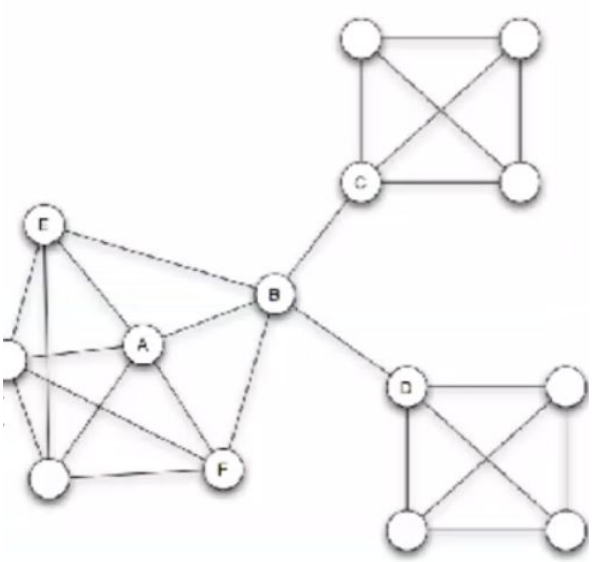
Neighborhood based estimate tie strength

- When we do not have a real observation for tie strength
- $N_r(p)$: *neighborhood of r hops centered at p*. Sometimes written as $B_r(p)$
  - $N(p) = N_1(p)$
  - Neighborhood overlap of ab (Jaccard Similarity):
    - A more continuous notion of strength
    - Derived from the network
  - Zero (or small, depending on definition of N) when ab is a local bridge

Embeddedness of an edge

- Number of common friends
- Higher embeddedness implies the more people monitoring the relation

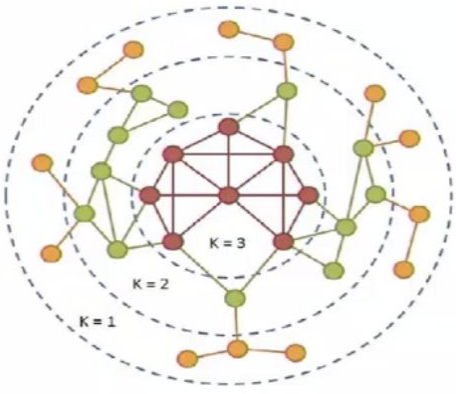
- B does not want to cheat A since E will no longer trust B
- B can sacrifice relation with C without losing any direct friend
- B is a part of a bridge that spans a gap in the network (a **structural hole)**



Social Capital ⇒ Ability to secure benefits by virtue of membership of network.

Centrality:

- Bridges are central to the network
  - They lie on the shortest path (betweenness)
- Other measures:
  - Degree centrality
  - Pagerank
  - Eigenvector centrality
- Closeness centrality (avg. distance to all nodes)
- K-core of a graph G
  - A maximal connected subgraph where each vertex has a degree at least k
  - Obtained by repeatedly deleting vertices of degree less than k



END [10]

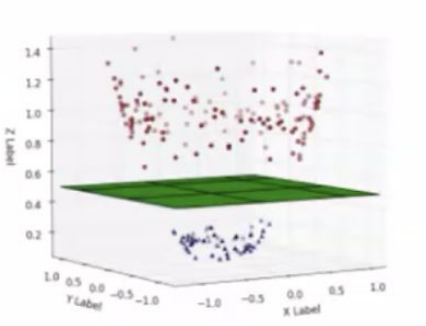
[11] Graph Kernels

- Kernels are a type of measure of similarity
- Important techniques in ML
- Used to increase power of many techniques
- Can be defined on graphs
- USED TO:
    - Compare, classify, cluster many small graphs
        - Eg: molecules, neighbors of people.
- video[1]

<u>The main ML question:</u>
For classes that can separated by a line...that's easy! But what if a separation is more complex!

Lifting to higher dimensions

- Suppose we life every (x,y) to (x,y)- > (x,y,x^2+y^2)
- Can now separate the data.



Kernels

A similarity measure K: X*X → Real is a kernel if:

- There an an embedding Φ (usually to a higher dimension),
    - s.t: K(u,v) = <Φ(u), Φ(v)>
    - Where <,> represents inner product
- e.g

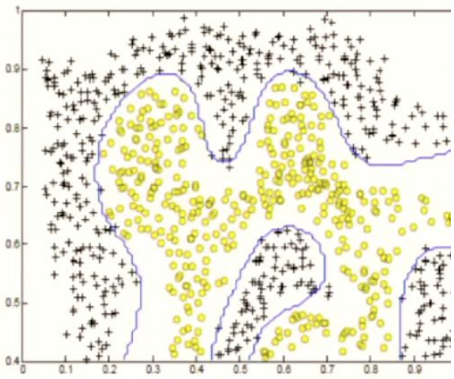
- $K(u, v) = (u \cdot v)^2$
  - This is true with lifting map $\psi(u) = (u_x^2, \sqrt{2}\, u_x u_v, u_y^2)$

Note: You don't need to do all the embedding...just can do this dot product.

[1] https://www.youtube.com/watch?v=mTyT-oHoivA

Other Kernels

- Polynnomial Kernel
- $K(u, v) = (1 + (u \cdot v))^k$
- Gaussian Kernel
- $K(u, v) = e^{-\frac{|u-v|^2}{2\sigma}}$
  - Sometimes called Radial Basis Function (RBF) kernel



Graph Kernel Applications

- To compute similarity between two attributed graphs.
  - Nodes can carry labels
  - E.g: Elements (C, N, H, etc..) in complex molecules
- **IDEA**: It is not obvious how to compare two graphs
  - Instead we compute walks, cycles,etc on graph and compare those...

Walk counting

- Count the number of walks of length k from i to j
- IDEA: i and j should be considered close if:
  - They aren't far in the shortest path distance
  - And there are many walks of short length between them (highly connected)
- So, there should be many walks of length <= k
- HOW:
  - Can be computed by taking the kth power of adjacency matrix A
    - If A^k(i, j) = c, that means there are c walks of length k between i and j,
    - NOte: A^k is expensive, but manageable for small graphs

Common walk kernel

- Count how many walks are common between the two graphs
- That is, take all possible walks of length k on both graphs.
  - Count the number that are exactly the same
  - Two walks are same if they follow the same sequence of labels

- Why is this a kernel?
    - We can argue that this is equal to taking some inner product in a higher dimension
    - The high dimensional space is:
        - Since we are taking all possible strings of length k in the space, this vector has each possibility.
        - Two vectors (for two graphs) can have a binary representation of whether string exists or not
        - Can take inner product between the two vectors
            - (where the resultant vector is true iff both vector A's value is 1 and vector B's value is 1).
        - In this case, we took high dimensionality (all string), and computed a dot product!

Random walk kernel

- Since common walk kernel is still kinda expensive, people use random walks.
    - Performs multiple random walks of length k on both graphs
    - Count the number of walks common to both graphs
    - This is usually good when you take walks with short k
- **Tottering**
    - Walks can move back and forth between adjacent vertices
        - Small structural similarities can produce a large score
    - Usual technique: for a walk v_1, v_2, …. prohibit return along an edge ie $v_i = v_{i+2}$
    - Prohibit immediate backtracking
    - "Kinda a hacky way of solving it"

Subtree Kernel

- From each node, compute a neighborhood upto distance h
- From every pair of nodes in two graphs, compare the neighborhoods
    - And count the number of matches

Shortest Path Kernel

- Compute all pairs of shortest paths in two graphs
- Compute the number of common sequences
- Tottering doesn't occur
- Problem:
    - There can be many (exp many) shortest paths between two nodes
        - Computational problems
        - Can bias the similarity
- Instead, just use the **shortest <u>distance</u> kernel**
    - Always unique
    - Method:
        - Compute all shortest distances SD(G1) and SD(G2) in graphs G1 and G2
        - Define kernel (e.g. Gaussian kernel) over pairs of distances: $k(s_1, s_2)$, where $s_1 \in SD(G_1), s_2 \in SD(G_2)$
        - Define shortest path (SP)kernel between graphs as sum of kernel values over all pairs of distances between two graphs
            - $\mathrm{K}_{SP}(G_1, G_2) = \sum_{s_1} \sum_{s_2} k(s_1, s_2)$
    -

END [11]

[12] Power law networks

Keywords: Preferential attachment, rich-get-richer

Embeddedness:
To talk about the structure around A it is useful to introduce an additional definition. We define the embeddedness of an edge in a network to be the number of common neighbors the two endpoints have. Thus, for example, the A-B edge has an embeddedness of two, since A and B have the two common neighbors E and F. This definition relates to two notions from earlier in the chapter. First, the embeddedness of an edge is equal to the numerator in the ratio that defines the neighborhood overlap in Equation (3.1) from Section 3.3. Second, we observe that local bridges are precisely the edges that have an embeddedness of zero — since they were defined as those edges whose endpoints have no neighbors in common

Degree Distribution

- A way of characterizing networks
- More complex than single numbers
- Many standard networks are known to have “standard” degree distributions
- Gives ways to incorporate notions of “popularity” and understand them
- Asks for a histogram:
  - As a function of **k**, what fraction of pages in the network have **k** links?
  - E.g: For random graph:
    - Not Gaussian/Normal Distribution (since edges not independent)
    - Probability that a node has degree k is:
    - Given by a binomial distribution

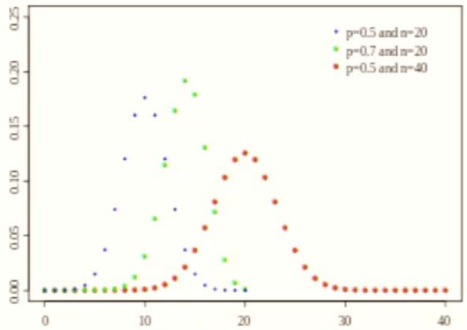
$$\binom{n-1}{k} p^k (1-p)^{n-1-k}$$

Possible sets of k edges

Probability that all k are chosen

Probability that others are not chosen

Degree distribution in a random graph



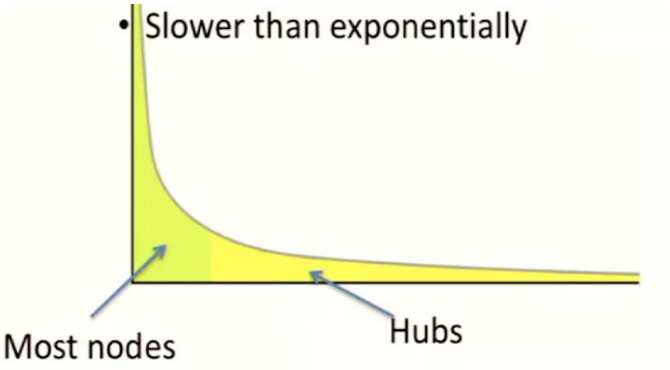
- Probabilities fall off really fast away from the peak
  - Exponentially fast with k
  - Very low and high degree are very very unlikely

This is the motivation...this distribution is maybe what we expect for the WWW, but that is not the case!

Degree distribution for the WWW : (1/k^2)

Power Law Networks

- Degree distribution of 1/k^a (for some constant a)
- What do power laws **mean?**
    - Most nodes have a low degree
    - There are several hubs with high degree
    - Heavy tail
    - Probability drops polynomially
    - slower than exponential
    - Take log/log plot for linear plot
        - But be careful - must extend quite a few order of magnitude



Hubs in power law networks

- Highly connected people/entities
- Critical in information dissemination
- Causes the network to have small diameter
- Examples:
    - Www, internet….
    - Social networks
    - Collaboration networks

Mean degree in a power law distribution

- The mean is finite iff a > 2
    - Assuming an infinite graph
- On the www, a is slightly larger than 2

Models of Power Law Networks

- We want a model that can be used to create power law networks
- Preferably one that mimics creation of actual power law networks like www
    - Gives us some idea of how these networks were created

<u>Preferential Attachment</u> ("Rich gets Richers") Mechanism

- IDEA: Older and established (popular sites) are likely to have more links to them (Google, Yahoo, etc.…)
- So how about, when a new page arrives, it links to older pages in proportion to their popularity
- When a new link is created on a new page, randomly link to older pages with probability of hitting a page x proportional to current popularity of x (number of links to x).

Preferential Attachment ("Rich gets Richer") Model

- Take a parameter **p** in [0,1]
- On a new page, create **k** links as follows:
    - When creating a new link:
        - With probability **p**:
            - Assign it with preferential attachment mechanism
        - With probability **1-p**:
            - Assign it with uniform random probability to any existing page.
- Takes into consideration that popularity is not the only force by link creation.
- Check kempe notes for proof
- Produces same exponent as www for **p ~= 0.9**
- *<u>Preferential attachment/power law are often a signature of artificial selection and popularity</u>*
- This isn't a perfect model, since it models acyclic data, and www can obviously link to future data.

Other reasons for Power Law

- Optimization:
    - Power law found in linguistics (zipf's)
- Random Processes:
    - Press space with probability p, else press a random letter key
    - This will produce a power law distribution of word lengths

End [12]

[13] Small world networks

Milgram Experiment - demonstrates small world (six degrees of separation) phenomenon
Idea:

- Shows that short paths exist between pairs (small diameter)
    - More surprisingly, people FIND these short paths w/o global network knowledge
- Idea of *Decentralized Search*
    - Analogous to routing without to routing table

Definition: <u>Small Worlds</u>

- Small diameter (O lg(n) -or- poly(lg(n)))
- Large clustering coefficient
    - Related to homophily - similar people connect to each other
    - “Similar” close in some coordinate value (or other metric)
- Supports decentralized search
    - People can find short paths *without* knowing the entire network
- (usually) has High Expansion

<u>SMALL WORLD MODELS</u>

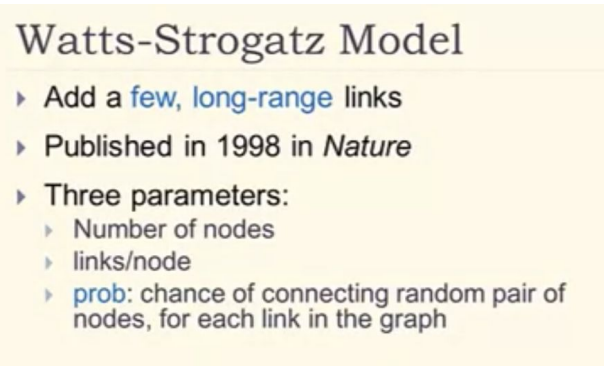
== Watts Strogatz Model ==

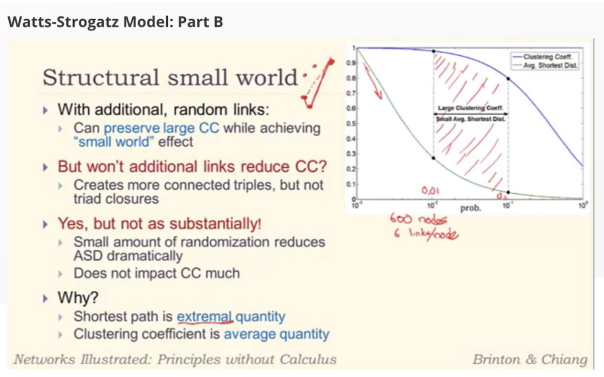
Parameters: **n, k, p** where => **n > k > ln n**

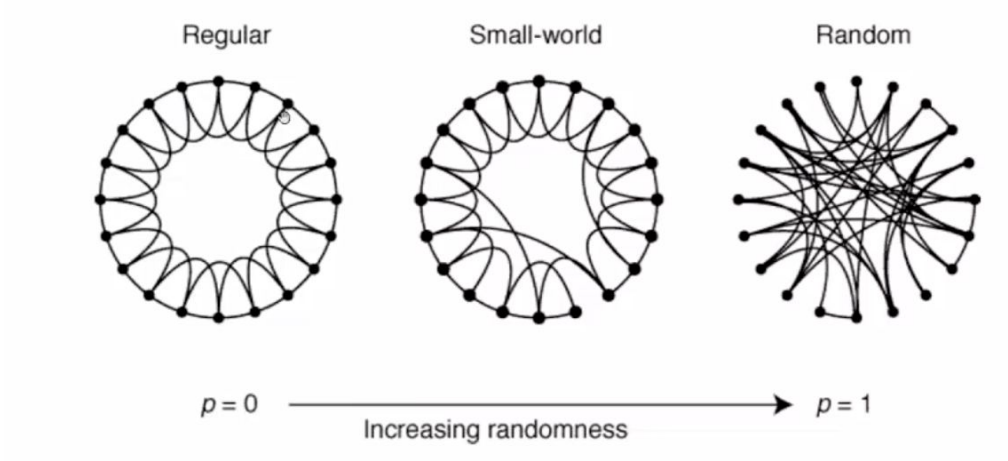
- Often k is taken to be a constant in practice with the idea that people cannot have infinitely large friend-circles

- Put nodes in a ring of size n
- Connect each to k/2 neighbors on each side
- Diameter: proportional to size of graph, CC is something high

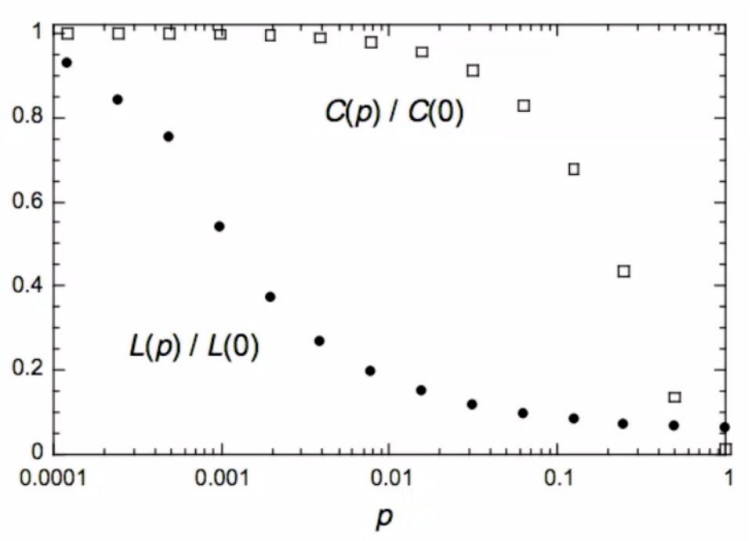
- <u>Create small world by:</u>
    - With probability p, rewire each edge of a vertex to a random vertex
    - Very quickly **reduces diameter** (jumping half way across basically halfs diameter.)







Adding a few random edges drops the diameter/shortest paths really quickly. We still maintain the high clustering coefficient for quite a while



The in-between phase is what we are interested in!

Watts-Strogatz model does NOT explain milgram's experiment because….

- Milgram's experiment was on a 2D plane
- Watts-strogatz does not support decentralized search poly(log(n)) steps to destination

- *A GOOD small world model should support poly(log(n)) steps to destination.*
- Following proof will show that watts-strogatz needs poly(n)
- Later, Kleinberg's model supports (log(n))^2 step route
    - Show that the route can be divided into O(log(n)) phases
        - Where each phase takes O(log(n))

PROOF
(TODO: check slides and book for better explanation)

Decentralized Search in random link networks

- Decentralized search does not work produce short path
- Consider 2D (NxN) grid.
- IDEA:
    - We want to show that if every node works only on its local information (its edges)
    - Then there is no algorithm that delivers the message in less than poly(n) hops.
- Proof:
    - Consider s and t separated by omega(n) hops
    - Take ball B of extrinsic radius around $n^{2/3}$ t
        - That means there's $O(n^{2/3})^2$ nodes in B
    - When we are already at distance $n^{2/3}$ (on the edge of B)
        - A long link can help us only if it falls inside B
    - Otherwise we can take a step along a short link
    - What is the probability that a random link from s hits b?
        - ~ $O((n^{2/3})^2 / n^2) = O(n^{-2/3})$
    - This is the expected number of steps before getting a useful long link:
        - Therefore at least omega($n^{2/3}$)
- Therefore, long links are not really useful in reaching t.
- **** Number of steps is poly(n)**

Model 2: Kleinberg's Model

IDEA: Long links are not helping much

- Getting closer to the destination does not increase the chances of getting a long link close to the destination.
- Make the probability of a long link sensitive to the distance
    - Nearby nodes are more likely to have a long link

- Suppose d(u,v) is the extrinsic distance between nodes u and v in the plane
- Then u connects its long link to v
- Connects with Probability inversely proportional to this distance
- <u>As a result:</u>
    - **Links to nearby nodes are more likely**
        - A node knows more people locally
        - With increasing distance, it knows fewer and fewer people
        - At the largest scale, knows only a handful
        - More representative of how people have their contacts spread
    - We want to show that the model permits short paths to be found.

TODO: Understand proof better

- PROOF:
    - Sketch of proof:
        - Takes rings of thickness 1 at distances 1,2,3…
        - ...
    - Basic Idea:
        - In O(log(n)) steps, the extrinsic distance is halved
            - Let us call this one “phase”
        - In O(log(n)) phases, this distance will be 1
        - So, we need to show the first claim: one phase lasts O(log(n)) steps.
    - “One Phase lasts log(n) steps:
        - Suppose distance from s to t is d
        - Take ball B of radius d/2 around t
        - There are about Theta(d^2) nodes in this area
        - The probability that a long link hits B is:

$$\frac{1}{\Theta(\log n)} \sum_{v \in B} d(s,v)^{-2} \geq \Theta\left(\frac{1}{\log n} d^2 d^{-2}\right) = \Theta\left(\frac{1}{\log n}\right)$$

        -
        - Thus, expected number of steps before we find a link into B is log(n).
        - In total process, there are log(n) such phases, thus log^2(n) steps.

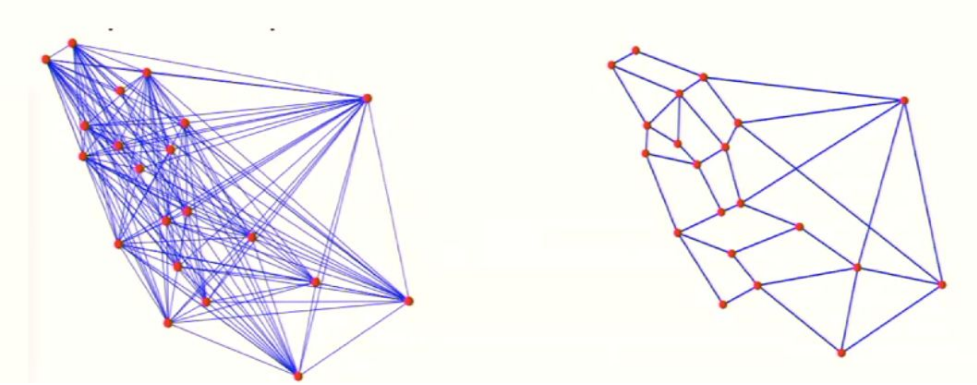
END [13]

[14] Spanners

Spanners are about distances in graphs

Suppose we are interest in finding distances, shortest paths, etc in a weighted graph G

- The problem:
  - A graph can have n^2 edges
  - Any computation is expensive
  - Storage is expensive
  - **IDEA: Use a "similar" graph with fewer edges**
    - A spanning graph H of a connected graph G:
      - H is connected and has same set of vertices
    - Construct an H with fewer edges



Stretch

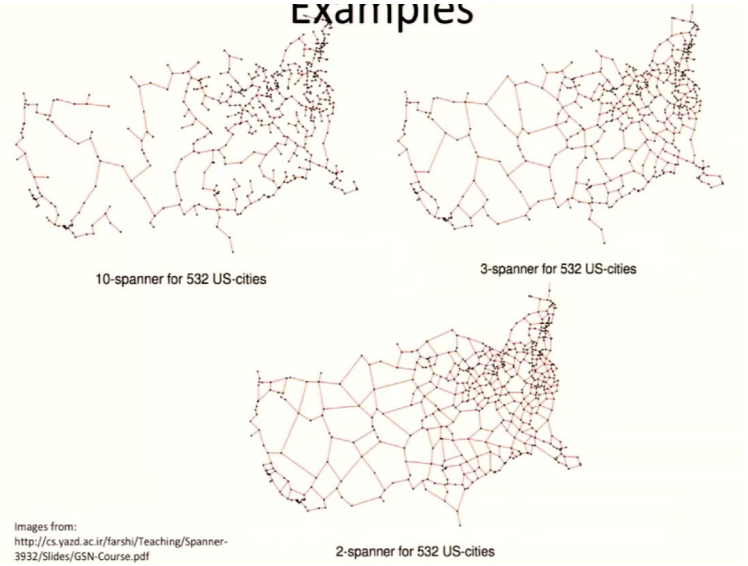
- Suppose d_G is the shortest path distance in G
- Suppose $d_h(u,v) = s * d_g(u,v)$
- S is called the stretch of distance between u and v
- **IDEA: have the compressed network (H) with a <u>small stretch</u> and few edges**

Spanners =====>

- Suppose d_G is the shortest path distance in G
- H is a t-spanner of G if:
- $d_H(u,v) \leq t \cdot d_G(u,v)$
  - A multiplicative spanner
  - The stretch of the spanner is t
- More generally, H is a (α, β)-spanner of G if:
- $d_H(u,v) \leq \alpha \cdot d_G(u,v) + \beta$

Spanner Examples

10-spanner: shortest path is allowed to be 10x the length of the original shortest path.



Trade-off: Lower stretch means you need to store more edges.

Examples:

- Compressing road maps and still finding good paths
- Compress computer/communication networks to get smaller routing tables
- "Bridges" are part of spanner
- Small set of distances among moving objects (eg. ROBOT COLLISIONS)
    - To detect possible collisions
    - A "short edge" must always be in the spanners
    - Thus, only need to check the spanner

Algorithms to generate spanners

Simple Greedy (for fixed examples)

This is basically kruskal's shortest path algo.

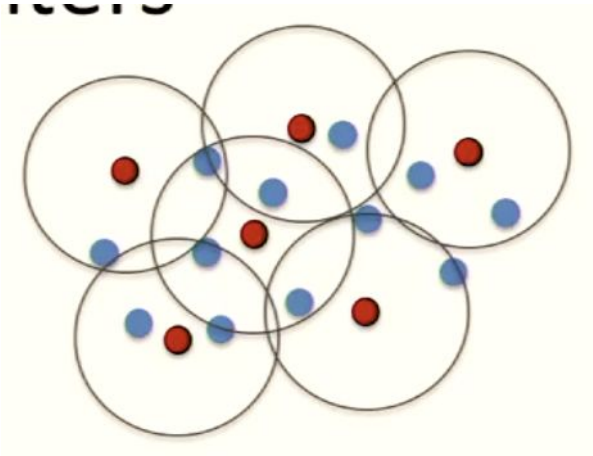
- Given graph G=(V, E) and stretch t
- We want to construct H=(V, E')
- Sort all edges in E by length
- Proceed from shortest to longest edge
  – Take edge e=(u,v)
  – If $d_H(u,v) > t \cdot d_G(u,v)$, add e to E'
- Output H

- Claim: $d_H(u,v) \leq t \cdot d_G(u,v)$
- If (u,v) is an edge in G, then this holds by construction.
- If not, suppose P is the path between them of length $d_G(u,v)$
- For each edge $(a,b) \in P$, the claim holds.
  – Therefore. It holds for the sum of their lengths.

[Thm] The greedily constructed t-spanner has **$O(n^{1+(2/t+1)})$ edges**

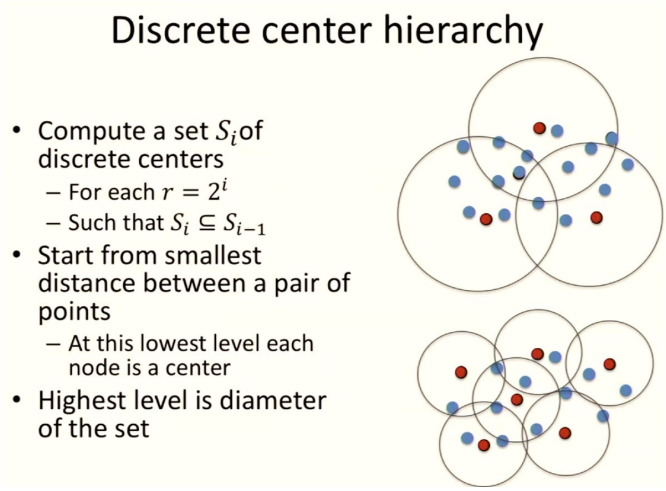
Deformable Spanners

- Suppose we have n points in R^d
- We want to compute a good spanner
  - With stretch **(1 + eps)**
  - Number of edges **n / eps^d**
  - Do this by computing **discrete centers**
    - Given radius r
    - A set S of discrete centers is a subset of V
    - S.t:
      - Any pt of V is within distance r of some s in S
      - Any two points s1, s2 in S are at least r apart
      - That is, a set of balls with far apart center, that covers all the points



Then, computes hierarchy of discrete centers

==========>



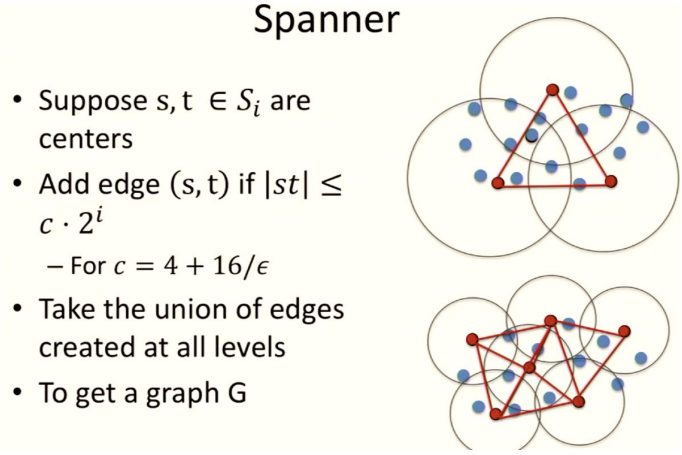
"Finding a set of points that is representative of the entire set"

Applying Discrete Centers to Spanners ==========>

"Like agglomerative clustering"

G is a (1+eps) spanner

- That is, for any two pts **p** and **q**, there is a path in G of length at most **(1+eps)|pq|**
- G has **n / eps^d** edges



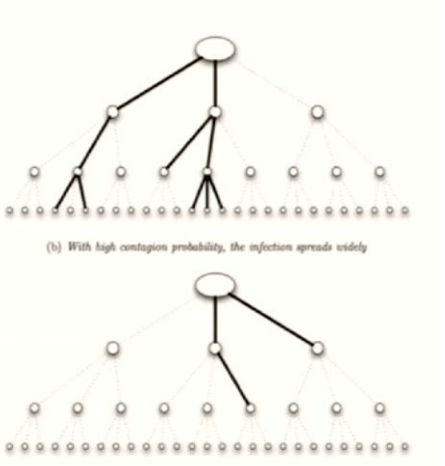
Useful properties of deformable spanners

- Applies to metrics of bounded doubling dimension
- Relatively small number of edges
- Each node has a small number of edges
    - Efficient in checking for collisions and near neighbors
    - Each robot has to keep small amount of information
- Can be updated easily as nodes enter, leave, etc..
    - That's why it is deformable
- *e.g) autonomous cars!*

END [14]

[15] Epidemics and Gossip

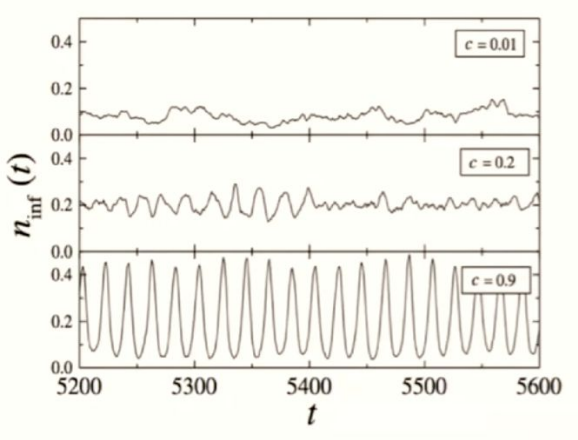
- Spread of disease
    - Pattern of spread depend on network **structure**
    - e.g spread of flu
    - Network of people
    - Network of airlines
    - Different from idea/innovation contagion
        - Does not need a “decision”
        - Does not need multiple support
            - Infectious disease passes easily with some probability


(b) With high contagion probability, the infection spreads widely

    - Suppose everyone meets k new people and infects each with probability **p**
    - That is, they infect **R = kp** people on average
    - If p is high, the disease will persist though several round
    - If p is low, it will die out after some rounds
    - Property:
        - When **R > 1**
            - Outbreak, infection spreads
        - When **R < 1**
            - Disease dies out
        - Phase transition at **R = 1**
            - **Small efforts have large effect on epidemic**
                - Awareness causing slight decrease in p
                - Quarantine/fear causing slight decrease in k

<u>Models</u>

- **(1) SIR Model**
    - Susceptible (initially)
    - Infectious (after being infected)
        - While infectious, it can pass disease to each neighbor in each step with prob. P
    - Removed (after given duration as infection)
        - dead/immune
- **(2) SIS Model**
    - No “removed” state
- **(3) SIRS Model**
    - Only immune for a little bit, then susceptible again (like the common flu)



SIRS oscillations in Watts-Strogatz small worlds:

- Nodes connected to few neighbors on a ring
- Fraction c of links modified to connect to random nodes
- *E.g. Flu season*

(lots infected → lots not infected → lots infected again)

Epidemic or Gossip Algorithm

- Emulates the spread of epidemic or a rumor in a network
- A node speaks to a random neighbor to spread the rumor message
- Useful for spreading information in computer networks
- **Spreading Message via gossip**
  - Complete Graph: Anyone can call anyone
  - PROBLEM: One node has a message or rumor to spread. How does it spread it to all nodes in the network.
  - Calling everyone will take O(n) round.
  - After round 1, you cooould use the nodes with rumor to help.
    - But how avoid collision?
    - STRATEGY:
      - In each round, anyone with the rumor **calls one random node** and passes the rumor.
    - THM: If you follow this random strategy, everyone gets the message in O(log(n)) rounds!

PROOF:

- n/3 nodes get the message in log n rounds
  - Current number of infected nodes m < n/3
  - Probability that a call goes to a new node is at least 2/3
    - Number of calls to new nodes: 2m/3
  - Probability of a collision at the new node is 1/(n-m)
  - $O(m^2)$ possible pairs for collision
  - Max possible collisions: $\frac{m^2}{2(n-m)} \leq \frac{m^2}{2} \cdot \frac{1}{2m} = \frac{m}{4}$
- Number of newly infected nodes at least $\frac{2m}{3} - \frac{m}{4} = \frac{5m}{12}$

- while m < n/3
  - m grows to m(1 + 5/12) = 17m/12 every round
  - m grows to n/3 in O(log n) rounds
- After m> n/3
  - Probability that a node is not called in 1 round is $(1 - \frac{1}{n})^{\frac{n}{3}} \leq e^{\frac{-1}{n} \cdot \frac{n}{3}} = e^{-1/3}$
  - Probability that 1 or more nodes are not called after O(log n) rounds
  - Less than $1/n^c$, where c depends on the constant in the O

Todo: See Kempe 18 for in depth proof

Basic idea:
When you first start, not many people are infected. This is good, because it means picking people at random to infect is generally very successful! Infected set grows really fast. In log(n) rounds, we will infect at least n/3 nodes due to this.
At this stage, you have ⅓ of the population trying to infect new nodes. Any one effort may not be successful, but since you have so many nodes working to infect others, people will not survive for too long. This second step can also be completed in log(n) time.

EXAMPLE Cases and Methods of GOSSIP:

- **In a computer network (imagine wireless network)**
- Spreading a piece of information
- Naive Technique: Flood
    - Nodes call all its neighbors to send message
    - Flood is wasteful: many nodes can have common nearby neighbors...wastes msg
    - Better to do it as one neighbor per round.
    - Still spreads slowly…
    - At least sqrt(n) rounds in a grid of n nodes
- Better technique:
    - Imagine nodes on a plane
    - Instead of only sending messages to neighbors
    - Sends to nodes at **random** (need some routing mechanism)
    - Easily reaches far away nodes
    - Faster:
        - O(log n) rounds
        - O(n log n) messages sent
    - But the routing costs more messages!
        - sqrt(n) hops if routing on a grid
        - (n*sqrt(n) log n TOTAL TRANSMISSIONS
- Best technique:
    - Spread via a small world distribution
        - Instead send message to a random node
            - **Fixes problem of sqrt(n) hops!**
            - Picking a node at distance d with probability 1/d^3
            - Short paths are more common in these distributions
            - Avg. msg. Cost ⇒ O(poly(log(n)))
            - Still there are enough long messages spreading the message to far away regions.

Advantages of Gossip

- Simple
- Robust algorithm
    - Node failure doesn't stop computation
    - Easy to add nodes
    - At the cost of a log factor of increased costs

Averaging Gossip

- Suppose all nodes have a "value"
- We will to compute a linear function of these values
- eg: The average.
- Algorithms:
    - The push-sum protocol
        - In every round:
            - Every node takes a fraction of its value, sends it to a random neighbor
            - Adds all received values to its current value.
    - The pairwise averaging protocol
        - In every round, node talks to one other random neighbor.
        - Both nodes set their values to the average of the two
        - Eventually, all are at the average. (process converges)
- Both protocols converge in O(log(n)) rounds on a complete graph
    - Could take a lot longer on a grids tho
    - On small world graphs
        - We don't know :O :O :O

END [15]