Graphs and Social Networks

Why Social Graphs Are Different
Communities
Community-Detection Methods

Jeffrey D. Ullman
Stanford University/Infolab

Social Graphs

- Graphs can be either directed or undirected.
- Example: The Facebook "friends" graph (undirected).
  - Nodes = people; edges between friends.
- Example: Twitter followers (directed).
  - Nodes = people; arcs from a person to one they follow.
- Example: Phonecalls (directed, but could be considered undirected as well).
  - Nodes = phone numbers; arc from caller to callee, or edge between both.

Properties of Social Graphs

1. *Locality* (edges are not randomly chosen, but tend to cluster in “communities”).
2. *Small-world property* (low diameter = maximum distance from any node to any other).

Locality

- A graph exhibits *locality* if when there is an edge from x to y and an edge from y to z, then the probability of an edge from x to z is higher than one would expect given the number of nodes and edges in the graph.
- Example: On Facebook, if y is friends with x and z, then there is a good chance x and z are friends.
- *Community* = set of nodes with an unusually high density of edges.

It's a Small World After All

- Many very large graphs have small *diameter* (maximum distance between two nodes).
  - Called the *small world* property.
- Example: 6 degrees of Kevin Bacon.
- Example: "Erdos numbers."
- Example: Most pairs of Web pages are within 12 links of one another.
  - But study at Google found pairs of pages whose shortest path has a length about a thousand.

Finding Communities

Betweenness
Cliques and Bi-Cliques
Laplacian Matrices
Association-Graph Model

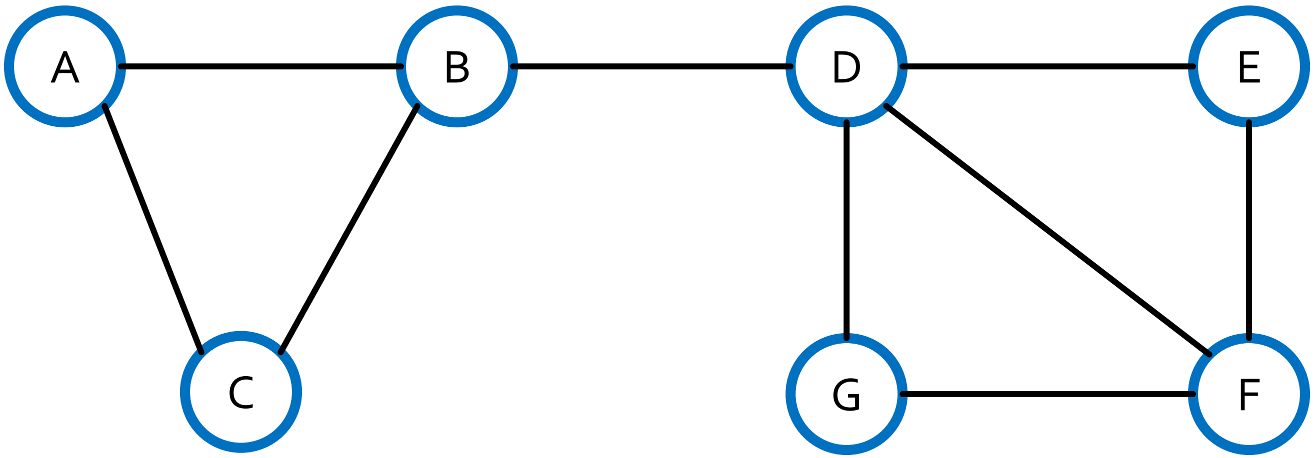
Two Approaches to Communities

1. Partition the graph into disjoint communities such that the number of edges that cross between two communities is minimized (subject to some constraints).
   - Question for thought: what would you do if you *only* wanted to minimize edges between communities?
2. Construct *overlapping communities*: a node can be in 0, 1, or more communities, but each community has many internal edges.

Betweenness

- Used to partition a graph into reasonable communities.
- Roughly: the betweenness of an edge e is the number of pairs of nodes (A,B) for which the edge e lies on the shortest path between A and B.
- More precisely: if there are several shortest paths between A and B, then e is credited with the fraction of those paths on which it appears.
- Edges of high betweenness separate communities.

Example: Betweenness



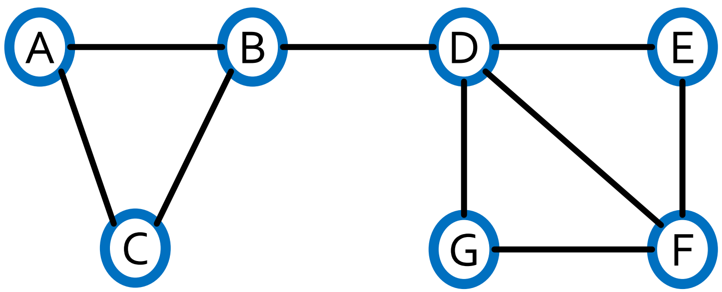
Edge (B,D) has betweenness = 12, since it is on the shortest path from each of {A,B,C} to each of {D,E,F,G}.

Edge (G,F) has betweenness = 1.5, since it is on no shortest path other than that for its endpoints and half the shortest paths between E and G.

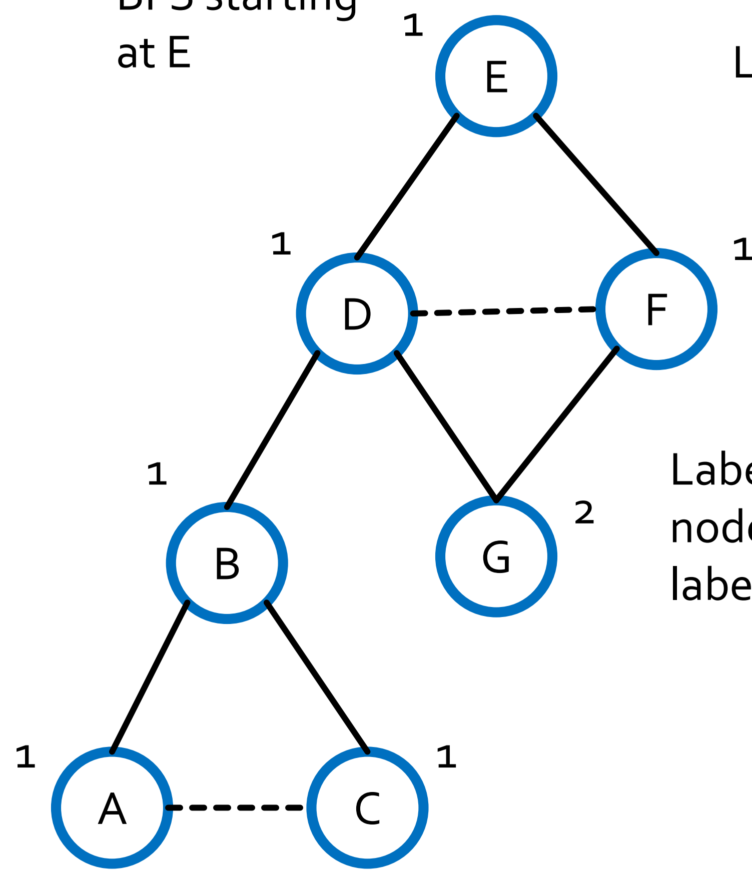
Girvan-Newman Algorithm

1. Perform a breadth-first search from each node of the graph.
2. Label nodes top-down (root to leaves) to count the shortest paths from the root to that node.
3. Label both nodes and edges bottom-up with sum, over all nodes N at or below, of the fraction of shortest paths from the root to N, passing through this node or edge.
4. The betweenness of an edge is half the sum of its labels, starting with each node as root.
   - Half to avoid double-counting each path.

Example: Steps 1 and 2



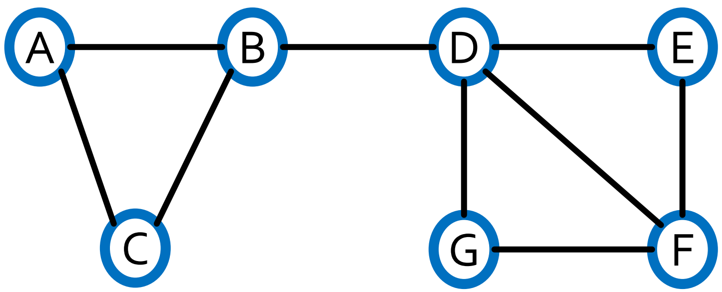
BFS starting at E

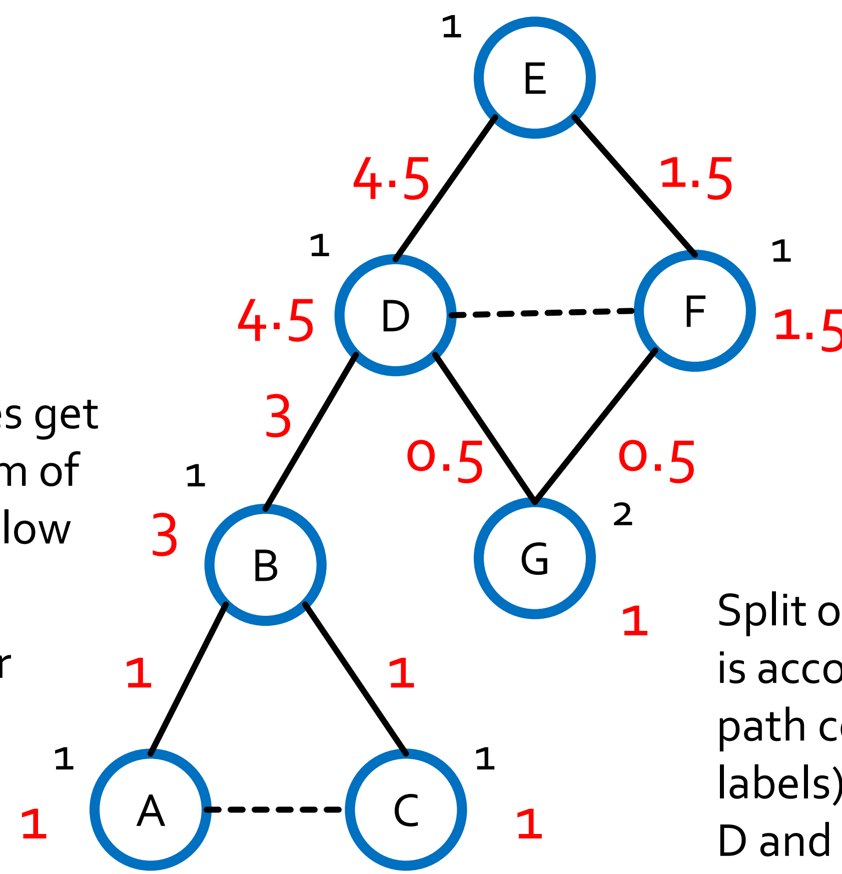


Label of root = 1

Label of other nodes = sum of labels of parents

Example: Step 3





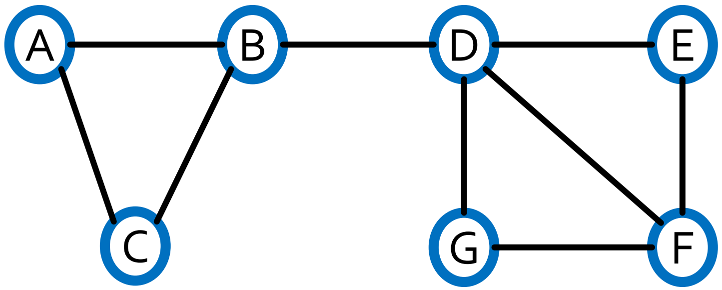
Interior nodes get 1 plus the sum of the edges below

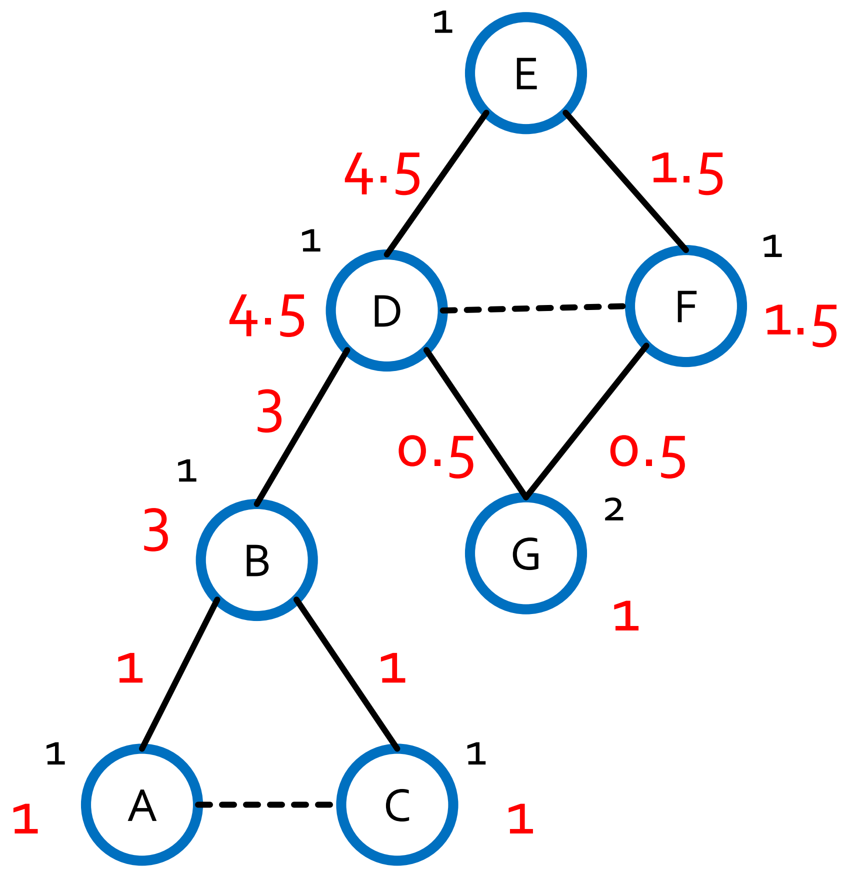
Edges get their share of their children

Split of G's label is according to the path counts (black labels) of its parents D and F.

Leaves get label 1

Sanity Check





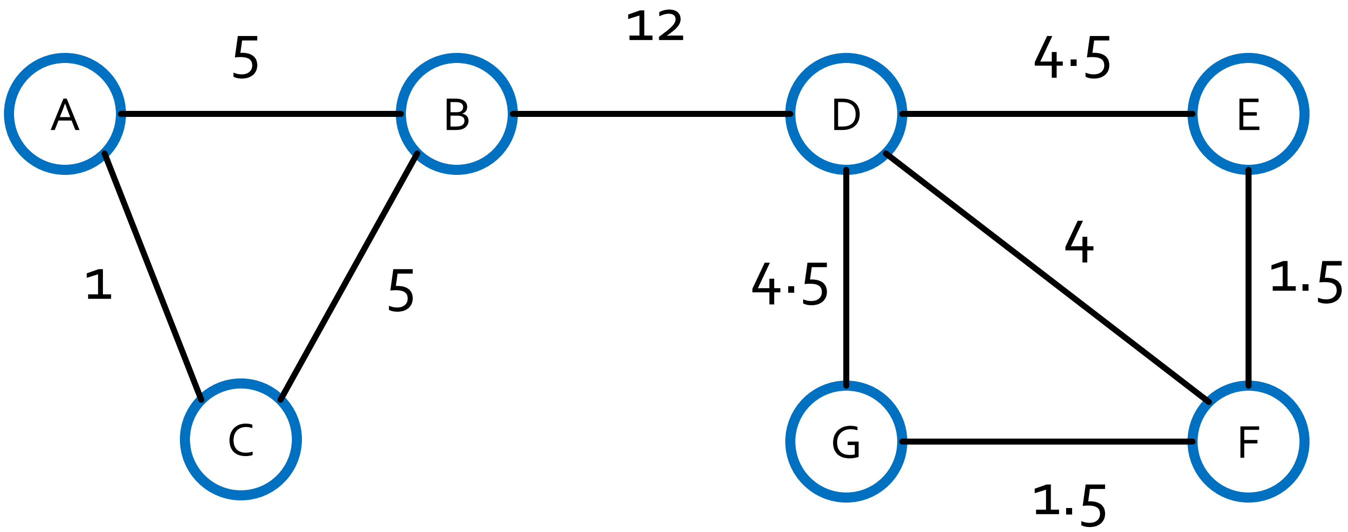
Edge (D,E) has label 4.5.

This edge is on all shortest paths from E to A, B, C, and D.

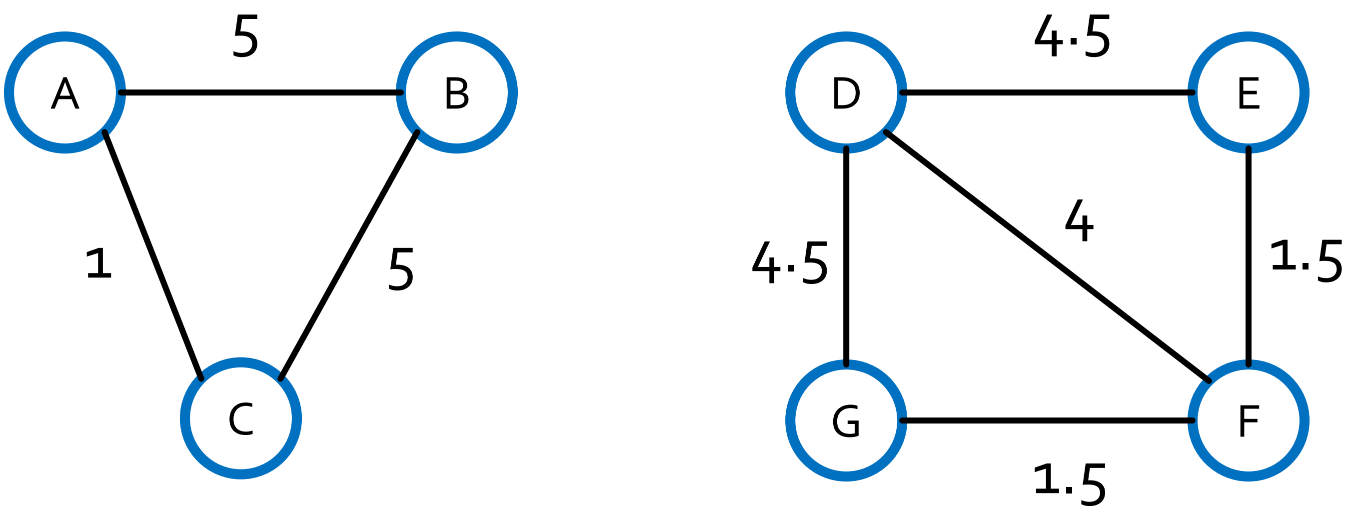
It is also on half the shortest paths from E to G.

But on none of the shortest paths from E to F.

Result of G-N Algorithm

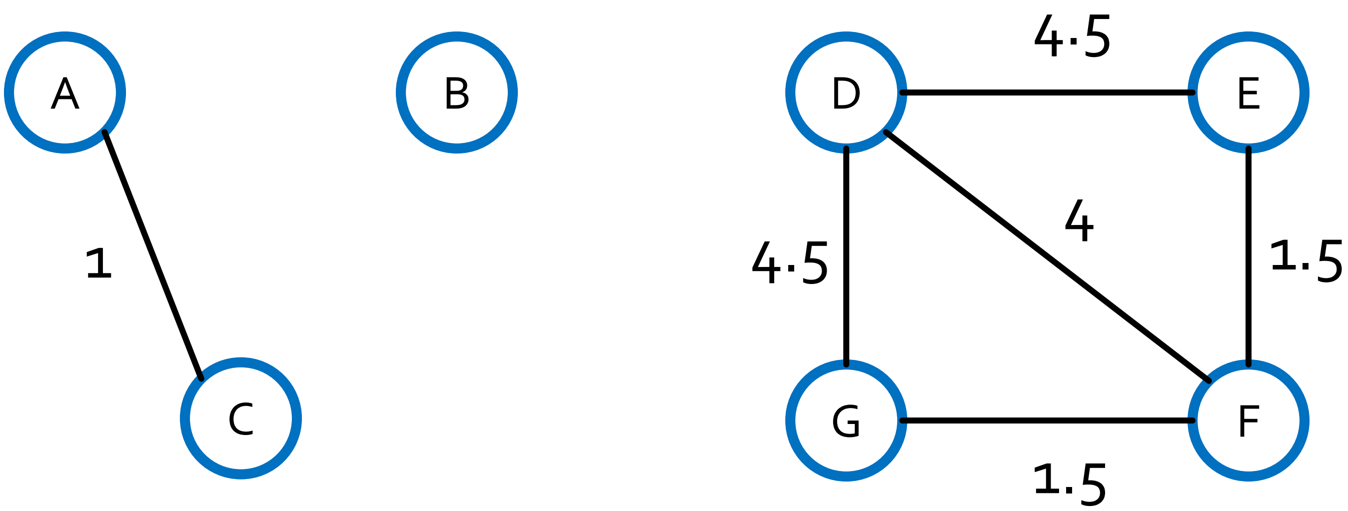


Remove Edge of Highest Betweenness



A sensible partition into communities

Remove Next-Highest Edge(s)



Why are A and C closer than B?
B is a "traitor" to the community,
being connected to D outside the group.

Paralllelizing G-N Algorithm

1. Algorithm can be done with each node as root, in parallel.
2. Depth of a breadth-first tree is no greater than the diameter of the graph.
3. One MapReduce round per level suffices for each part.

Communities Via Complete Bipartite Graphs

Growing Communities
Existence of Large Bi-Cliques

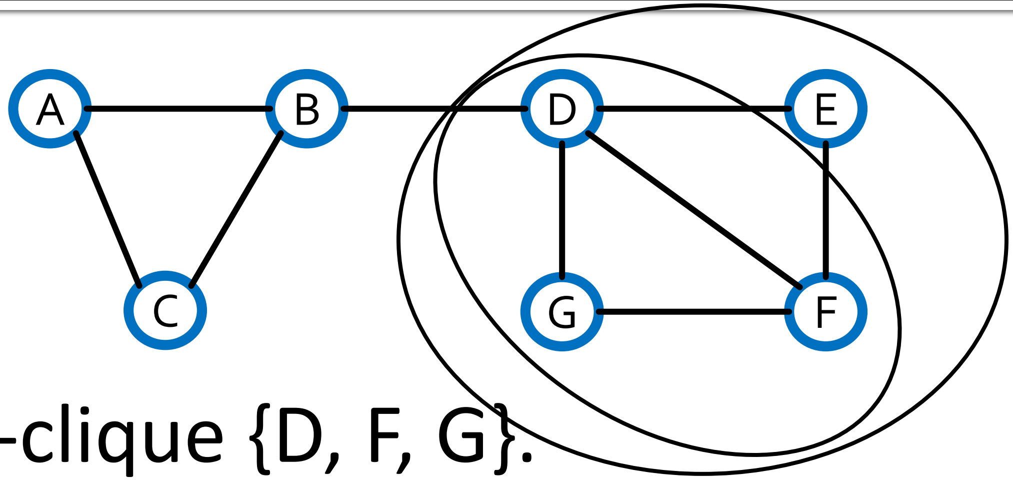
Growing Communities

- Recall a *community* in a social graph is a set of nodes that have an unusually high number of edges among those nodes.
- Example: A family (mom+dad+kids) might form a complete subgraph on Facebook.
  - In addition, more distant relations (e.g., cousins) might be connected to many if not all of the family members and frequently connected to each other.

Cliques

- One approach to finding communities is to start by finding *cliques* = sets of nodes that are fully connected.
- Grow a community from a clique by adding nodes that connect to many of the nodes chosen so far.
  - Prefer nodes that add more edges.
  - Keep the fraction of possible edges that are present suitably high.
  - May not yield a unique result.
  - May produce overlapping communities.

Example: Growing Communities



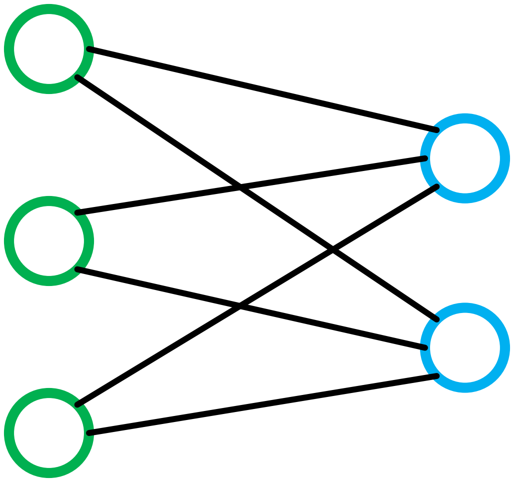
- Start with 3-clique {D, F, G}.
- Can add E, and the fraction of edges present becomes 5/6.
- Better than adding B to {D, F, G}, because that would result in an edge fraction of only 4/6.
- And adding B to {D, E, F, G} would give a fraction 6/10, perhaps too low.

Problems with Cliques

1. Finding largest cliques is highly intractable.
2. Large cliques may not exist, even if the graph is very *dense* (most pairs of nodes are connected by an edge).

- Strangely, a similar approach based on *bi-cliques* (two sets of nodes S and T with an edge from every member of S to every member of T) works.
  - We can grow a bi-clique by adding more nodes, just as we suggested for cliques.

Example: A Bi-Clique



Finding Bi-Cliques

- It's an application of "frequent itemsets."
- Think of the nodes on the left as "items" and the nodes on the right as "baskets."
- If we want bi-cliques with t nodes on the left and s nodes on the right, then look for itemsets of size t with support s.
- Note: We find frequent itemsets for the whole graph, but we'll argue that if there is a dense community, then the nodes of that community have a large bi-clique.

Finding Bi-Cliques –(2)

- Divide the nodes of the graph randomly into two equal-sized sets (“left” and “right”).
- For each node on the right, make a basket.
- For each node on the left make an item.
- The basket for node N contains the item for node M iff there is an edge between N and M.
- Key points: A large community is very likely to have about half its nodes on each side.
  - And there is a good chance it will have a fairly large bi-clique.
    - Question for thought: Why?

Dense Communities Have Big Bi-Cliques

- Suppose we have a community with 2n nodes, divided into left and right sides of size n.
- Suppose the average degree of a node within the community is 2d, so the average node has d edges connecting to the other side.
- Then a "basket" (right-side node) with d_i items generates about $\binom{d_i}{t}$ itemsets of size t.
- Minimum number of itemsets of size t is generated when all d_i's are the same and therefore = d.
- That number is $n\binom{d}{t}$.

Bi-Cliques Exist – (2)

- Total number of itemsets of size t is $\binom{n}{t}$.
- Average number of baskets per itemset is at least $n\binom{d}{t}/\binom{n}{t}$.
- Assume n > d >> t, and we can approximate the average by $n(d/n)^t$.
- At least one itemset of size t must appear in an average number of baskets, so there will be an itemset of size t with support s as long as $n(d/n)^t \geq s$.

Uses approximation x choose y is about $x^y/y!$ when x >> y.

Example: Bi-Cliques Exist

- Suppose there is a community of 200 nodes, which we divide into the two sides with $n = 100$ each.
- Suppose that within the community, half of all possible edges exist, so $d = 50$.
- Then there is a bi-clique with t nodes on the left and s nodes on the right as long as $100(1/2)^t \geq s$.
- For instance, (t, s) could be (2, 25), (3,13), or (4, 6).

Communities Via Laplacian Matrices

- Degree, Adjacency, and Laplacian Matrices
- Eigenvectors of Laplacian Matrices

The Laplacian Approach

- As with "betweenness" approach, we want to divide a social graph into communities with most edges contained within a community.
- A surprising technique involving the eigenvector with the second-smallest eigenvalue serves as a good heuristic for breaking a graph into two parts that have the smallest number of edges between them.
- Can iterate to divide into as many parts as we like.

Three Matrices That Describe Graphs

1. *Degree matrix*: entry (i, i) is the degree of node i; off-diagonal entries are 0.
2. *Adjacency matrix*: entry (i, j) is 1 if there is an edge between node i and node j, otherwise 0.
3. *Laplacian matrix* = degree matrix minus adjacency matrix.

Example: Matrices

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \qquad \begin{matrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{matrix} \qquad \begin{matrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{matrix}$$

Degree matrix	Adjacency matrix	Laplacian matrix

Every Laplacian Has Zero as an Eigenvalue

- Proof: Each row has a sum of 0, so Laplacian L multiplying an all-1's vector is all 0's, which is also 0 times the all-1's vector.
- Example:

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

The Second-Smallest Eigenvalue

- Let L be a Laplacian matrix, so L = D – A, where D and A are the degree matrix and adjacency matrix for some graph.
- The second eigenvector **x** can be found by minimizing $\mathbf{x}^T L \mathbf{x}$ subject to the constraints:
  1. The length of **x** is 1.
  2. **x** is orthogonal to the eigenvector associated with the smallest eigenvalue.
     - The all-1's vector for Laplacian matrices L.
- And the minimum of $\mathbf{x}^T L \mathbf{x}$ is the eigenvalue.

Meaning of Second Eigenvector

- Let the i-th component of **x** be x_i.
- Aside: Constraint that **x** is orthogonal to all-1's vector says sum of x_i's = 0.
- Break up $\mathbf{x}^T L \mathbf{x}$ as $\mathbf{x}^T L \mathbf{x} = \mathbf{x}^T D \mathbf{x} - \mathbf{x}^T A \mathbf{x}$.
- Since D is diagonal, with degree d_i as i-th diagonal entry, $D\mathbf{x}$ = vector with i-th element $d_i x_i$.
- Therefore, $\mathbf{x}^T D \mathbf{x}$ = sum of $d_i x_i^2$.
- i-th component of $A\mathbf{x}$ = sum of x_j's where node j is adjacent to node i.
- $-\mathbf{x}^T A \mathbf{x}$ = sum of $-2x_i x_j$ over all adjacent i and j.

Second Eigenvector – (2)

- Now we know $\mathbf{x}^T L \mathbf{x} = \Sigma_i d_i x_i^2 - \Sigma_{i,j \text{ adjacent}} 2x_i x_j$.
- Distribute $d_i x_i^2$ over all nodes adjacent to node i.
- Gives us $\mathbf{x}^T L \mathbf{x} = \Sigma_{i,j \text{ adjacent}}\ x_i^2 - 2x_i x_j + x_j^2 = \Sigma_{i,j \text{ adjacent}} (x_i - x_j)^2$.
- Remember: we're minimizing $\mathbf{x}^T L \mathbf{x}$.
- The minimum will tend to make x_i and x_j close when there is an edge between i and j.
- Also, constraint that sum of x_i's = 0 means there will be roughly the same number of positive and negative x_i's.

Second Eigenvector – (3)

- Put another way: if there is an edge between i and j, then there is a good chance that both x_i and x_j will be positive or both negative.
- So partition the graph according to the sign of x_i.
- Likely to minimize the number of edges with one end in either side.

Example: Second Eigenvector.

$$\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ \sqrt{2}-1 \\ 1-\sqrt{2} \\ -1 \end{bmatrix} = \begin{bmatrix} 2-\sqrt{2} \\ 3\sqrt{2}-4 \\ 4-3\sqrt{2} \\ \sqrt{2}-2 \end{bmatrix} = \begin{bmatrix} .586 \\ .242 \\ -.242 \\ -.586 \end{bmatrix}$$

Laplacian matrix

Eigenvalues: $0,\ 2-\sqrt{2},\ 2,\ 2+\sqrt{2}$

Puts A and B in the positive group, C and D in the negative group.

The Affiliation-Graph Model

Overlapping Communities
Maximum-Likelihood Estimation

Elements of the AGM

- We are given a graph and a set of communities.
- We want to find a model that best explains the edges in the graph, assuming:
  1. Nodes (people) can be in any subset of the communities.
  2. For each community C, there is a probability p_C that this community will cause there to be an edge between two members.
  3. Each community independently “inspires” edges.

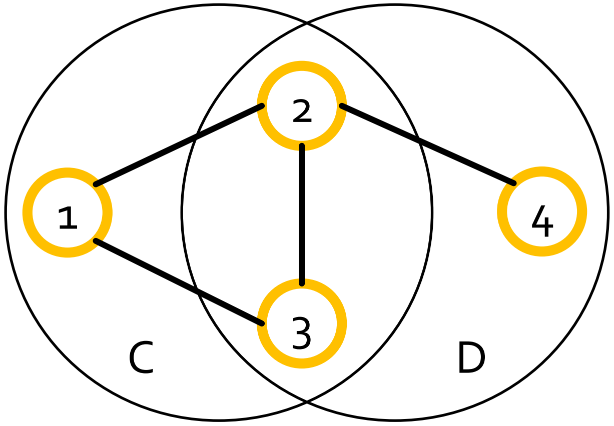
Computing Probabilities of Edges

- Consider two nodes i and j, both members of communities C and D, and no others.
- The probability that there is no edge between i and j is $(1-p_C)(1-p_D)$.
- Thus, the probability of an edge (i, j) is $1 - (1-p_C)(1-p_D)$.
- Generalizes to 1 minus the product of $1-p_C$ over all communities C containing both i and j.
- Tiny ε if i and j do not share any communities.
  - Else there is no way to explain an edge not contained in any community.

Likelihood of a Community Assignment

- Given a graph and a tentative assignment of nodes to sets of communities, we can compute the likelihood of the graph by taking the product of:
  1. For each edge in the graph, the probability this assignment would cause this edge.
  2. For each pair of nodes not connected by an edge, 1 minus the probability that the assignment would cause an edge between these nodes.

Example: Likelihood of a Graph



Technically this is $1 - (1-p_C)$.

$p_{12} = p_{13} = p_C$

$p_{23} = 1 - (1-p_C)(1-p_D)$

$p_{24} = p_{34} = p_D$

$p_{14} = \varepsilon$

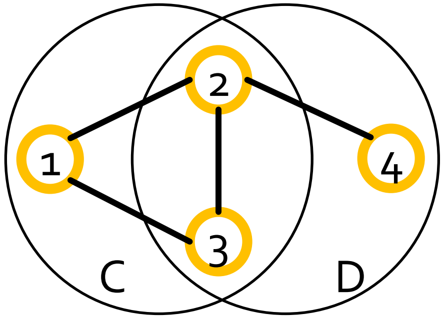
Likelihood of this graph =
$p_{12}p_{13}p_{23}p_{24}(1-p_{14})(1-p_{34})$ =
$(p_C)^2(1 - (1-p_C)(1-p_D))p_D(1-\varepsilon)(1-p_D)$

Very close to 1; drop this factor

Maximum-Likelihood Probabilities

- Given our assignment to communities, we can find the probability p_C associated with each community C that maximizes the probability of seeing this graph.
- Find maximum by gradient descent; i.e., change each p_C slightly in the direction that the partial derivative wrt this variable says will increase the likelihood expression.
  - Iterate to convergence.

Example: Optimizing Probabilities



- For this likelihood expression
  $(p_C)^2(1 - (1\text{-}p_C)(1\text{-}p_D))p_D(1\text{-}p_D)$
  first notice that increasing p_C can only increase the expression.
- Thus, choose $p_C = 1$.
- Expression becomes $p_D(1\text{-}p_D)$.
- Maximized when $p_D = ½$.
- Question for thought: given $p_C = 1$ and $p_D = ½$, what graphs could possibly be generated and what are their probabilities?

Gradient Descent

- The general idea is that we maximize a function of many variables by:

1. Take the partial derivative of the function with respect to each variable.
2. Evaluate the derivatives at the current point (values of all the variables).
3. Change the value of each variable by a small fraction of its partial derivative to get a new point.
4. Repeat (1) – (3) until convergence.

Example: Gradient Descent

- Suppose we start at the point $p_C = .7$, $p_D = .6$
- The derivative of $(p_C)^2(1 – (1-p_C)(1-p_D))p_D(1-p_D)$, with p_C a variable and $p_D = .6$ is $.288p_C + .288p_C^{\,2}$.
- Evaluated at $p_C = .7$ is .34272.
- The derivative with p_D variable and $p_C = .7$ is $.343 - .392p_D - .441p_D^2$.
- Evaluated at $p_D = .6$ is -.05096.
- We might then add 10% of their derivatives to p_C and p_D, yielding $p_C = .734272$ and $p_D = .594904$.

Changing the Node-Community Assignment

- While gradient descent is dandy for finding the best p_C values given an assignment of nodes to communities, it doesn't help with optimizing that assignment.
- Classical approach is to allow incremental changes to a discrete value such as the node-community assignment.
- Example: allow changes that add or delete one node from one community; see if it gives a higher likelihood for its best p_C values.

Making Membership Continuous

- An alternative is to make membership in communities a continuous so you can use gradient descent to optimize them.
- Create a variable F_{xC} for each node x and community C that represents the "degree" to which x is a member of community C.
- Probability that community C will cause an edge between nodes u and v is taken to be $1\text{-}\exp\{-F_{uC}F_{vC}\}$.
- Probability of an edge is constructed from the contributions of all the communities as before.