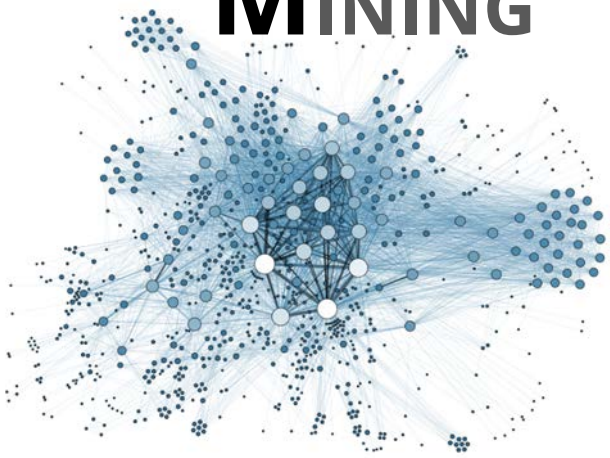
Social Media Mining



Community Analysis

Social Community

[real-world] community

A group of individuals with common *economic*, *social*, or *political* interests or characteristics, often living in *relative proximity*.

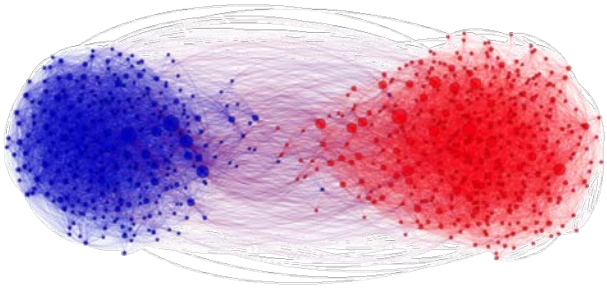
Why analyze communities?

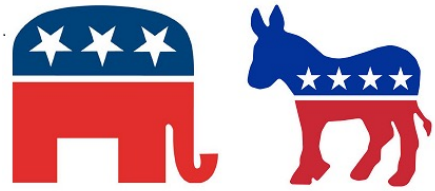
Analyzing communities helps better understand users

– Users form groups based on their interests

Groups provide a clear global view of user interactions

- E.g., find polarization





Some behaviors are only observable in a group setting and not on an individual level

– Some republican can **agree** with some democrats, but their parties can **disagree**

Social Media Communities

- **Formation:**
  – When like-minded users on social media form a link and start interacting with each other

- **More Formal Formation**:
  1. A set of at least two nodes sharing some interest, and
  2. Interactions with respect to that interest.

- Social Media Communities
  – **Explicit** (*emic*): formed by user subscriptions
  – **Implicit** (*etic*): implicitly formed by social interactions
    - **Example:** individuals calling Canada from the United States
    - Phone operator considers them one community for promotional offers

- Other community names: *group*, *cluster*, *cohesive subgroup*, or *module*

Examples of Explicit Social Media Communities

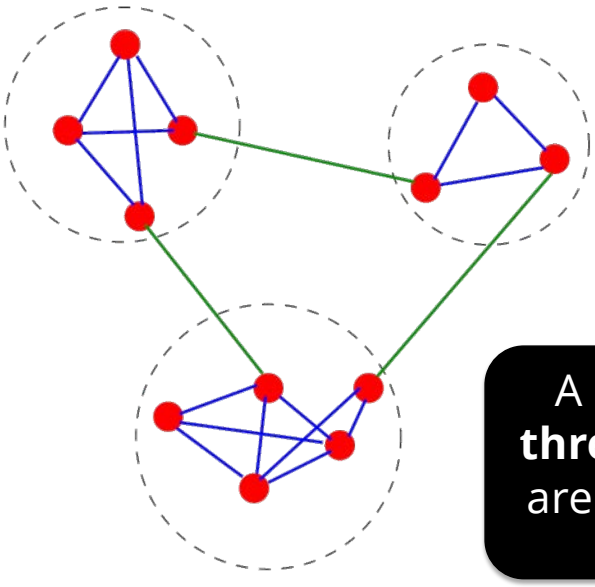
Facebook has groups and communities. Users can
- post messages and images,
- can comment on other messages,
- can like posts, and
- can view activities of other users

In Google+, Circles represent communities

In Twitter, communities form as lists.
- Users join lists to receive information in the form of tweets

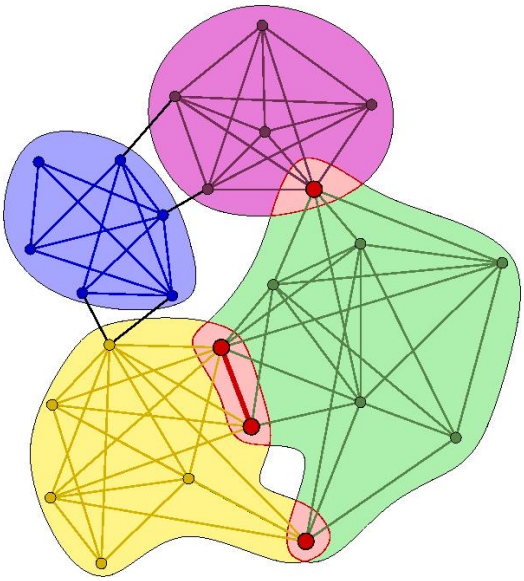
LinkedIn provides *Groups* and *Associations*.
- Users can join professional groups where they can post and share information related to the group

Finding Implicit Communities: An Example

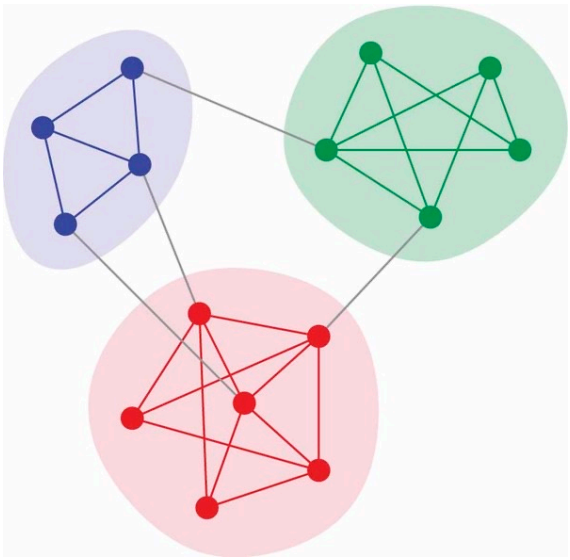


A simple graph in which **three** implicit communities are found, enclosed by the dashed circles

Overlapping vs. Disjoint Communities



Overlapping Communities

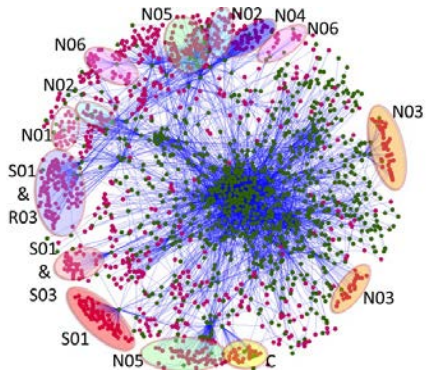


Disjoint Communities

Implicit communities in other domains

Protein-protein interaction networks

- Communities are likely to group proteins having the same specific function within the cell



World Wide Web

- Communities may correspond to groups of pages dealing with the same or related topics

Metabolic networks

- Communities may be related to functional modules such as cycles and pathways

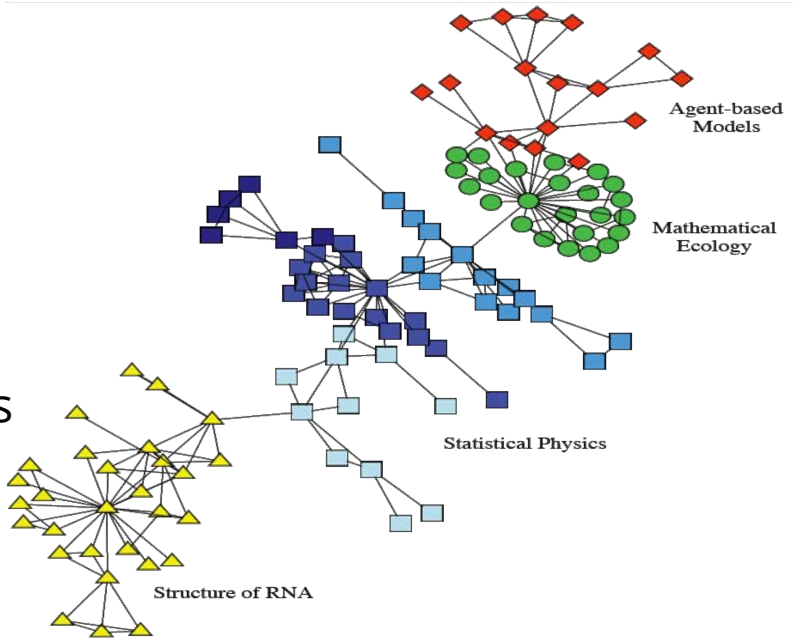
Food webs

- Communities may identify compartments

Real-world Implicit Communities

Collaboration network between scientists working at the Santa Fe Institute.

The colors indicate high level communities and correspond to research divisions of the institute



What is Community Analysis?

- **Community detection**
  - Discovering implicit communities
- **Community evolution**
  - Studying temporal evolution of communities
- **Community evaluation**
  - Evaluating Detected Communities

Community Detection

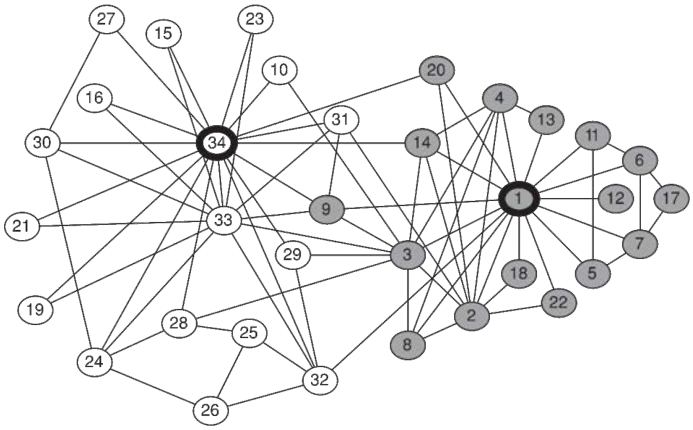
What is community detection?

- The process of finding clusters of nodes ("*communities*")
  - With **Strong** internal connections and
  - **Weak** connections between different communities
- Ideal decomposition of a large graph
  - Completely disjoint communities
  - There are no interactions between different communities.
- In practice,
  - find community partitions that are maximally decoupled.

Why Detecting Communities is Important?

Zachary's karate club

Interactions between 34 members of a karate club for over two years



- The club members split into two groups (**gray** and **white**)
- Disagreement between the administrator of the club (node **34**) and the club's instructor (node **1**),
- The members of one group left to start their own club

The same communities can be found using community detection

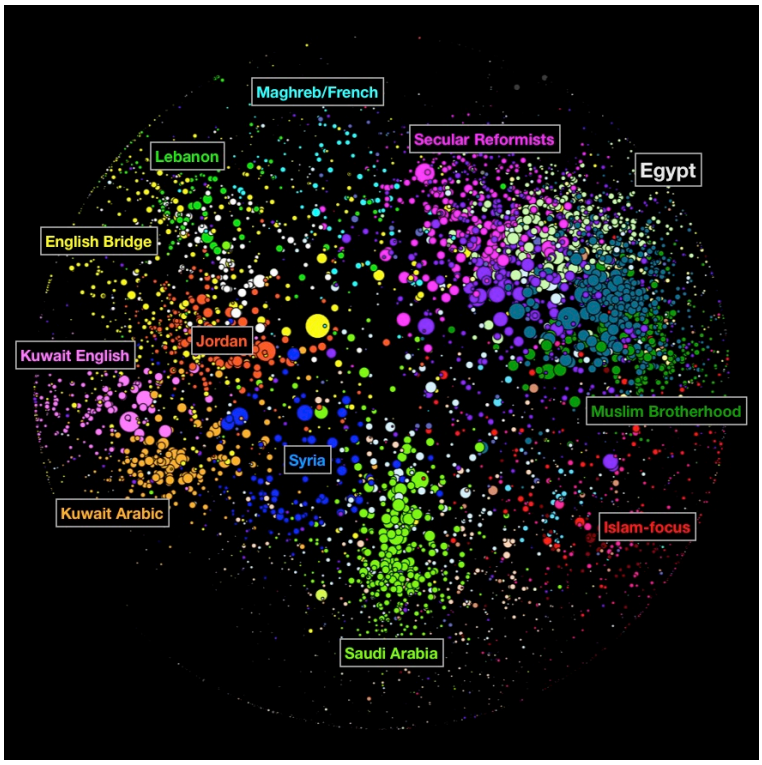
Why Community Detection?

Network Summarization

- A community can be considered as a summary of the whole network
- Easier to visualize and understand

Preserve Privacy

- [Sometimes] a community can reveal some properties without releasing the individuals' privacy information.



Community Detection vs. Clustering

Clustering

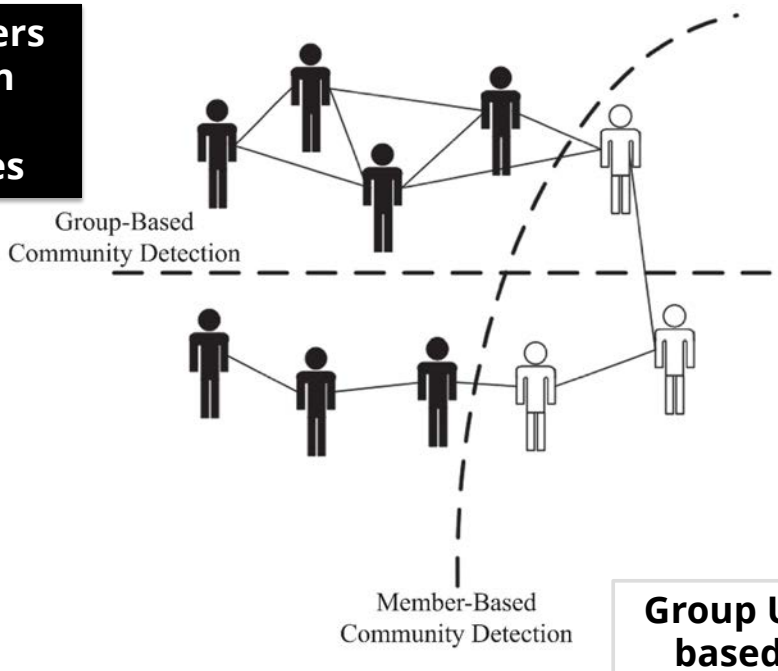
- Data is often non-linked (matrix rows)
- Clustering works on the distance or similarity matrix, e.g., k-means.
- If you use k-means with adjacency matrix rows, you are only considering the ego-centric network

Community detection

- Data is linked (a graph)
- Network data tends to be "discrete", leading to algorithms using the graph property directly
  - k-clique, quasi-clique, or edge-betweenness

Community Detection Algorithms

Group Users based on Group attributes



Group Users based on Member attributes

Member-Based Community Detection

Member-Based Community Detection

- Look at node characteristics; and
- Identify nodes with similar characteristics and consider them a community

Node Characteristics

A. Degree

- Nodes with same (or similar) degrees are in one community
- Example: cliques

B. Reachability

- Nodes that are close (small shortest paths) are in one community
- Example: k-cliques, k-clubs, and k-clans

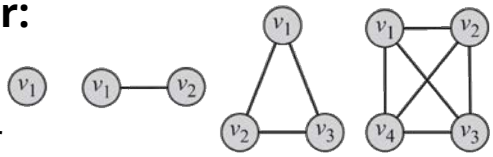
C. Similarity

- Similar nodes are in the same community

A. Node Degree

Most common subgraph searched for:

- **Clique**: a maximum complete subgraph in which all nodes inside the subgraph adjacent to each other



Find communities by searching for

1. **The maximum clique**: **the one** with the largest number of vertices, or
2. **All maximal cliques**: cliques that are not subgraphs of a larger clique; i.e., cannot be further expanded

To overcome this, we can

I. Brute Force
II. Relax cliques
III. Use cliques as the core for larger communities

Both problems are NP-hard

I. Brute-Force Method

Can find all the maximal cliques in the graph

For each vertex v_x, we find the maximal clique that contains node v_x

Algorithm 1 Brute-Force Clique Identification

Require: Adjacency Matrix A, Vertex v_x

1: **return** Maximal Clique C containing v_x
2: CliqueStack = $\{\{v_x\}\}$, Processed = $\{\}$;
3: **while** CliqueStack not empty **do**
4: C=pop(CliqueStack); push(Processed,C);
5: v_{last} = Last node added to C;
6: $N(v_{last}) = \{v_i | A_{v_{last},v_i} = 1\}$.
7: **for all** $v_{temp} \in N(v_{last})$ **do**
8: **if** $C \bigcup \{v_{temp}\}$ is a clique **then**
9: push(CliqueStack, $C \bigcup \{v_{temp}\}$);
10: **end if**
11: **end for**
12: **end while**
13: Return the largest clique from Processed

Impractical for large networks:

- For a complete graph of only 100 nodes, the algorithm will generate at least $2^{99} - 1$ different cliques starting from any node in the graph

Enhancing the Brute-Force Performance

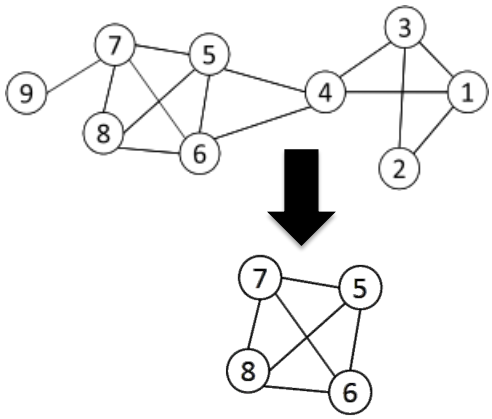
[Systematic] Pruning can help:

- When searching for cliques of size k or larger
- If the clique is found, each node should have a degree equal to or more than $k-1$
- We can first prune all nodes (and edges connected to them) with degrees less than $k-1$
  - More nodes will have degrees less than $k-1$
  - Prune them recursively
- For large k, many nodes are pruned as social media networks follow a power-law degree distribution

Maximum Clique: Pruning…

Example. to find a clique ≥ 4, remove all nodes with degree $\leq (4-1)-1=2$

- Remove nodes 2 and 9
- Remove nodes 1 and 3
- Remove node 4



Even with pruning, cliques are less desirable means for finding communities

- Cliques are rare
- A clique of 1000 nodes, has 999x1000/2 edges
- A single edge removal destroys the clique
- That is less than 0.0002% of the edges!

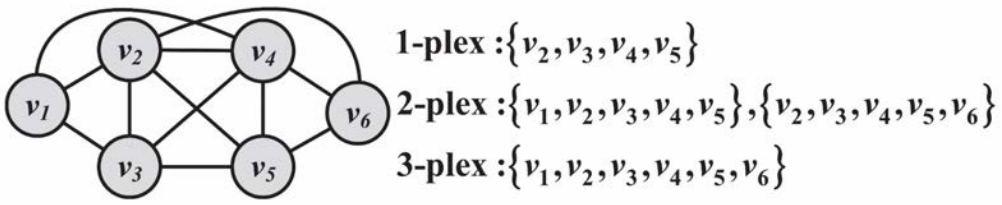
II. Relaxing Cliques

- ***k*-plex**: a set of vertices V in which we have

$$d_v \geq |V| - k, \forall v \in V$$

- d_v is the degree of v in the induced subgraph
  - Number of nodes from V that are connected to v

- Clique of size k is a 1-plex
- Finding the maximum k-plex: **NP-hard**
  - In practice, relatively easier due to smaller search space.



Maximal k-plexes

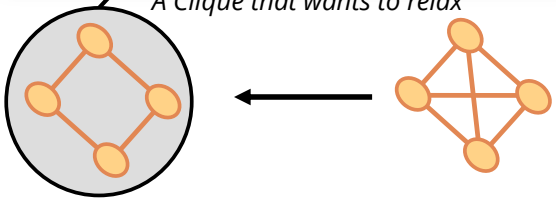
More Cliques Relaxing...

- ***k*-core**: a maximal connected subgraph in which all vertices have degree at least k
  - **Difference with *k*-plex?**

- ***k*-shell**: nodes that are part of the k-core, but are not part of the $(k+1)$-core.

Questions
- 0-core?
- 0-shell?
- 1-core?
- k-cores of the complete graph?

A Clique that wants to relax



A 2-plex or a 2-core?

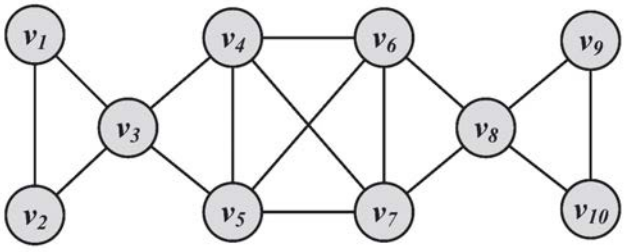
III. Using Cliques as a seed of a Community

Clique Percolation Method (CPM)

- Uses cliques as seeds to find larger communities
- CPM finds overlapping communities

- **Input**
  - A parameter k, and a network
- Procedure
  - Find out all cliques of size k in the given network
  - Construct a clique graph.
    - Two cliques are adjacent if they share $k-1$ nodes
  - Each connected components in the clique graph form a community

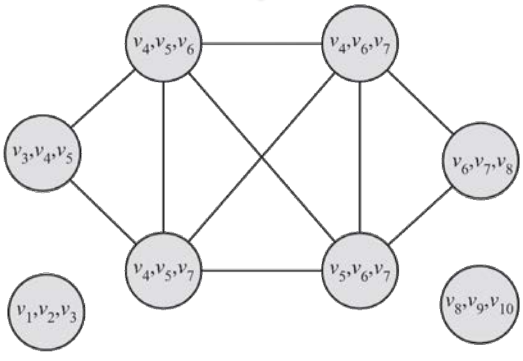
Clique Percolation Method: Example



(a) Graph

Cliques of size 3:

$\{v_1, v_2, v_3\}$, $\{v_3, v_4, v_5\}$,
$\{v_4, v_5, v_6\}$, $\{v_4, v_5, v_7\}$,
$\{v_4, v_6, v_7\}$, $\{v_5, v_6, v_7\}$,
$\{v_6, v_7, v_8\}$, $\{v_8, v_9, v_{10}\}$



(b) CPM Clique Graph

Communities:

$\{v_1, v_2, v_3\}$,
$\{v_8, v_9, v_{10}\}$,
$\{v_3, v_4, v_5, v_6, v_7, v_8\}$

B. Node Reachability

The two extremes

Nodes are assumed to be in the same community

1. If there is a path between them (regardless of the distance) or
2. They are so close as to be immediate neighbors.

How? Find using BFS/DFS

Challenge: most nodes are in one community (giant component)

How? Finding Cliques

Challenge: Cliques are challenging to find and are rarely observed

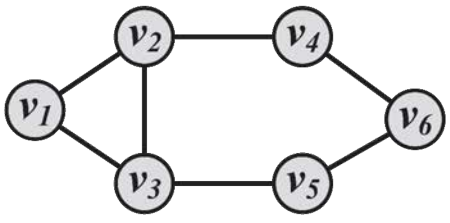
Solution: find communities that are in between cliques and connected components in terms of connectivity and have small shortest paths between their nodes

Special Subgraphs

1. k-**Clique**: a **maximal** subgraph in which the largest shortest path distance between any nodes is less than or equal to k
   - Nodes on the shortest path should not be necessarily be part of the subgraph

2. k-**Club**: follows the same definition as a k-clique
   - **Additional Constraint:** nodes on the shortest paths should be part of the subgraph (i.e., diameter)

3. k-**Clan:** a k-clique where for all shortest paths within the subgraph the distance is equal or less than k.
   - All k-clans are k-cliques, but not vice versa.



2-cliques : $\{v_1, v_2, v_3, v_4, v_5\}, \{v_2, v_3, v_4, v_5, v_6\}$

2-clubs : $\{v_2, v_3, v_4, v_5, v_6\}, \{v_1, v_2, v_3, v_4\}, \{v_1, v_2, v_3, v_5\}$

2-clans : $\{v_2, v_3, v_4, v_5, v_6\}$

More Special Subgraphs!

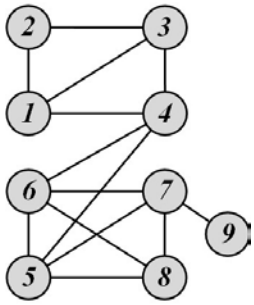
- **k-truss:** the largest subgraph where all edges belong to $k - 2$ triangles
- What is the relationship between a **k-core** and **k-truss**?

C. Node Similarity

- Similar (or most similar) nodes are assumed to be in the same community.
  - A classical clustering algorithm (e.g., k-means) is applied to node similarities to find communities.
- Node similarity can be defined
  - Using the similarity of node neighborhoods (**Structural Equivalence**) – Ch. 3
  - Similarity of social circles (**Regular Equivalence**) – Ch. 3

Structural equivalence: two nodes are structurally equivalent iff. they are connecting to the same set of actors

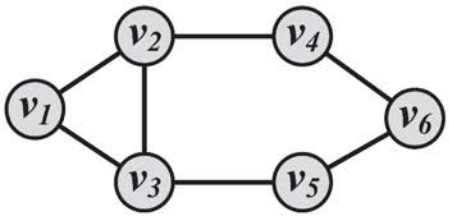
Nodes 1 and 3 are structurally equivalent, So are nodes 5 and 7.



Node Similarity (Structural Equivalence)

Jaccard Similarity $\sigma_{\text{Jaccard}}(v_i, v_j) = \frac{|N(v_i) \cap N(v_j)|}{|N(v_i) \cup N(v_j)|}$

Cosine similarity $\sigma_{\text{Cosine}}(v_i, v_j) = \frac{|N(v_i) \cap N(v_j)|}{\sqrt{|N(v_i)||N(v_j)|}}$



$$\sigma_{\text{Jaccard}}(v_2, v_5) = \frac{|\{v_1, v_3, v_4\} \cap \{v_3, v_6\}|}{|\{v_1, v_3, v_4, v_6\}|} = 0.25$$

$$\sigma_{\text{Cosine}}(v_2, v_5) = \frac{|\{v_1, v_3, v_4\} \cap \{v_3, v_6\}|}{\sqrt{|\{v_1, v_3, v_4\}||\{v_3, v_6\}|}} = 0.40$$

Group-Based Community Detection

Group-Based Community Detection

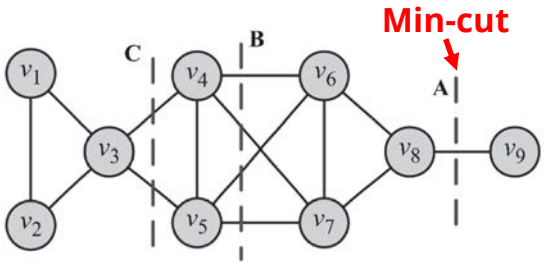
Group-based community detection: finding communities that have certain **group properties**

Group Properties:

I. **Balanced:** Spectral clustering
II. **Robust:** k-connected graphs
III. **Modular:** Modularity Maximization
IV. **Dense:** Quasi-cliques
V. **Hierarchical:** Hierarchical clustering

I. Balanced Communities

- Community detection can be thought of *graph clustering*
- **Graph clustering:** we cut the graph into several partitions and assume these partitions represent communities
- **Cut**: partitioning (*cut*) of the graph into two (or more) sets (*cutsets*)
  - **The size of the cut** is the number of edges that are being cut
- **Minimum cut (min-cut) problem**: find a graph partition such that the number of edges between the two sets is minimized



Min-cuts can be computed efficiently using the max-flow min-cut theorem

Min-cut often returns an imbalanced partition, with one set being a singleton

Ratio Cut and Normalized Cut

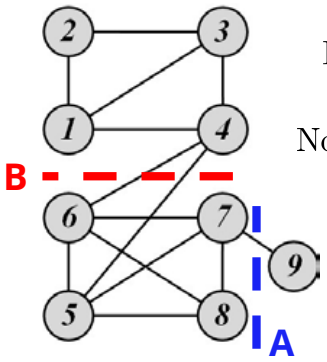
- To mitigate the min-cut problem we can change the objective function to consider community size

$$\text{Ratio Cut}(P) = \frac{1}{k}\sum_{i=1}^{k}\frac{\text{cut}(P_i,\bar{P}_i)}{|P_i|}$$

$$\text{Normalized Cut}(P) = \frac{1}{k}\sum_{i=1}^{k}\frac{\text{cut}(P_i,\bar{P}_i)}{\text{vol}(P_i)}$$

- $\bar{P}_i = V - P_i$ is the complement cut set
- $cut(P_i, \bar{P}_i)$ is the size of the cut
- $vol(P_i) = \sum_{v \in P_i} d_v$

Ratio Cut & Normalized Cut: Example



$$\text{Ratio Cut}(P) = \frac{1}{k}\sum_{i=1}^{k}\frac{\text{cut}(P_i,\bar{P}_i)}{|P_i|}$$

$$\text{Normalized Cut}(P) = \frac{1}{k}\sum_{i=1}^{k}\frac{\text{cut}(P_i,\bar{P}_i)}{\text{vol}(P_i)}$$

$$vol(P_i) = \sum_{v \in P_i} d_v$$

For Cut A

$$\text{Ratio Cut}(\{1,2,3,4,5,6,7,8\},\{9\}) = \tfrac{1}{2}(\tfrac{1}{1}+\tfrac{1}{8}) = 9/16 = 0.56$$

$$\text{Normalized Cut}(\{1,2,3,4,5,6,7,8\},\{9\}) = \tfrac{1}{2}(\tfrac{1}{1}+\tfrac{1}{27}) = 14/27 = 0.52$$

For Cut B

$$\text{Ratio Cut}(\{1,2,3,4\},\{5,6,7,8,9\}) = \tfrac{1}{2}(\tfrac{2}{4}+\tfrac{2}{5}) = 9/20 = 0.45 < 0.56$$

$$\text{Normalized Cut}(\{1,2,3,4\},\{5,6,7,8,9\}) = \tfrac{1}{2}(\tfrac{2}{12}+\tfrac{2}{16}) = 7/48 = 0.15 < 0.52$$

Both ratio cut and normalized cut prefer a balanced partition

Spectral Clustering

Reformulating ratio cut (or normalized cut) in matrix format

- Let $X_{ij} = 1$, when node i is member of community j; 0, otherwise
- Let $D = diag(d_1, d_2, \cdots, d_n)$ be the diagonal degree matrix
- The ith entry on the diagonal of X^TAX is the number of edges that are inside community i.
- The ith element on the diagonal of X^TDX is the number of edges that are connected to members of community i.
- The ith element on the diagonal of $X^T(D-A)X$ is the number of edges in the cut that separates community i from other nodes.

The ith diagonal element of $X^T(D-A)X$ is equivalent to $\text{cut}(P_i, \bar{P}_i)$

Spectral Clustering

So ratio cut is $X^T(D-A)X$ $\quad \text{Ratio Cut}(P) = \frac{1}{k}\sum_{i=1}^{k}\frac{\text{cut}(P_i,\bar{P}_i)}{|P_i|}$

$$\begin{aligned}\text{Ratio Cut}(P) &= \frac{1}{k}\sum_{i=1}^{k}\frac{\text{cut}(P_i,\bar{P}_i)}{|P_i|}\\ &= \frac{1}{k}\sum_{i=1}^{k}\frac{X_i^T(D-A)X_i}{X_i^TX_i}\\ &= \frac{1}{k}\sum_{i=1}^{k}\hat{X}_i^T(D-A)\hat{X}_i\end{aligned}$$

$$\hat{X}_i = \hat{X}_i/(\hat{X}_i^T\hat{X}_i)^{1/2}$$

Spectral Clustering

Both ratio/normalized cut can be reformulated as

$$\min_{\hat{X}} \mathrm{Tr}(\hat{X}^T L \hat{X})$$

$$L = \begin{cases} D - A & \text{Ratio Cut Laplacian, i.e., Unnormalized Laplacian} \\ I - D^{-1/2} A D^{-1/2} & \text{Normalized Laplacian for Normalized Cut.} \end{cases}$$

$D = diag(d_1, d_2, \cdots, d_n)$ is a diagonal degree matrix

- Spectral relaxation:

$$\min_{\hat{X}} \mathrm{Tr}(\hat{X}^T L \hat{X})$$
$$s.t.\ \hat{X}^T \hat{X} = I_k$$

Optimal Solution

$\hat{X}$ is the top eigenvectors with the smallest eigenvalues

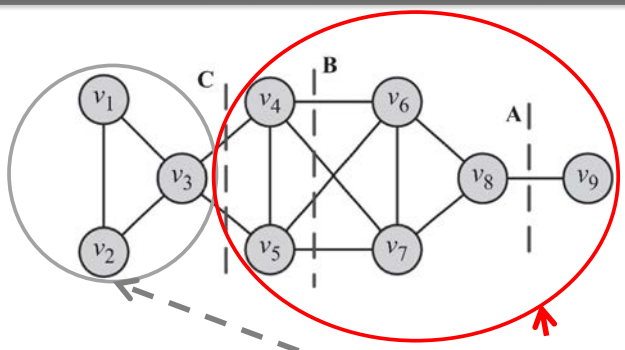
Recovering Integer Membership Values

- Because we performed spectral relaxation, the matrix obtained is not integer valued

- To recover X from $\hat{X}$ we can run k-means on $\hat{X}$

Spectral Clustering: Example

$$A = \begin{bmatrix} 0&1&1&0&0&0&0&0&0 \\ 1&0&1&0&0&0&0&0&0 \\ 1&1&0&1&1&0&0&0&0 \\ 0&0&1&0&1&1&1&0&0 \\ 0&0&1&1&0&1&1&0&0 \\ 0&0&0&1&1&0&1&1&0 \\ 0&0&0&1&1&1&0&1&0 \\ 0&0&0&0&0&1&1&0&1 \\ 0&0&0&0&0&0&0&1&0 \end{bmatrix}$$



Two communities:
$\{v_1, v_2, v_3\}$
$\{v_4, v_5, v_6, v_7, v_8, v_9\}$

$D = diag(2,2,4,4,4,4,4,3,1)$

$\min_{\hat{X}} \mathrm{Tr}(\hat{X}^T L \hat{X})$

$$L = D - A = \begin{bmatrix} 2&-1&-1&0&0&0&0&0&0 \\ -1&2&-1&0&0&0&0&0&0 \\ -1&-1&4&-1&-1&0&0&0&0 \\ 0&0&-1&4&-1&-1&-1&0&0 \\ 0&0&-1&-1&4&-1&-1&0&0 \\ 0&0&0&-1&-1&4&-1&-1&0 \\ 0&0&0&-1&-1&-1&4&-1&0 \\ 0&0&0&0&0&-1&-1&3&-1 \\ 0&0&0&0&0&0&0&-1&1 \end{bmatrix}$$

2 Eigenvectors i.e., we want 2 communities

$$\begin{bmatrix} 0.33 & -0.46 \\ 0.33 & -0.46 \\ 0.33 & -0.26 \\ 0.33 & 1.16\times10^{-16} \\ 0.33 & 1.16\times10^{-16} \\ 0.33 & 0.13 \\ 0.33 & 0.13 \\ 0.33 & 0.33 \\ 0.33 & 0.59 \end{bmatrix}$$

k-means, $k = 2$

More on Laplacian

Laplacian and Diffusion:

- Consider a substance that diffuses from node to node on a graph (e.g., gas or fluid)
- The substance quantity for node j is $\varphi(j)$
- The diffusion depends on the difference in quantity between nodes
  - e.g., gas pressure difference
- Then we have

$$\frac{d\phi_i}{dt} = \alpha \sum_j A_{ji}(\phi_j - \phi_i)$$

Laplacian and Diffusion

$$\frac{d\phi_i}{dt} = \alpha \sum_j A_{ji}(\phi_j - \phi_i)$$
$$= \alpha \sum_j A_{ji}\phi_j - \alpha \sum_j A_{ji}\phi_i$$
$$= \alpha \sum_j A_{ji}\phi_j - \alpha\phi_i \sum_j A_{ji}$$
$$= \alpha \sum_j A_{ji}\phi_j - \alpha\phi_i d_i$$
$$= \alpha \sum_j (A_{ji} - \delta_{ij}d_i)\phi_j \quad \rightarrow \quad -L_{ij}$$

$$\frac{d\phi}{dt} = -\alpha L\phi \quad \rightarrow \quad \frac{d\phi}{dt} + \alpha L\phi = 0$$

Why do we call it the Laplacian? As this is equivalent to heat equation ($\frac{d\varphi}{dt} + \alpha\nabla^2\varphi = 0$) and L replaces the Laplace operator ∇^2

Laplacian and Diffusion

$$\frac{d\phi}{dt} + \alpha L\phi = 0$$

State $\varphi(t)$ in terms of eigenvalue-eigenvectors of L

– L is symmetric and has real eigenvalues.

$$\phi(t) = \sum_i w_i(t)v_i$$ Some coefficient!

$$\sum_i (\frac{dw_i(t)}{dt} + \alpha\lambda_i w_i(t))v_i = 0$$

Eigenvalues are orthogonal, so right multiply by any v_j

$$\forall i, \quad \frac{dw_i(t)}{dt} + \alpha\lambda_i w_i(t) = 0$$

$$w_i(t) = w_i(0)e^{-\alpha\lambda_i t}$$

II. Robust Communities

- The goal is find subgraphs robust enough such that removing some edges or vertices does not disconnect the subgraph

- **k-vertex connected (k-connected) graph:**
  - k is the minimum number of nodes that must be removed to disconnect the graph
- **k-edge connected:** at least k edges must be removed to disconnect the graph

- Examples:
  - Complete graph of size n: unique n-connected graph
  - A cycle: 2-connected graph

III. Modular Communities

Consider a graph $G(V, E)$, where the degrees are known beforehand however edges are not

- Consider two vertices v_i and v_j with degrees d_i and d_j.
- $|E| = m$

What is an expected number of edges between v_i and v_j?

- For any edge going out of v_i randomly the probability of this edge getting connected to vertex v_j is

$$\frac{d_j}{\sum_i d_i} = \frac{d_j}{2m}$$

Modularity and Modularity Maximization

- Given a degree distribution, we know the expected number of edges between any pairs of vertices
- We assume that real-world networks should be far from random. Therefore, the more distant they are from this randomly generated network, the more structural they are.
- Modularity defines this distance and modularity maximization tries to maximize this distance

Normalized Modularity

Consider a partitioning of the data $P = (P_1, P_2, P_3, \dots, P_k)$

For partition P_x, this distance can be defined as

$$\sum_{i,j \in P_x} A_{ij} - \frac{d_i d_j}{2m}$$

This distance can be generalized for a partitioning P

$$\sum_{x=1}^{k} \sum_{i,j \in P_x} A_{ij} - \frac{d_i d_j}{2m}$$

The normalized version of this distance is defined as Modularity

$$Q = \frac{1}{2m} \sum_{x=1}^{k} \sum_{i,j \in P_x} A_{ij} - \frac{d_i d_j}{2m}$$

Modularity Maximization

Modularity matrix $B = A - dd^T/2m$

$d \in \mathbb{R}^{n\times 1}$ is the degree vector for all nodes

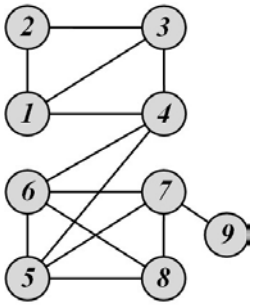
$$Q = \frac{1}{2m}\sum_{x=1}^{k}\sum_{i,j\in P_x} A_{ij} - \frac{d_i d_j}{2m}$$

Reformulation of the modularity

$$Q = \frac{1}{2m}\mathrm{Tr}(X^T B X)$$

- $X \in \mathbb{R}^{n\times k}$ is the indicator (partition membership) function:
  - $X_{ij} = 1$ iff. $v_i \in P_j$
- Similar to Spectral clustering,
  - We relax X to be orthogonal, i.e., matrix $\hat{X}$
  - The optimal solution for $\hat{X}$ is the top k eigenvectors of B.
  - To recover the original X, we can run k-means on $\hat{X}$
- Difference: in Modularity, the top k eigenvalues have to be positive!

Modularity Maximization: Example



$$Q = \frac{1}{2m}\mathrm{Tr}(X^T B X)$$

Two Communities:
{1, 2, 3, 4}
and
{5, 6, 7, 8, 9}

k-means

$B = A - dd^T/2m$

$B_{ij} = A_{ij} - d_i d_j/2m$

$$B = \begin{bmatrix}
-0.32 & 0.79 & 0.68 & 0.57 & -0.43 & -0.43 & -0.43 & -0.32 & -0.11 \\
0.79 & -0.14 & 0.79 & -0.29 & -0.29 & -0.29 & -0.29 & -0.21 & -0.07 \\
0.68 & 0.79 & -0.32 & 0.57 & -0.43 & -0.43 & -0.43 & -0.32 & -0.11 \\
0.57 & -0.29 & 0.57 & -0.57 & 0.43 & 0.43 & -0.57 & -0.43 & -0.14 \\
-0.43 & -0.29 & -0.43 & 0.43 & -0.57 & 0.43 & 0.43 & 0.57 & -0.14 \\
-0.43 & -0.29 & -0.43 & 0.43 & 0.43 & -0.57 & 0.43 & 0.57 & -0.14 \\
-0.43 & -0.29 & -0.43 & -0.57 & 0.43 & 0.43 & -0.57 & 0.57 & 0.86 \\
-0.32 & -0.21 & -0.32 & -0.43 & 0.57 & 0.57 & 0.57 & -0.32 & -0.11 \\
-0.11 & -0.07 & -0.11 & -0.14 & -0.14 & -0.14 & 0.86 & -0.11 & -0.04
\end{bmatrix}$$

Modularity Matrix

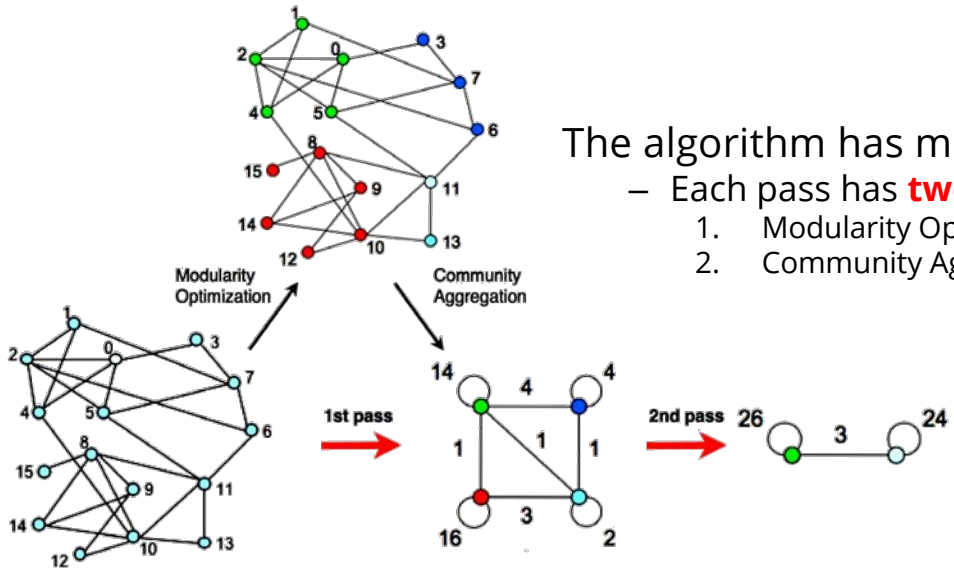
2 eigenvectors

$$\begin{bmatrix}
0.44 & -0.00 \\
0.38 & 0.23 \\
0.44 & -0.00 \\
0.17 & -0.48 \\
-0.29 & -0.32 \\
-0.29 & -0.32 \\
-0.38 & 0.34 \\
-0.34 & -0.08 \\
-0.14 & 0.63
\end{bmatrix}$$

Making Modularity Optimization Faster

- The matrix method discussed is slow and does not scale to millions of nodes (and billions of edges)
- We can perform greedy optimization of modularity to speed-up the process
- Louvain Method
  - A greedy modularity optimization method for community detection
    - Invented when all authors affiliated with ***Université catholique de Louvain (UCL)***

Louvain Method



The algorithm has multiple passes
- Each pass has **two phases**
  1. Modularity Optimization
  2. Community Aggregation

Image from
Blondel, Vincent D., Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. "Fast unfolding of communities in large networks." Journal of statistical mechanics: theory and experiment 2008, no. 10 (2008): P10008.

Louvain Method

Start with a weighted network where all nodes are in their own communities (i.e., n communities)

First Phase:

- For each node v_i,
  - For all neighbors $v_j \in N(v_i)$:
    - compute the modularity gain if v_i is removed from its community and placed in the community of v_j.
  - Find the community with the maximum modularity gain
  - If the maximum gain is positive, remove v_i from its community, and place it in that community
  - If no positive gain, do not change communities
- Repeat until no node changes its community

Important Points about Phase I

- A point can be considered multiple times
- A Local Minima of modularity maximization is achieved in phase I
- Phase I is order dependent
  - The modularity achieved is more or less stable and is less dependent on the initial order
  - The computation time depends on the initial order.

Louvain Method

Second Phase:

- Build a new network
  - Nodes are communities
  - Edges are the edges between nodes in the corresponding communities (weights are sum of the weights)
  - Self-loops represent edges within the community

- The algorithm creates hierarchies of communities
- It usually ends in less than 10 passes
- It is seems to be an $O(n \log n)$ algorithm

Why is it fast?

$$Q = \frac{1}{2m} \sum_{x=1}^{k} \sum_{i,j \in P_x} A_{ij} - \frac{d_i d_j}{2m}$$

$$\delta(x, y) = \begin{cases} 0, & \text{if } x \neq y \\ 1, & \text{if } x = y \end{cases}$$

Modularity can be restated as

Kronecker delta function

$$Q = \frac{1}{2m} \sum_{i,j} [A_{ij} - \frac{d_i d_j}{2m}] \delta(c_i, c_j)$$

community of node i

Consider node i, in an isolated community, when it moves to community C

- The gain in modularity can be computed easily

$$\Delta Q = [\frac{\sum_{in} + 2\sum_{i,in}}{2m} - (\frac{\sum_{total} + d_i}{2m})^2] - [\frac{\sum_{in}}{2m} - (\frac{\sum_{total}}{2m})^2 - (\frac{d_i}{2m})^2]$$

Edges within C **Edges between i and C** **All edges connected to members of C**

IV. Dense Communities: γ-*dense*

- The density of a graph defines how close a graph is to a clique

$$\gamma = \frac{|E|}{\binom{|V|}{2}}$$

- A subgraph $G(V, E)$ is a γ-*dense* (or quasi-clique) if

$$|E| \geq \gamma \binom{|V|}{2}$$

- A 1-dense graph is a clique
- We can find quasi-cliques using the brute force algorithm discussed previously, but there are more efficient methods.

Finding Maximal γ-dense Quasi-Cliques

We can use a two-step procedure consisting of "local search" and "heuristic pruning"

Local search

- Sample a subgraph, and find a maximal γ-*dense* quasi-clique
  - A greedy approach is to expand a quasi-clique by all of its high-degree neighbors until the density drops below γ

Heuristic pruning

- For a γ-*dense* quasi-clique of size k, we recursively remove nodes with degree less than γk and incident edges
  - We can start from low-degree nodes and recursively remove all nodes with degree less that γk

V. Hierarchical Communities

- Previous methods consider communities at a single level
  - Communities may have hierarchies.
    - Each community can have sub/super communities.
  - Hierarchical clustering deals with this scenario and generates community hierarchies.

- Initially n members are considered as either 1 or n communities in hierarchical clustering. These communities are gradually
  - merged (agglomerative hierarchical clustering) or
  - split (divisive hierarchical clustering)

Hierarchical Community Detection

- **Goal**: build a hierarchical structure of communities based on network topology

- Allow the analysis of a network at different resolutions

- Representative approaches:
  - Divisive Hierarchical Clustering
  - Agglomerative Hierarchical clustering

Divisive Hierarchical Clustering

- Divisive clustering
  - Partition nodes into several sets
  - Each set is further divided into smaller ones
  - Network-centric partition can be applied for the partition
- One particular example:

Girvan-Newman Algorithm: recursively remove the "weakest" links within a "community"

  - Find the edge with the weakest link
  - Remove the edge and update the corresponding strength of each edge

- Recursively apply the above two steps until a network is discomposed into a desired number of connected components.
- Each component forms a community

Edge Betweenness

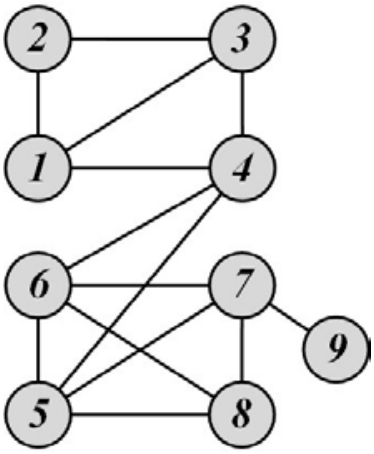
- To determine weakest links, the algorithm uses **edge betweenness**.

Edge betweenness is the number of shortest paths that pass along with the edge

- Edge betweenness measures the "bridgeness" of an edge between two communities

- The edge with high betweenness tends to be the bridge between two communities.

Edge Betweenness: Example

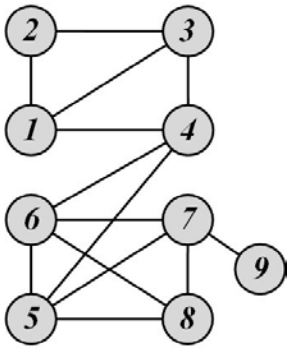


The edge betweenness of $e(1,2)$ is $6/2 + 1 = 4$, as all the shortest paths from 2 to $\{4,5,6,7,8,9\}$ have to either pass $e(1,2)$ or $e(2,3)$, and $e(1,2)$ is the shortest path between 1 and 2

The Girvan-Newman Algorithm

1. Calculate edge betweenness for all edges in the graph.
2. Remove the edge with the highest betweenness.
3. Recalculate betweenness for all edges affected by the edge removal.
4. Repeat until all edges are removed.

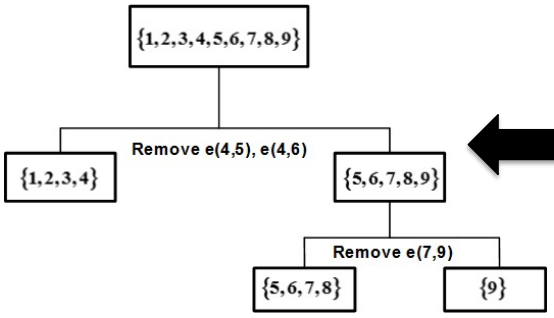
Edge Betweenness Divisive Clustering: Example



Initial betweenness value

	1	2	3	4	5	6	7	8	9
1	0	4	1	9	0	0	0	0	0
2	4	0	4	0	0	0	0	0	0
3	1	4	0	9	0	0	0	0	0
4	9	0	9	0	10	10	0	0	0
5	0	0	0	10	0	1	6	3	0
6	0	0	0	10	1	0	6	3	0
7	0	0	0	0	6	6	0	2	8
8	0	0	0	0	3	3	2	0	0
9	0	0	0	0	0	0	8	0	0

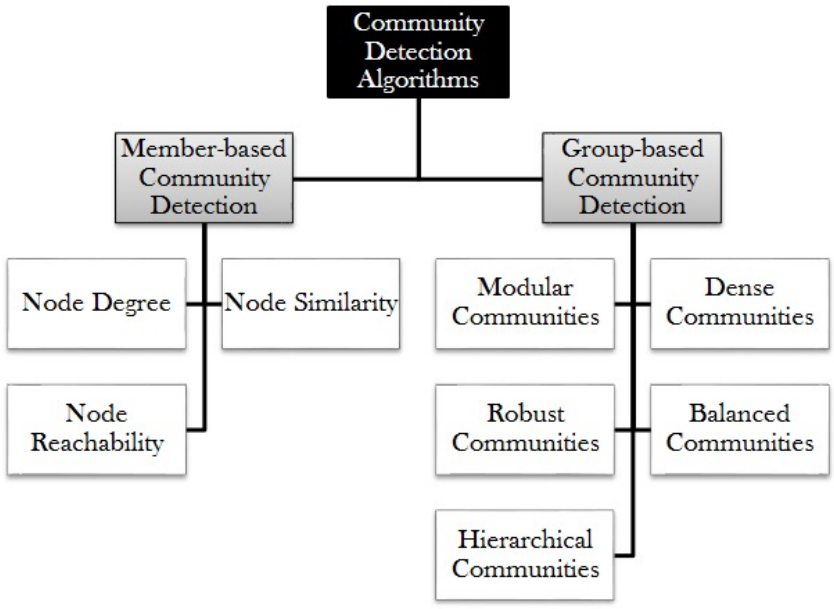
the first edge that needs to be removed is e(4, 5) (or e(4, 6))



By removing e(4, 5), we compute the edge betweenness once again; this time, e(4, 6) has the highest betweenness value: 20.

This is because all shortest paths between nodes {1,2,3,4} to nodes {5,6,7,8,9} must pass e(4, 6); therefore, it has betweenness 4×5 = 20.

Community Detection Algorithms



Community Evolution

Network and Community Evolution

- How does a **network** change over time?
- How does a **community** change over time?
- What properties do you expect to remain roughly constant?
- What properties do you expect to change?
- For example,
  - Where do you expect new edges to form?
  - Which edges do you expect to be dropped?

How Networks Evolve?

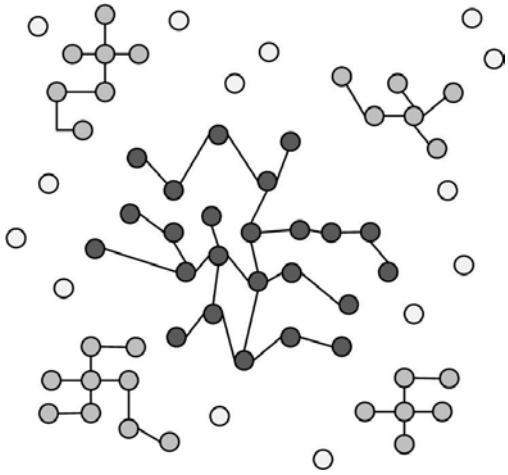
Network Growth Patterns

1. Network Segmentation
2. Graph Densification
3. Diameter Shrinkage

1. Network Segmentation

- Often, in evolving networks, segmentation takes place, where the large network is decomposed over time into three parts

1. **Giant Component**: As network connections stabilize, a giant component of nodes is formed, with a large proportion of network nodes and edges falling into this component.
2. **Stars**: These are isolated parts of the network that form star structures. A star is a tree with one internal node and n leaves.
3. **Singletons**: These are orphan nodes disconnected from all nodes in the network.



2. Graph Densification

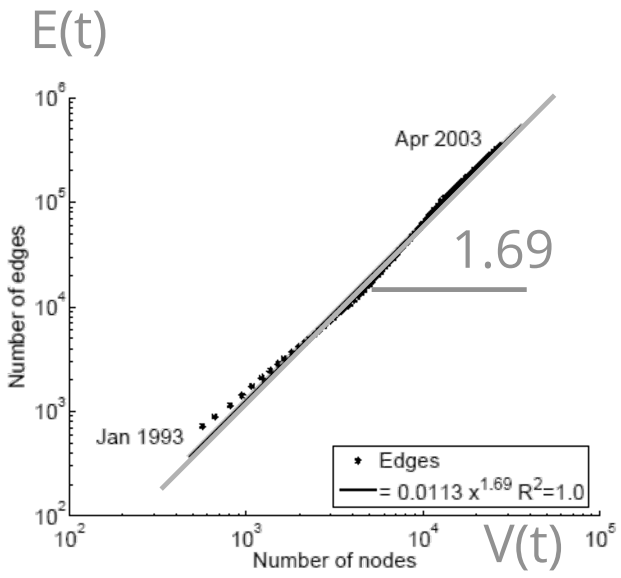
- The density of the graph increases as the network grows
  - The number of edges increases faster than the number of nodes does

$$E(t) \propto V(t)^{\alpha}$$

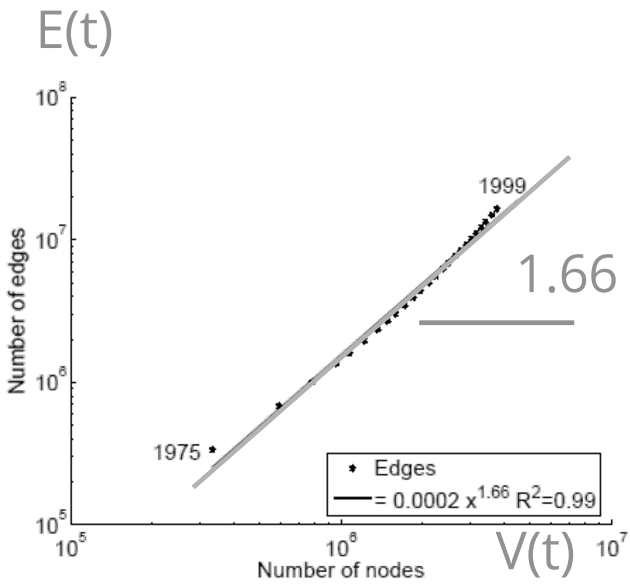
- Densification exponent: $1 \leq \alpha \leq 2$:
  - $\alpha = 1$: linear growth – constant out-degree
  - $\alpha = 2$: quadratic growth – clique

$E(t)$ and $V(t)$ are numbers of edges and nodes respectively at time t

Densification in Real Networks



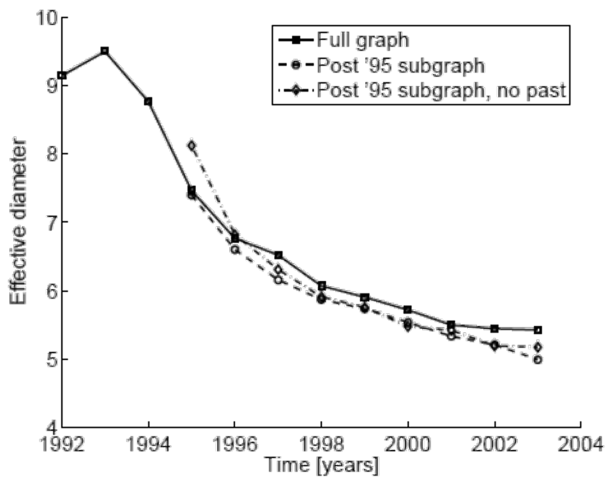
Physics Citations



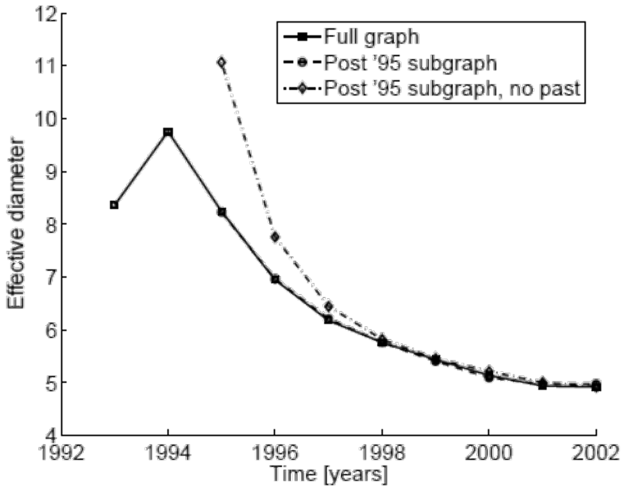
Patent Citations

3. Diameter Shrinking

- In networks diameter shrinks over time



ArXiv citation graph

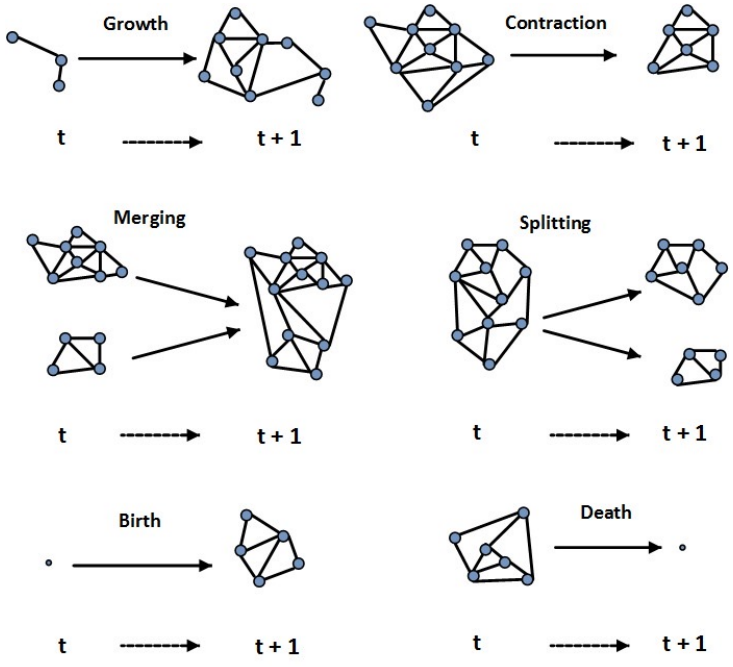


Affiliation Network

How Communities Evolve?

Community Evolution

- Communities also expand, shrink, or dissolve in dynamic networks



Community Detection in Evolving Networks

Extending Previous Methods

1. Take t snapshots of the network, $G_1, G_2, \cdots, G_t$ where G_i is a snapshot at time i
2. Perform a static community detection algorithm (all methods discussed before) on all snapshots independently
3. Assign community members based on communities found in all t time stamps.
   - E.g., Assign nodes to communities based on voting (assign nodes to communities they belong to the most over time)

Unstable in highly dynamic networks as community memberships are always changing

Evolutionary Clustering

- Assume that communities don't change most of the time
- Minimize an objective function that considers
  - **Snapshot Cost**. Communities at different times (**SC**)
  - **Temporal Cost**. How communities evolve (**TC**)
- Objective function is defined as

$$Cost = \alpha SC + (1-\alpha)TC$$

$$0 \leq \alpha \leq 1$$

Evolutionary Clustering

- If we use spectral clustering for each snapshot
- Then, $Cost_t$, the objective function at time t is

$$\begin{aligned} Cost_t &= \alpha\ SC + (1-\alpha)\ TC \\ &= \alpha\ Tr\left(X_t^T L X_t\right) + (1-\alpha)\ TC \end{aligned}$$

- X_t is the community membership matrix at t
- To define TC we can use $TC = ||X_t - X_{t-1}||^2$
- Challenges with this definition
  - Assumes that we have the same number of communities at time $t-1$ and t
  - X_t is non-unique (any orthogonal transformation is still a solution)

Evolutionary Clustering

- We can define TC as

$$\begin{aligned} TC &= \frac{1}{2}||X_t X_t^T - X_{t-1}X_{t-1}^T||^2, \\ &= \frac{1}{2}Tr\big((X_t X_t^T - X_{t-1}X_{t-1}^T)^T (X_t X_t^T - X_{t-1}X_{t-1}^T)\big), \\ &= \frac{1}{2}Tr\big(X_t X_t^T X_t X_t^T - 2X_t X_t^T X_{t-1}X_{t-1}^T + X_{t-1}X_{t-1}^T X_{t-1}X_{t-1}^T\big), \\ &= Tr\big(I - X_t X_t^T X_{t-1}X_{t-1}^T\big), \\ &= Tr\big(I - X_t^T X_{t-1}X_{t-1}^T X_t\big) \end{aligned}$$

$$Tr(AB) = Tr(BA)$$

- Hence cost will be

$$\begin{aligned} Cost_t &= \alpha\, Tr\big(X_t^T L X_t\big) + (1-\alpha)\, \frac{1}{2}||X_t X_t^T - X_{t-1}X_{t-1}^T||^2, \\ &= \alpha\, Tr\big(X_t^T L X_t\big) + (1-\alpha)\, Tr\big(I - X_t^T X_{t-1}X_{t-1}^T X_t\big), \\ &= \alpha\, Tr\big(X_t^T L X_t\big) + (1-\alpha)\, Tr\big(X_t^T I X_t - X_t^T X_{t-1}X_{t-1}^T X_t\big), \\ &= Tr\big(X_t^T \alpha L X_t\big) + Tr\big(X_t^T (1-\alpha) I X_t - X_t^T (1-\alpha) X_{t-1}X_{t-1}^T X_t\big). \end{aligned}$$

Evolutionary Clustering

- Assuming Normalized Laplacian is used

$$L = I - D_t^{-1/2} A_t D_t^{-1/2}$$

$$\begin{aligned} Cost_t &= Tr\big(X_t^T \alpha\big(I - D_t^{-1/2} A_t D_t^{-1/2}\big)\, X_t\big) \\ &\quad + Tr\big(X_t^T\, (1-\alpha)\, I\, X_t - X_t^T\, (1-\alpha)\, X_{t-1}\, X_{t-1}^T\, X_t\big), \\ &= Tr\big(X_t^T\, \big(I - \alpha D_t^{-1/2} A_t D_t^{-1/2} - (1-\alpha)\, X_{t-1}X_{t-1}^T\big)\, X_t\big), \\ &= Tr(X_t \hat{L} X_t) \end{aligned}$$

$$\hat{L} = I - \alpha D_t^{-1/2} A_t D_t^{-1/2} - (1-\alpha) X_{t-1}X_{t-1}^T$$

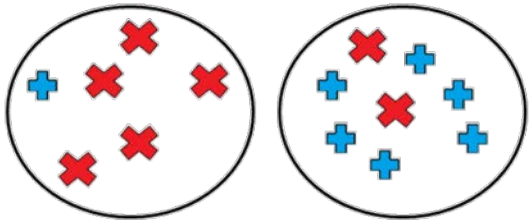
Similar to Spectral Clustering
X_t can be obtained by taking the top eigenvectors of $\hat{L}$

Community Evaluation

Evaluating the Communities

We are given objects of two different kinds (+, ×)

- **The perfect community:** all objects inside the community are of the same type



- **Evaluation with ground truth**
- **Evaluation without ground truth**

Evaluation with Ground Truth

- When ground truth is available
  - We have partial knowledge of what communities should look like
  - We are given the correct community (clustering) assignments

- **Measures**
  - Precision and Recall, or F-Measure
  - Purity
  - Normalized Mutual Information (NMI)

Precision and Recall

$$Precision = \frac{Relevant\ and\ retrieved}{Retrieved}$$

$$P = \frac{TP}{TP + FP}$$

$$Recall = \frac{Relevant\ and\ retrieved}{Relevant}$$

$$R = \frac{TP}{TP + FN}$$

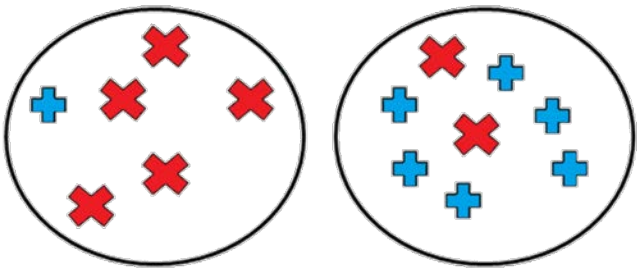
True Positive (TP) :
- When similar members are assigned to the same communities
- A **correct** decision.

False Negative (FN) :
- When similar members are assigned to different communities
- An **incorrect** decision

True Negative (TN) :
- When dissimilar members are assigned to different communities
- A **correct** decision

False Positive (FP) :
- When dissimilar members are assigned to the same communities
- An **incorrect** decision

Precision and Recall: Example



$$TP = \binom{5}{2} + \binom{6}{2} + \binom{2}{2} = 26,$$
$$FP = (5 \times 1) + (6 \times 2) = 17,$$
$$FN = (5 \times 2) + (6 \times 1) = 16,$$
$$TN = (6 \times 5) + (2 \times 1) = 32.$$

$$P = \frac{26}{26+17} = 0.60$$

$$R = \frac{26}{26+16} = 0.61$$

F-Measure

Either P or R measures one aspect of the performance,

- To integrate them into one measure, we can use the harmonic mean of precision of recall

$$F = 2 \cdot \frac{P \cdot R}{P+R}$$

For the example earlier,

$$F = 2 \times \frac{0.6 \times 0.61}{0.6 + 0.61} = 0.60$$

Purity

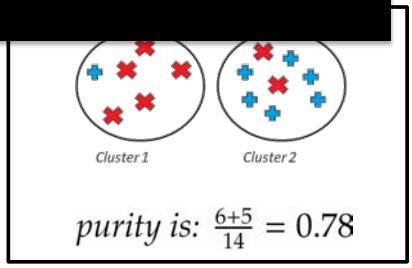
We can assume the majority of a community represents the community

- We use the label of the majority against the label of each member to evaluate the communities

Purity can be easily tampered by

- Points being singleton communities (of size 1); or by
- Very large communities

$$Purity = \frac{1}{N}\sum_{i=1} \max_j |C_i \cap L_j|$$



- k: the number of communities
- N: total number of nodes,
- L_j: the set of instances with label j in all communities
- C_i: the set of members in community i

Mutual Information

- **Mutual information (MI).** The amount of information that two random variables share.
  - By knowing one of the variables, it measures the amount of uncertainty reduced regarding the others

$$MI = I(H, L) = \sum_{h \in H} \sum_{l \in L} \frac{n_{h,l}}{n} \log \frac{n \cdot n_{h,l}}{n_h n_l}$$

- L and H are labels and found communities;
- n_h and n_l are the number of data points in community h and with label l, respectively;
- $n_{h,l}$ is the number of nodes in community h and with label l; and n is the number of nodes

Normalizing Mutual Information (NMI)

- Mutual information (MI) is unbounded
- To address this issue, we can normalize MI
- How? We know that

$$MI \leq min(H(L), H(H)),$$
$$(MI)^2 \leq H(H)H(L).$$
$$MI \leq \sqrt{H(H)}\sqrt{H(L)}.$$

- $H(.)$ is the entropy function

$$H(L) = -\sum_{l \in L} \frac{n_l}{n} \log \frac{n_l}{n}$$
$$H(H) = -\sum_{h \in H} \frac{n_h}{n} \log \frac{n_h}{n}.$$

Normalized Mutual Information

Normalized Mutual Information

$$NMI = \frac{MI}{\sqrt{H(L)}\sqrt{H(H)}}.$$

$$NMI = \frac{\sum_{h \in H} \sum_{l \in L} n_{h,l} \log \frac{n \cdot n_{h,l}}{n_h n_l}}{\sqrt{(\sum_{h \in H} n_h \log \frac{n_h}{n})(\sum_{l \in L} n_l \log \frac{n_l}{n})}}.$$

We can also define it as

Note that $MI < 1/2(H(H) + H(L)$

$$NMI = \frac{I(H;L)}{\frac{1}{2}(H(L) + H(H))}$$

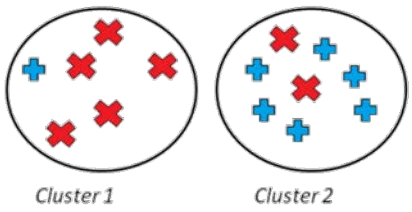
Normalized Mutual Information

$$NMI = \frac{\sum_{h,l} n_{h,l} \log \frac{n \cdot n_{h,l}}{n_h n_l}}{\sqrt{(\sum_h n_h \log \frac{n_h}{n})(\sum_l n_l \log \frac{n_l}{n})}}$$

- where l and h are known (with labels) and found communities, respectively
- n_h and n_l are the number of members in the community h and l, respectively,
- $n_{h\ l}$ is the number of members in community h and labeled l,
- n is the size of the dataset

- **NMI** values close to one indicate high similarity between communities found and labels
- Values close to zero indicate high dissimilarity between them

Normalized Mutual Information: Example



Found communities (H)
- [1,1,1,1,1,1, 2,2,2,2,2,2,2,2]

Actual Labels (L)
- [2,1,1,1,1,1, 2,2,2,2,2,2,1,1]

$n = 14$

	n_h
h=1	6
h=2	8

	n_l
$l = 1$	7
$l = 2$	7

$n_{h,l}$	$l = 1$	$l = 2$
h=1	5	1
h=2	2	6

Evaluation without Ground Truth

(a) U.S . Constitution

(b) Sports

- **Evaluation with Semantics**
  - A simple way of analyzing detected communities is to analyze other attributes (posts, profile information, content generated, etc.) of community members to see if there is a coherency among community members
  - The coherency is often checked via human subjects.
    - Or through labor markets: Amazon Mechanical Turk
  - To help analyze these communities, one can use word frequencies. By generating a list of frequent keywords for each community, human subjects determine whether these keywords represent a coherent topic.

- **Evaluation Using Clustering Quality Measures**
  - Use clustering quality measures (SSE)
  - Use more than two community detection algorithms and compare the results and pick the algorithm with better quality measure