



Community Detection

CE642: Social and Economic Networks
Maryam Ramezani
Sharif University of Technology
maryam.ramezani@sharif.edu

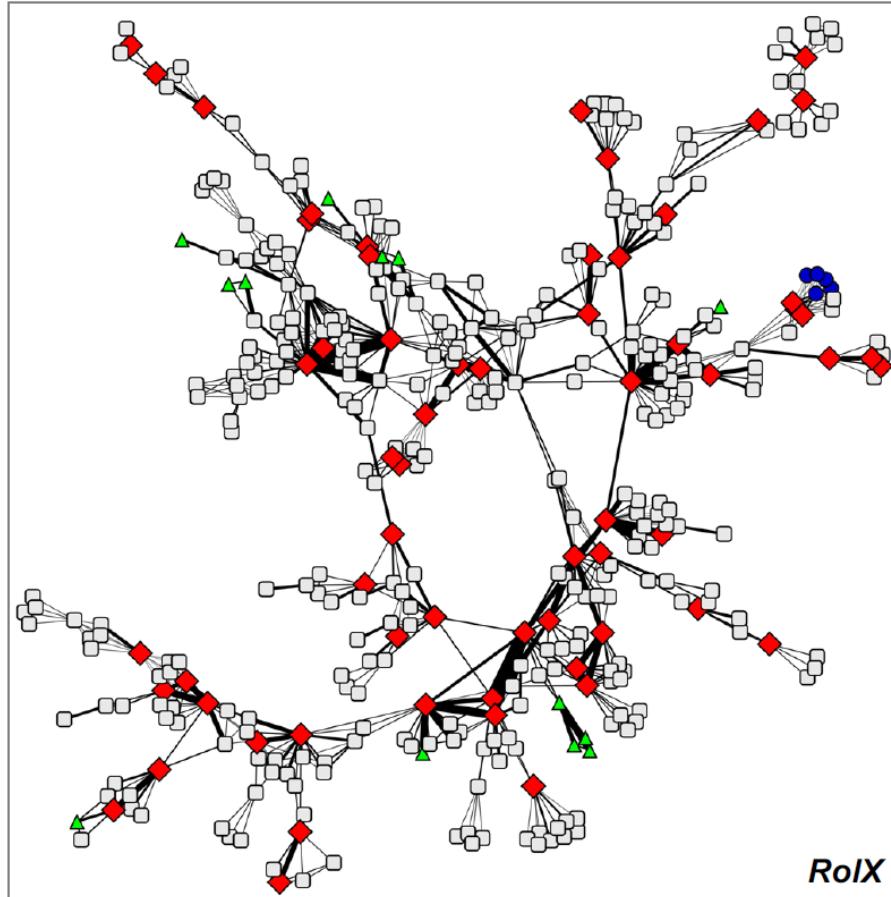


01

Introduction

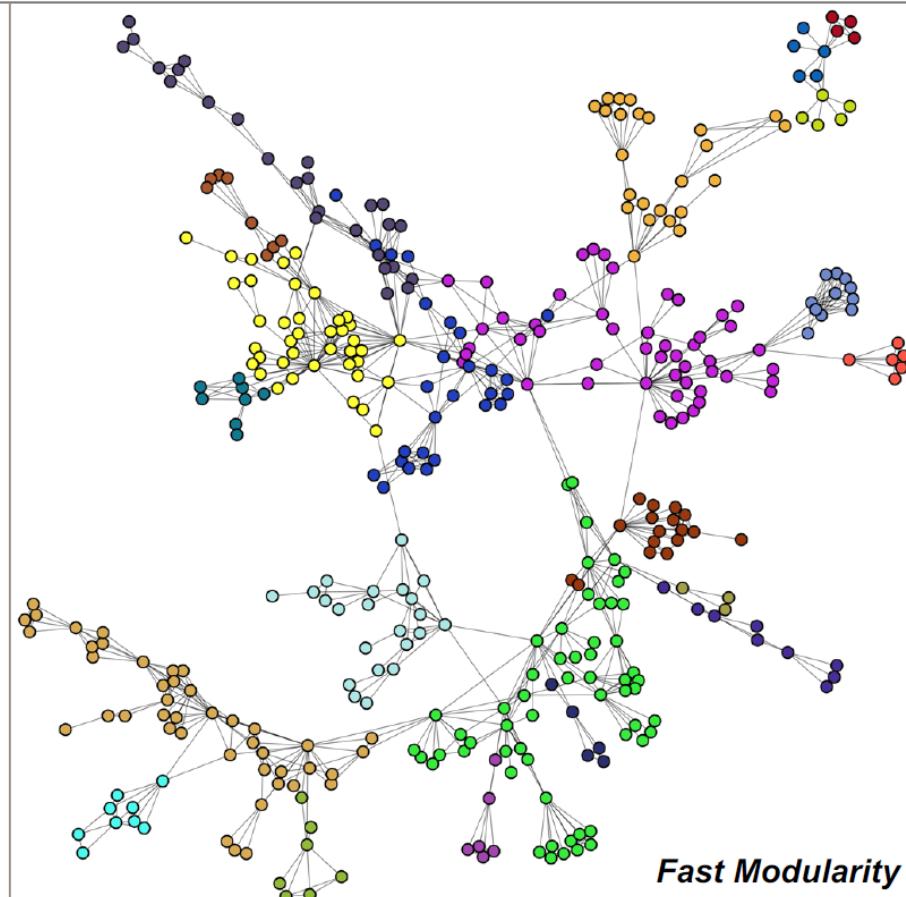
Roles vs Communities

Last Lecture: Roles



RoIX

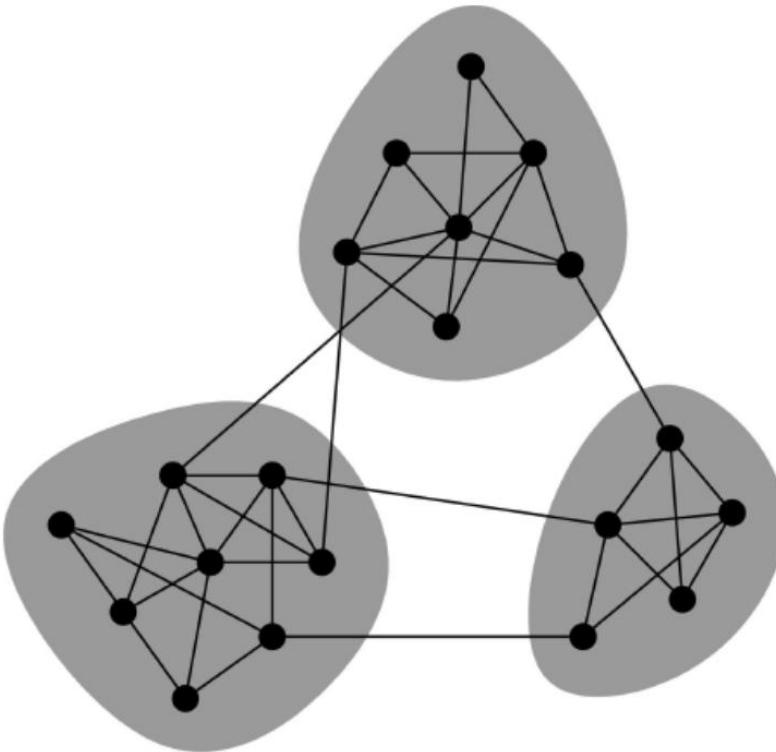
This Lecture: Communities



Fast Modularity

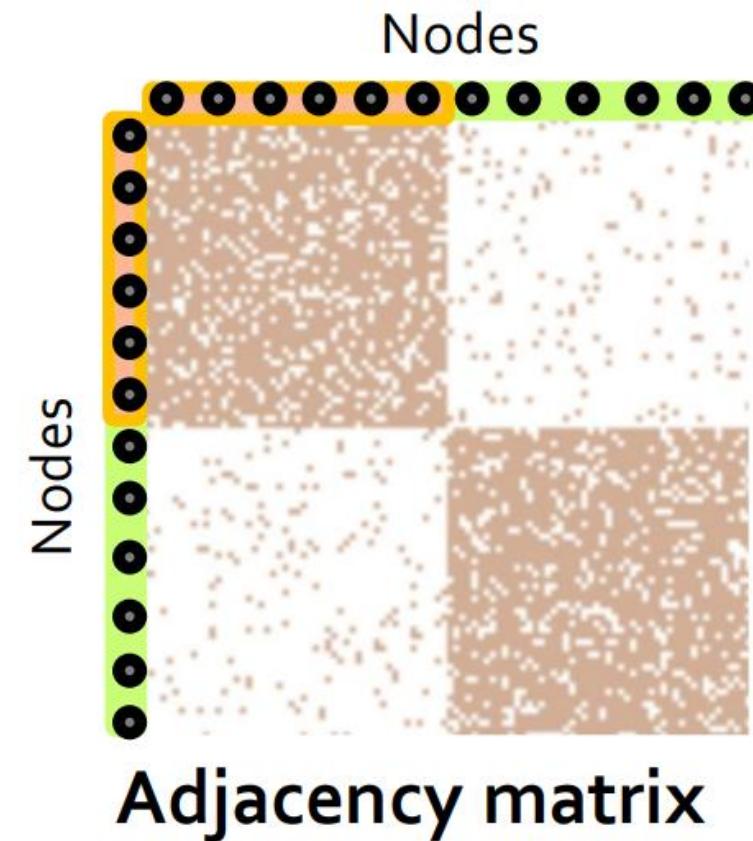
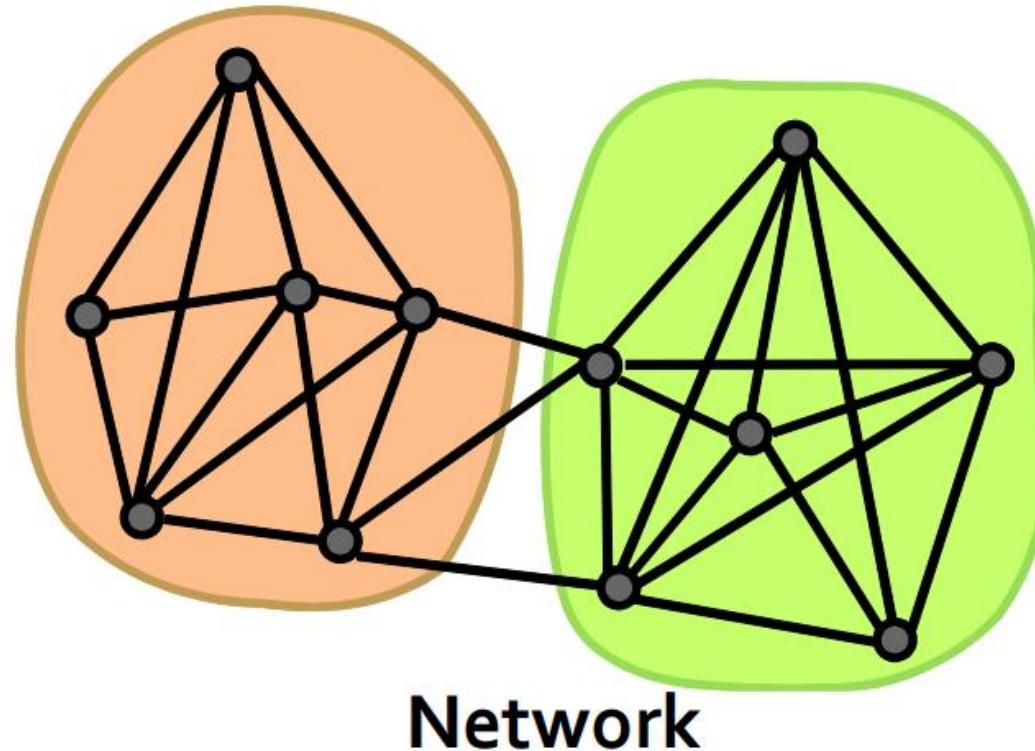
Network & Communities

- We often think of networks “looking” like this:



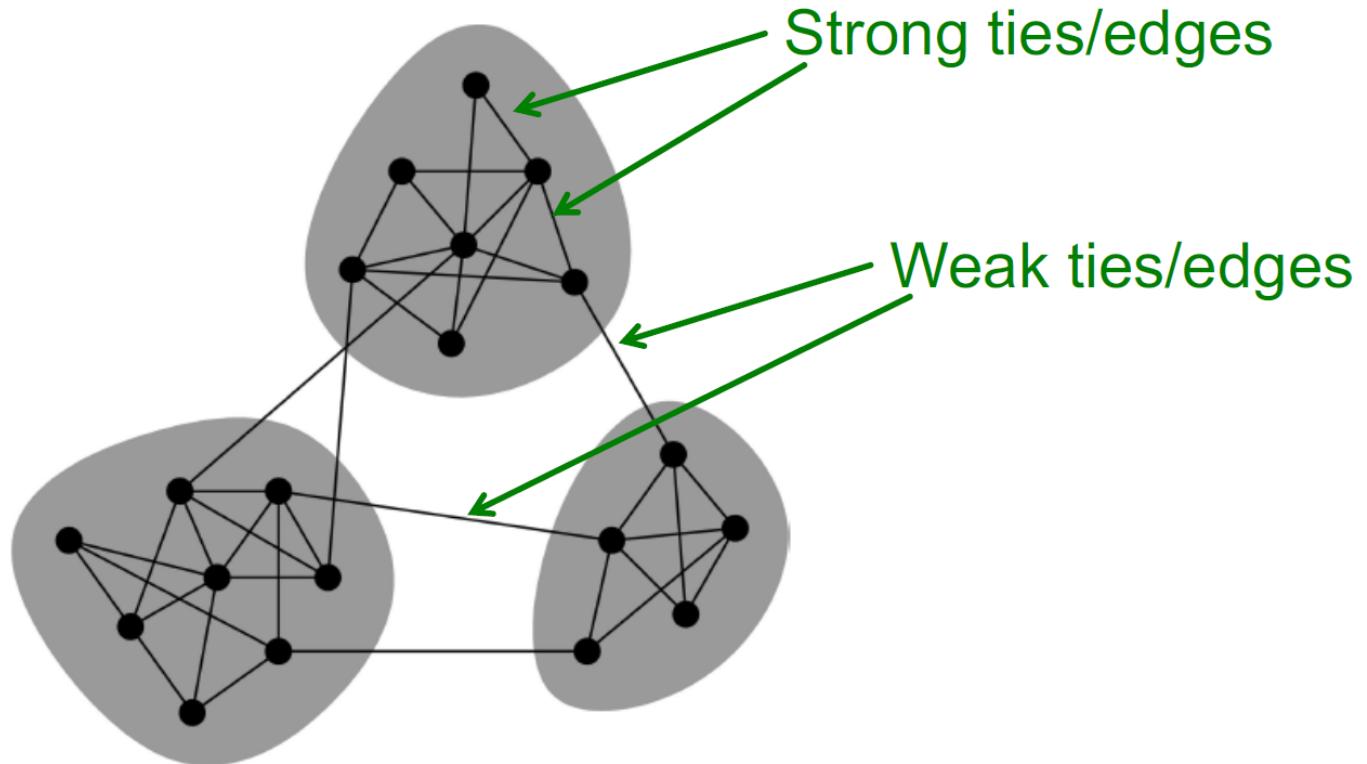
- What led to such a conceptual picture?

Non-overlapping Clusters



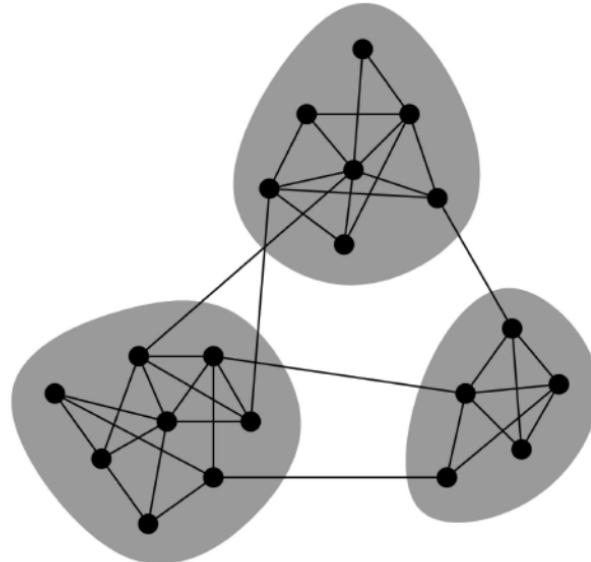
Network & Communities

- Granovetter's theory leads to the following conceptual picture of networks



Network & Communities

- Granovetter's theory suggests that networks are composed of **tightly connected sets of nodes**



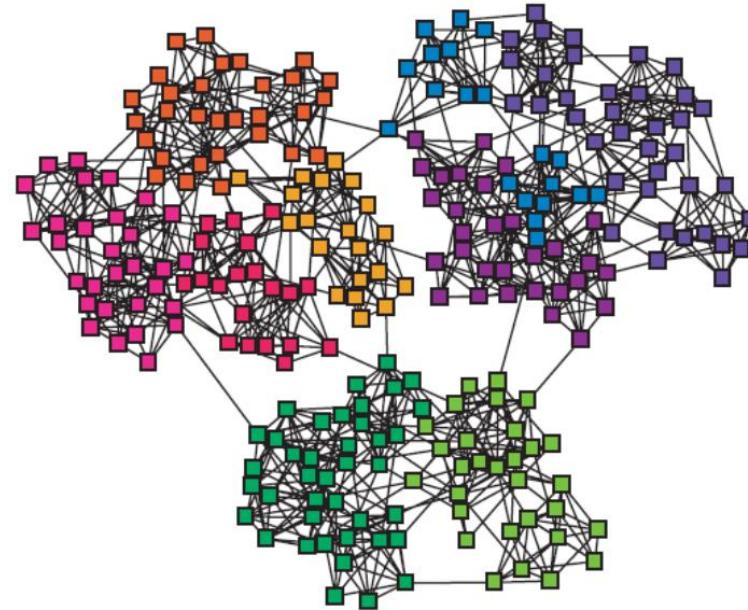
Communities, clusters,
groups, modules

- **Network communities:**

- Sets of nodes with **lots of internal** connections and **few external** ones (to the rest of the network).

Finding Network Communities

- How do we automatically find such densely connected groups of nodes?
- Ideally such automatically detected clusters would then correspond to real groups
- For example:

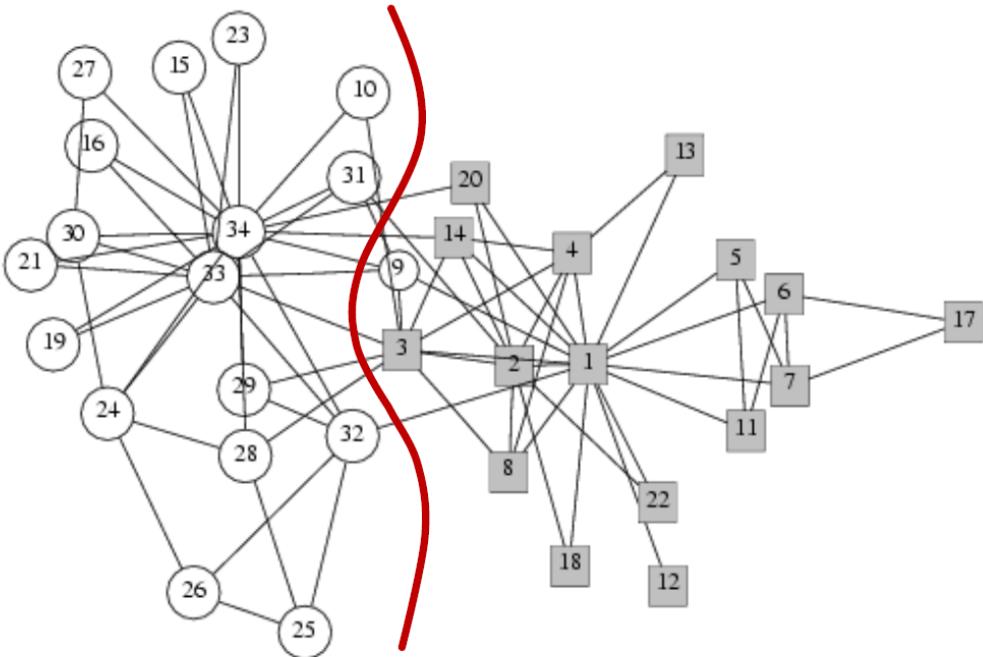


Communities, clusters,
groups, modules

Community Detection

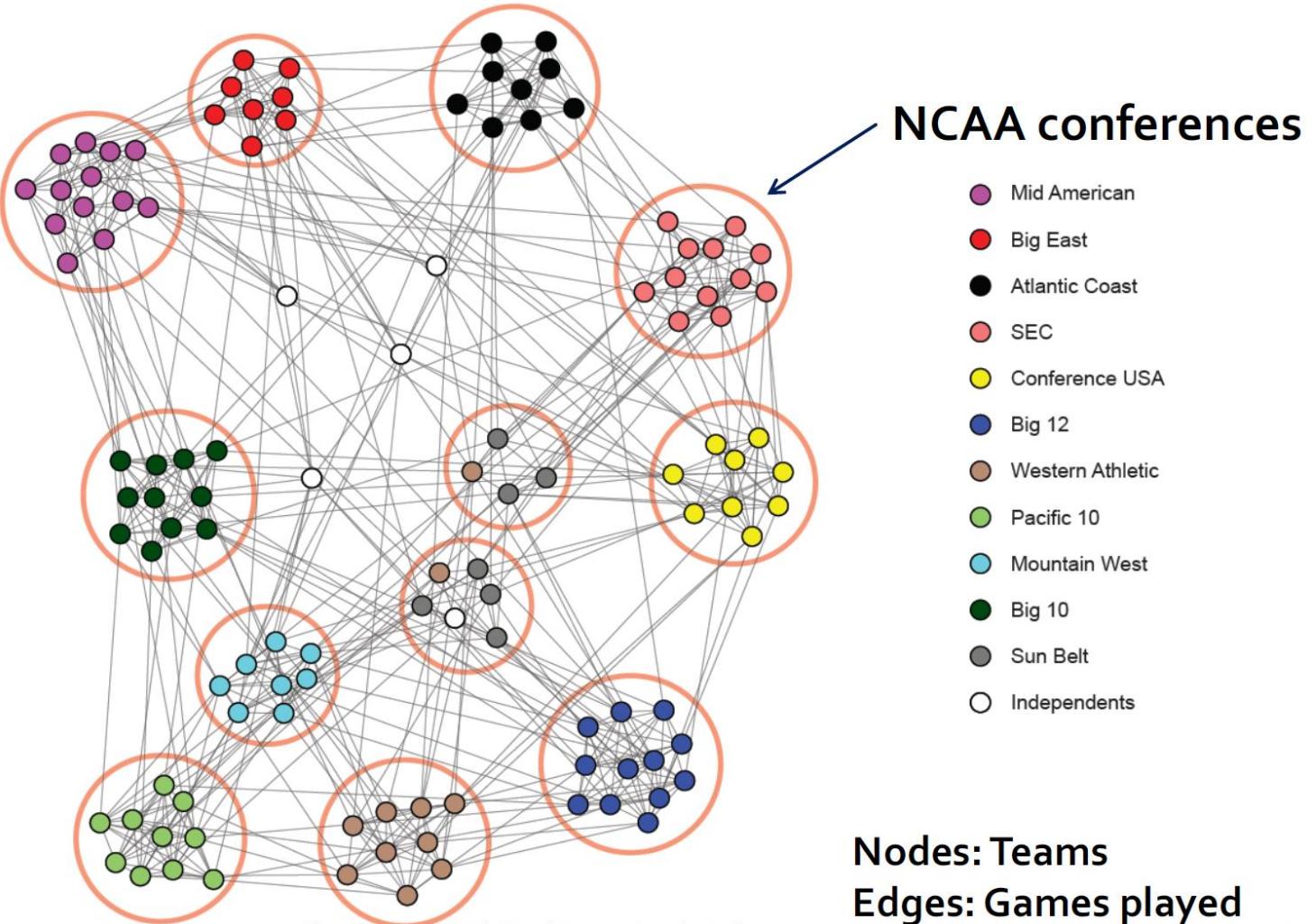
- Given a network: What are the “communities”?
 - Closely connected groups of nodes
 - Relatively few edges to outside the community
- Similar to clustering in datasets
 - Group together points that are more close or similar to each other than other points

Social Network Data



- **Zachary's Karate club network:**
 - Observed social ties & rivalries in a university karate club
 - During the study, conflicts led the group to split
 - Split could be explained by a minimum cut in the network

Example

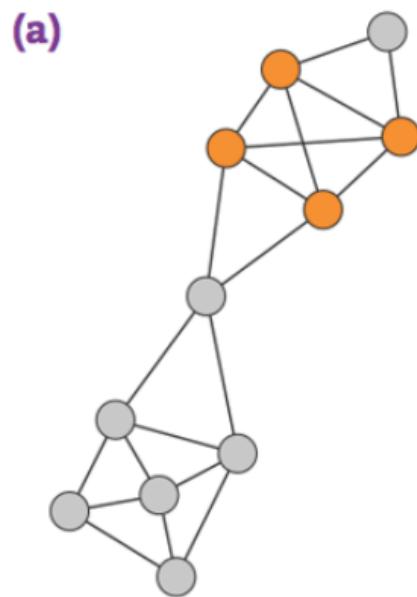


Intra & Inter Community Links

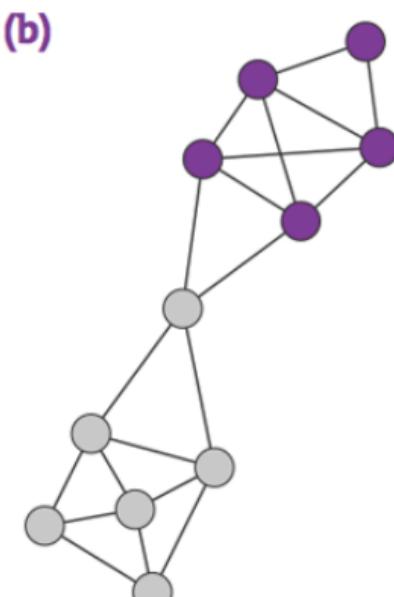
- **Intra-community links** that connect nodes in the same community. These nodes interact frequently; therefore, these links are known as strong ties. The nodes that only have links to other nodes in their community are called core nodes. The core nodes tend to influence their cohorts in a community more than other network members.
- **Inter-community links** which connect nodes of different communities. Due to low interactions between such nodes, they are called weak ties. Nodes that have at least one inter-community link are called boundary nodes. Such nodes play an important role in information dissemination over different communities.

Strong and Weak Communities

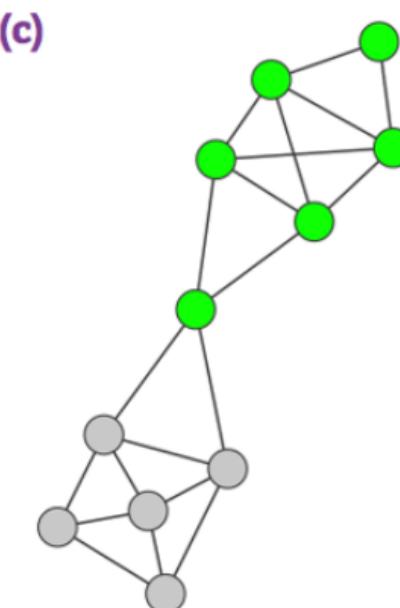
In a *strong community* each node of C has more links within the community than with the rest of the graph, $k_i^{int}(C) > k_i^{ext}(C)$. In a *weak community*, the total internal degree of C exceeds its total external degree, $k^{in}(C) > k^{out}(C)$.



Clique



Strong



Weak



02

Applications in Various Fields

Cybersecurity and Fraud Detection

- Networks representing user interactions or system communications can be monitored for unusual community behavior that might indicate a security breach or coordinated attack.
 - Intrusion Detection: By establishing baseline community structures in a network, deviations can flag potential intrusions.
 - Fraud Detection: In financial transactions, community detection can uncover suspicious clusters of activity that diverge from normative behavior, hinting at fraudulent networks.

Applications in Social Network Analysis and Biology

- **Marketing and Influence Analysis:** Brands can target influential communities within social media networks to optimize advertising campaigns.
- **Epidemiological Studies:** Understanding how individuals are grouped can help trace the spread of diseases or misinformation, supporting public health initiatives.
- **Gene Regulatory Networks:** Identifying communities within complex gene interaction networks can shed light on co-expressed gene clusters, aiding in understanding of genetic conditions.



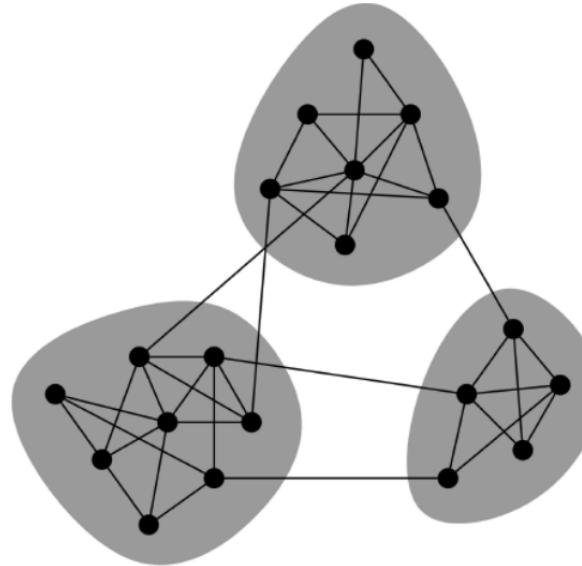
03

Modularity Metric

Network Communities

- **Communities:** sets of **tightly connected nodes**
- Define: **Modularity Q**
 - A measure of how well a network is partitioned into communities
 - Given a **partitioning** of the network into groups disjoint $s \in S$:

$$Q \propto \sum_{s \in S} [(\# \text{ edges within group } s) - \underbrace{(\text{expected } \# \text{ edges within group } s)}_{\text{Need a null model}}]$$



Network Communities

- Random networks are supposed without modular structure
- A random network with the same degree distribution
- We may compare the modular structure of the network with that of a random network (Newman and Girvan 2002)

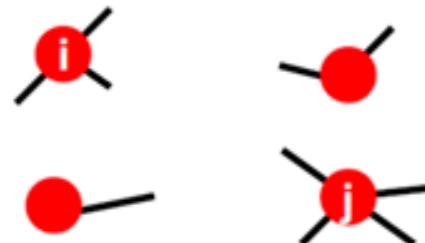
$$Q = \frac{1}{2m} \sum_{ij} (A_{ij} - P_{ij}) \delta(C_i, C_j) = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(C_i, C_j)$$

- m: number of edges
- A: adjacency matrix
- P: expected number of edges between the nodes in the random network
- The δ -function yields one if i and j are in the same community ($C_i = C_j$), zero otherwise
- k_i : degree of node i

Null Model Configuration

- Given real G on n nodes and m edges, construct rewired network G'

- Same degree distribution but uniformly random connections



- Consider G' as a multigraph

- The expected number of edges between nodes

i and j of degrees k_i and k_j equals: $k_i \cdot \frac{k_j}{2m} = \frac{k_i k_j}{2m}$

Null Model Configuration (Proof the formula)

- Null Model Configuration is a multigraph which means we can have more than one edge between each pairs of nodes.
- Probability that an edge from i connects directly to one possible side of node j because there are $2m-1$ possible choices (after picking one edge already): $\frac{k_j}{2m-1}$
 - If m is large, then $2m - 1 \approx 2m$
- Number of "opportunities" of node i : $k_i \cdot \frac{k_j}{2m}$
- Expected number of successes=number of trials \times success probability

$$\mathbb{E}[A_{ij}] = k_i \times \frac{k_j}{2m} = \frac{k_i k_j}{2m}$$

Null Model Configuration

- The expected number of edges in (multigraph) \mathbf{G}' :

- $= \frac{1}{2} \sum_{i \in N} \sum_{j \in N} \frac{k_i k_j}{2m} = \frac{1}{2} \cdot \frac{1}{2m} \sum_{i \in N} k_i (\sum_{j \in N} k_j) =$

- $= \frac{1}{4m} 2m \cdot 2m = m$

Note:

$$\sum_{u \in N} k_u = 2m$$

● What is $\mathbb{E}[A_{ij}]$ exactly?

- A_{ij} is a random variable:
 - $A_{ij} = 1$ if there is an edge between node i and node j ,
 - $A_{ij} = 0$ if there is no edge.

Thus, the expected value $\mathbb{E}[A_{ij}]$ is:

$$\mathbb{E}[A_{ij}] = 1 \times \Pr(A_{ij} = 1) + 0 \times \Pr(A_{ij} = 0)$$

Which simplifies to:

$$\mathbb{E}[A_{ij}] = \Pr(A_{ij} = 1)$$

- ✓ So the expected value of A_{ij} is exactly the probability that there is an edge between i and j .

● Now: Can we say "expected number = number \times probability"?

In general, YES:

If you have a number of opportunities and each opportunity succeeds independently with some probability, then:

Expected number of successes = number of trials \times success probability

But careful!

In this specific case (Configuration Model):

- We are pairing stubs randomly.
- Node i has k_i stubs,
- Node j has k_j stubs.

Each stub from i has a chance of connecting to any stub from j .

Thus:

- ✓ Number of "opportunities" = k_i stubs from i ,
- ✓ Probability that a stub connects to j $\approx \frac{k_j}{2m}$.

Therefore, the expected number of connections is:

$$\mathbb{E}[A_{ij}] = k_i \times \frac{k_j}{2m} = \frac{k_i k_j}{2m}$$

Modularity

■ Modularity of partitioning S of graph G :

- $Q \propto \sum_{s \in S} [(\# \text{ edges within group } s) - (\text{expected } \# \text{ edges within group } s)]$

- $$Q(G, S) = \frac{1}{2m} \sum_{s \in S} \sum_{i \in s} \sum_{j \in s} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$$

Normalizing const.: $-1 \leq Q \leq 1$

$A_{ij} = 1$ if $i \rightarrow j$,
0 otherwise

■ Modularity values take range $[-1, 1]$

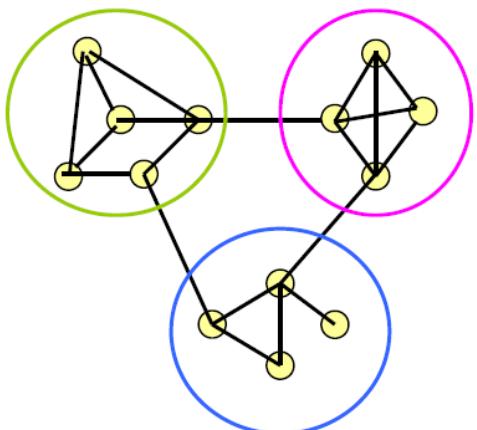
- It is positive if the number of edges within groups exceeds the expected number
- Q greater than 0.3-0.7 means **significant community structure**

Another view of Modularity

- Groups of vertices within which connections are dense, but between which connections are sparser.
 - Because we don't have any prior information about network.

Modularity

$$Q = \sum_i e_{ii} - \sum_{ijk} e_{ij} e_{ki} = \sum_i \left(e_{ii} - \left(\sum_j (e_{ij}) \right)^2 \right) = \text{Tr}(e) - \|e^2\|$$



e_{ij} : the fraction of edges in the original network connecting nodes in community i to those in community j
Note that the above representation is another form of modularity presented in the previous slide

Another view of Modularity

■ Proof

- e_{ij} = fraction of all edges that go from community i to community j .
 - So:

$$e_{ij} = \frac{\text{number of edges between communities } i \text{ and } j}{2m}$$

- a_i = total fraction of edges connected to community i :

$$a_i = \sum_j e_{ij}$$

which simply means: add up all fractions of edges going from community i to all communities j (including i itself).

Another view of Modularity

- Suppose **edges are placed randomly, but the number of edges touching each community stays the same** (this is called a "null model").
 - Community i "owns" fraction a_i of all edges (half-edges or stubs).
 - The probability that one end of a random edge falls into community $i = a_i$.
 - The probability that both ends of a random edge fall into community $i = a_i \times a_i = a_i^2$.
 - Real** fraction of edges **inside** community $i = e_{ii}$
 - Expected** fraction (if random) = a_i^2
For community i , the contribution to modularity is:

$$e_{ii} - a_i^2$$

And the total modularity Q is:

$$Q = \sum_i (e_{ii} - a_i^2)$$

Social and Economic Networks

Modularity

$$Q(G, S) = \frac{1}{2m} \sum_{s \in S} \sum_{i \in s} \sum_{j \in s} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$$

Equivalently modularity can be written as:

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$$

- A_{ij} represents the edge weight between nodes i and j ;
- k_i and k_j are the sum of the weights of the edges attached to nodes i and j , respectively;
- $2m$ is the sum of all of the edge weights in the graph;
- c_i and c_j are the communities of the nodes; and
- δ is an indicator function $\delta(c_i, c_j) = 1$ if $c_i = c_j$ else 0

Idea: We can identify communities by maximizing modularity

Modularity Limitations

- Resolution Limit:
 - Modularity may miss small communities, especially in large sparse graphs.
 - It tends to merge small but meaningful groups into larger ones because that gives a higher modularity score overall.
- Degeneracy:
 - There can be many partitions with similar modularity scores – especially in sparse networks – making it hard to pick the "best" community structure.
- Random graph bias:
 - In very sparse graphs, the null model (which assumes random connection probability based on node degree) may be too weak, so modularity might find communities even in random noise.



04

Normalized Mutual Information (NMI) Metric

Entropy

- $H(X)$: is called the Entropy of X.
- It measures the uncertainty or randomness in the random variable.
 - Uncertainty means how much you don't know about the outcome before it happens.
 - In information theory, entropy measures uncertainty.
 - If you are very unsure about what will happen, entropy is high.
 - If you are very sure (the outcome is predictable), entropy is low.

Individual Entropy

Examples:

- Suppose you toss a fair coin:

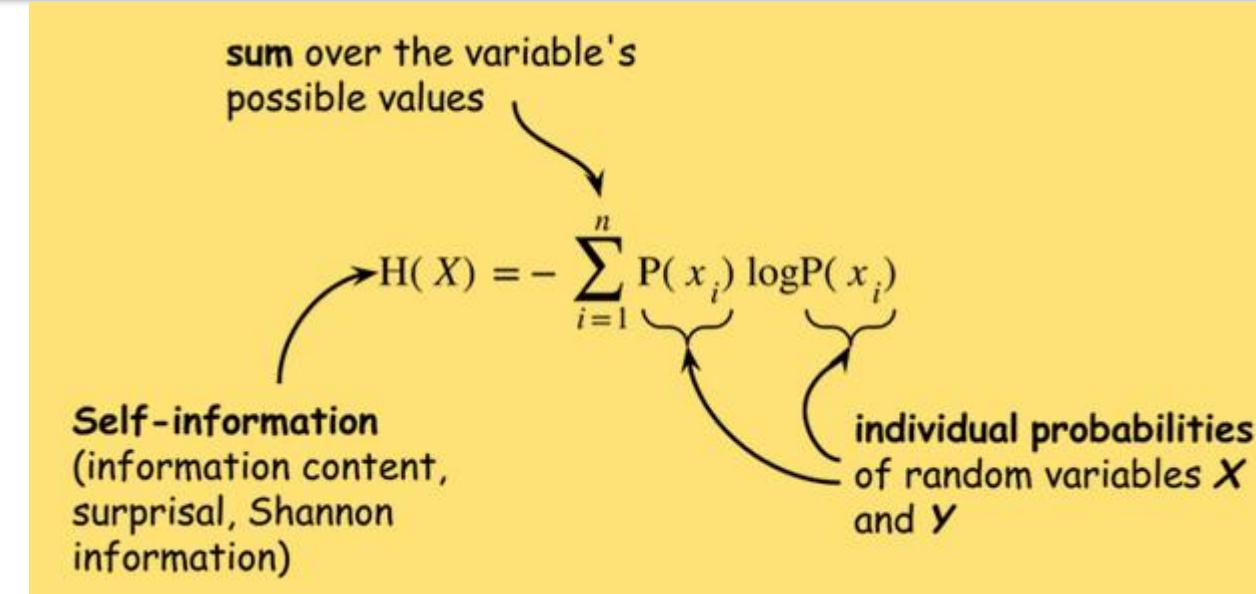
- Heads (H) or Tails (T)
- Probability of Heads = 0.5
- Probability of Tails = 0.5

$$H(\text{Coin}) = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 1$$

- Now imagine a trick coin that always lands on Heads.

- Heads (H): probability = 1
- Tails (T): probability = 0

$$H(\text{Broken Coin}) = -(1 \log_2 1 + 0 \log_2 0) = 0$$



More outcomes → more uncertainty → higher entropy.

Joint Entropy

2 random variables
 X and Y

discrete case

$$H(X, Y) = - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) \log_2[P(x, y)]$$

particular values
of X and Y

joint probability of
values occurring
together

continuous case

$$h(X, Y) = - \int_{\mathcal{X}, \mathcal{Y}} f(x, y) \log f(x, y) dx dy$$

Motivation for Mutual Information

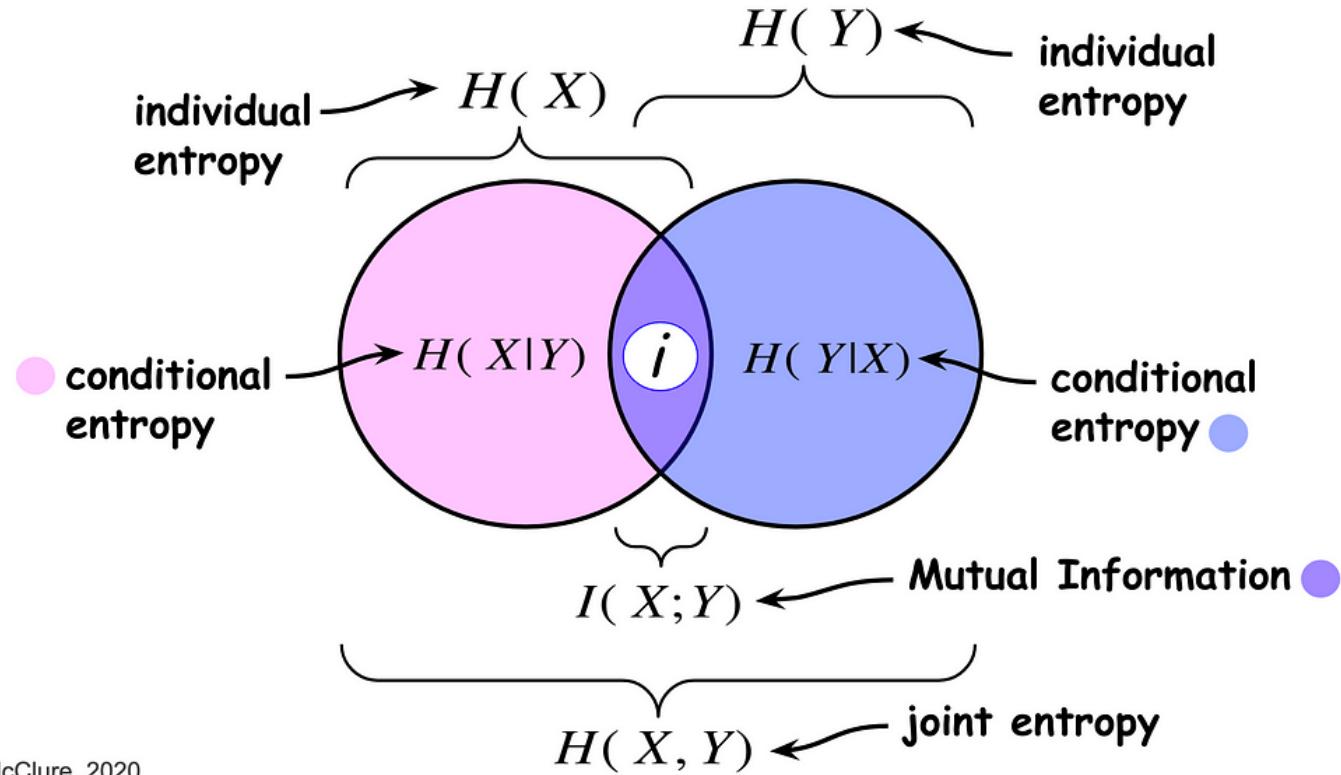
- Mutual information measures the information that X and Y share. It measures how much knowing one of these variables reduces uncertainty about the other.
 - If X and Y are independent:
 - $MI(X,Y)=0$
 - If X is a deterministic function of Y and Y is a deterministic function of X then all information conveyed by X is shared with Y

Mutual Information

- $H(X) + H(Y)$ = the uncertainty if we knew nothing about how X and Y are related (just sum of individual uncertainties).
- $H(X, Y)$ = the actual uncertainty when considering X and Y together.
- The difference between these two is the shared information – that is, Mutual Information.

$$MI = H(X) + H(Y) - H(X, Y)$$

Motivation for MI



$$\text{Mutual Information: } I(X;Y) = H(X) + H(Y) - H(X,Y)$$

MI for two Communities

1

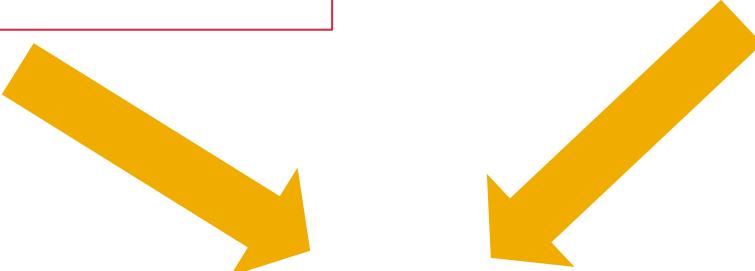
$$H(C_A) = - \sum_i P(i) \log P(i)$$

$$H(C_B) = - \sum_j P(j) \log P(j)$$

$$H(C_A, C_B) = - \sum_i \sum_j P(i, j) \log P(i, j)$$

2

$$MI = H(X) + H(Y) - H(X, Y)$$



$$MI(C_A, C_B) = \left(- \sum_i P(i) \log P(i) - \sum_j P(j) \log P(j) \right) + \left(\sum_i \sum_j P(i, j) \log P(i, j) \right)$$

MI for two Communities

$$P(i) = \sum_j P(i, j)$$



$$\begin{aligned} \sum_i P(i) \log P(i) &= \sum_i \left(\sum_j P(i, j) \right) \log P(i) \\ &= \sum_i \sum_j P(i, j) \log P(i) \end{aligned}$$

$$MI(C_A, C_B) = \sum_i \sum_j P(i, j) \log P(i, j) - \sum_i \sum_j P(i, j) \log P(i) - \sum_i \sum_j P(i, j) \log P(j)$$

$$MI(C_A, C_B) = \sum_i \sum_j P(i, j) \boxed{(\log P(i, j) - \log P(i) - \log P(j))} \quad \log \left(\frac{P(i, j)}{P(i)P(j)} \right)$$

$$MI(C_A, C_B) = \sum_{i=1}^{|C_A|} \sum_{j=1}^{|C_B|} P(i, j) \log \left(\frac{P(i, j)}{P(i)P(j)} \right)$$

Normalized Mutual Information

- The Normalized Mutual Information is used to scale MI between 0 and 1. It's symmetric and normalized.

$$NMI(C_A, C_B) = \frac{2 \times MI(C_A, C_B)}{H(C_A) + H(C_B)}$$

$P(i)$ = probability that a sample belongs to cluster i in C_A

$P(j)$ = probability that a sample belongs to cluster j in C_B

$P(i, j)$ = probability that a sample belongs **both** to cluster i in C_A **and** cluster j in C_B

Normalized Mutual Information

$$MI(C_A, C_B) = \sum_{i=1}^{|C_A|} \sum_{j=1}^{|C_B|} P(i, j) \log \left(\frac{P(i, j)}{P(i)P(j)} \right)$$

$$NMI(C_A, C_B) = \frac{2 \times MI(C_A, C_B)}{H(C_A) + H(C_B)}$$

$P(i)$ = probability that a sample belongs to cluster i in C_A

$P(j)$ = probability that a sample belongs to cluster j in C_B

$P(i, j)$ = probability that a sample belongs **both** to cluster i in C_A **and** cluster j in C_B

Normalized Mutual Information

N = total number of samples

N_{ij} = number of samples simultaneously in cluster i in C_A and cluster j in C_B

$N_{i\cdot}$ = number of samples in cluster i in C_A

$N_{\cdot j}$ = number of samples in cluster j in C_B

$$P(i) = \text{probability that a sample belongs to cluster } i \text{ in } C_A \longrightarrow P(i) = \frac{N_{i\cdot}}{N}$$

$$P(j) = \text{probability that a sample belongs to cluster } j \text{ in } C_B \longrightarrow P(j) = \frac{N_{\cdot j}}{N}$$

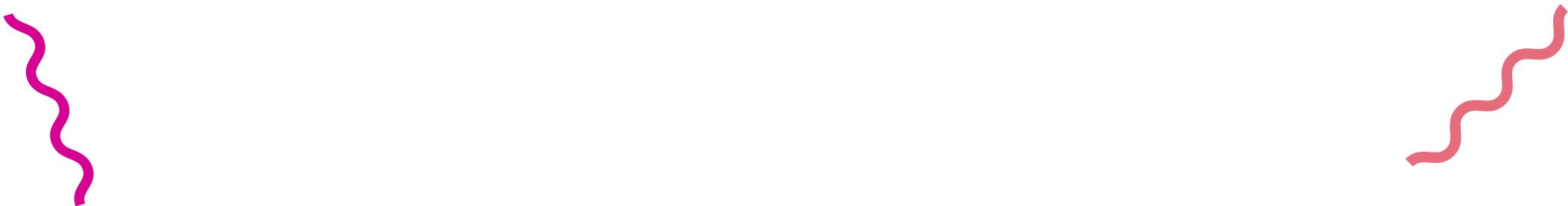
$$P(i, j) = \text{probability that a sample belongs \textbf{both} to cluster } i \text{ in } C_A \text{ and cluster } j \text{ in } C_B \longrightarrow P(i, j) = \frac{N_{ij}}{N}$$

Normalized mutual information

- NMI is a probabilistic information theoretical metric that compares the similarity of two sets of clusters.

$$NMI(C_A, C_B) = \frac{-2 \sum_{i=1}^{|C_A|} \sum_{j=1}^{|C_B|} N_{ij} \log\left(\frac{N_{ij} N_{..}}{N_{i.} N_{.j}}\right)}{\sum_{i=1}^{|C_A|} N_{i.} \log\left(\frac{N_{i.}}{N_{..}}\right) + \sum_{j=1}^{|C_B|} N_{.j} \log\left(\frac{N_{.j}}{N_{..}}\right)}$$

Where C_A and C_B are two sets of clusters. N is a $|C_A| \times |C_B|$ matrix in which N_{ij} is equal to the number of common members of i 'th cluster of C_A and j 'th cluster of C_B . $N_{i.}$, $N_{.j}$, and $N_{..}$ are equal to the sum of i 'th row, j 'th column, and the whole of the matrix N , respectively. The range of NMI values is between 0 and 1, where NMI of 1 represents the exact matching of two sets, and 0 shows the maximal difference.



05

Classification Metric

Pairwise F1-score

- PWF: Consider a pair of nodes ν and ν . These two nodes can be in same or different community in the ground truth and the inferred network.
 - Consider the classification problem where we want to determine whether nodes ν and ν are in the same community or not.

$$PW_{precision} = \frac{|H_G \cap H_{G^*}|}{|H_{G'}|}$$

$$PW_{recall} = \frac{|H_G \cap H_{G^*}|}{|H_{G^*}|}$$

$$PWF = \frac{2PW_{precision}PW_{recall}}{PW_{precision} + PW_{recall}}$$



06

Graph Partition -based Metrics

Conductance

- Graph Conductance:

$$\varphi(G) = \min_U \frac{E(U, V - U)}{\min\{d(U), d(V - U)\}}$$

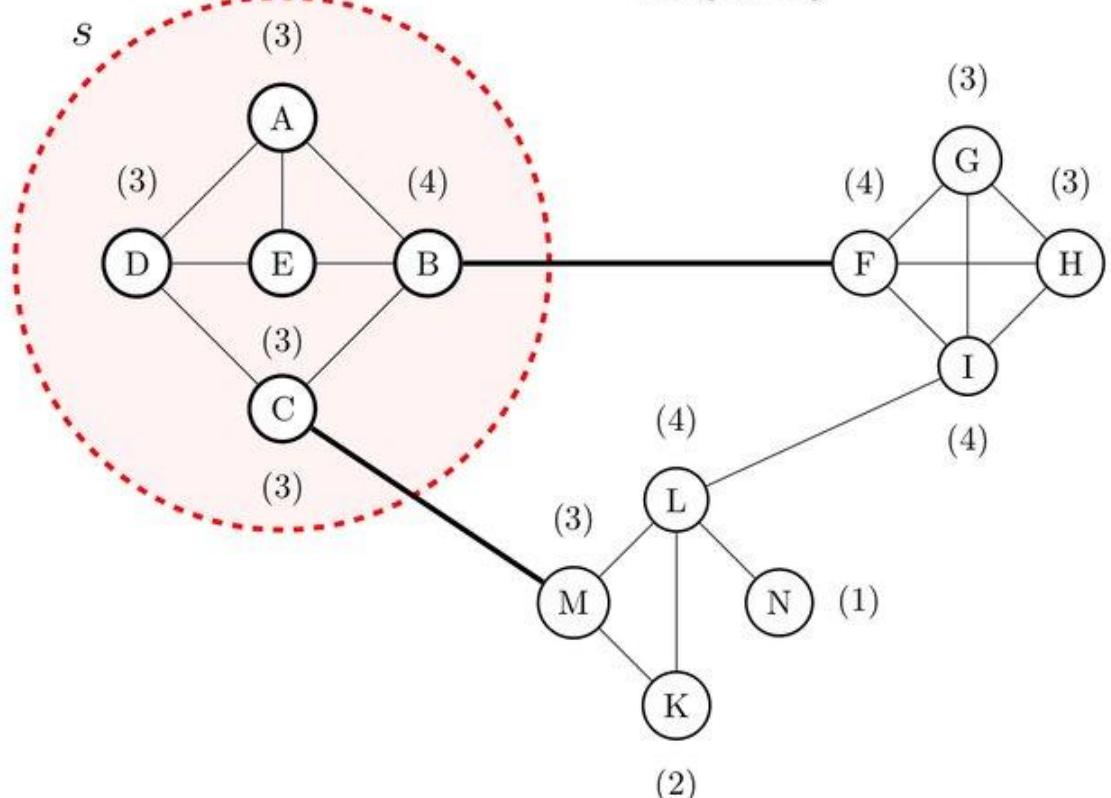
- $d(U) = \sum_{i \in U} \sum_{j \in U} A[i, j]$ is weighted degrees of nodes in U
- Consider the normalized stochastic matrix $M = D^{-1}A$, where A is the adjacency matrix and D the diagonal degrees
 - Conductance φ is related to the second eigenvalue of the matrix M
- Low conductance means that there is some bottleneck and a subset of nodes not well connected with the rest of the graph
- High conductance means that the graph is well connected
- Communities are likely to have low conductance

Conductance

Graph Conductance:

$$\varphi(G) = \min_U \frac{E(U, V - U)}{\min\{d(U), d(V - U)\}}$$

$$COND(s) = \frac{2}{\min\{16, 24\}} = 0.125$$



Clustering

- Given a set of objects V , and a notion of **similarity** (or **distance**) between them, partition the objects into disjoint sets S_1, S_2, \dots, S_k , such that objects within each set are **similar**, while objects across different sets are **dissimilar**
- Graph clustering:
 - Input: a graph $G=(V,E)$
 - edge (u,v) denotes **similarity** between u and v
 - weighted graphs: weight of edge captures the degree of similarity
 - Clustering: Partition the nodes in the graph such that nodes within clusters are well interconnected (high edge weights), and nodes across clusters are sparsely interconnected (low edge weights)
 - The number of edges between clusters is called **cut-size**
 - most graph partitioning problems are NP hard

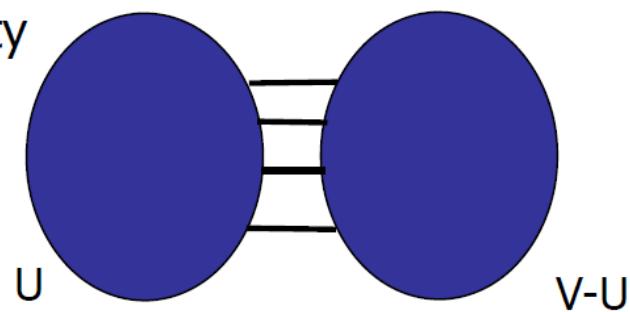
The methods

- What property or measure of network is used in this algorithm or method?
 - eigenvalue and eigenvector, spectrum of adjacency matrix (we briefly discussed this one)
 - Edge betweenness, information centrality
 - Distance, dissimilarity index, edge clustering coefficient, etc
- Agglomerative or divisive?
- What is the required prior information here?
 - Whether there is community or not
 - How many communities are there
- Performance of partitioning results and computational complexity

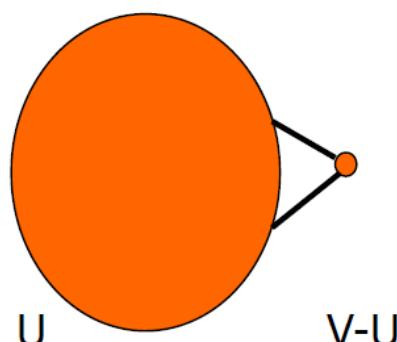
Measuring connectivity

- What does it mean that a set of nodes are well or sparsely interconnected?
- **min-cut**: the min number of edges such that when removed cause the graph to become disconnected

- small min-cut implies sparse connectivity
- $\min_U E(U, V - U) = \sum_{i \in U} \sum_{j \in V - U} A[i, j]$



- not always a good idea!



Graph Expansion

- Normalize the cut by the size of the smallest component
- Cut ratio:

$$\alpha = \frac{E(U, V - U)}{\min\{|U|, |V - U|\}}$$

- Graph expansion:

$$\alpha(G) = \min_U \frac{E(U, V - U)}{\min\{|U|, |V - U|\}}$$

Graph Expansion vs Conductance

- **Conductance** = edges leaking / how many edges are inside
- **Expansion** = edges leaking / how many nodes are inside
- **Conductance is more careful because it knows that some nodes are "heavy" (high degree) and others are "light."**
- The expansion does not capture the inter-cluster similarity well
 - **The nodes with high degree are more important**

07

Girvan-Newman algorithm

Girvan-Newman

- 1. Calculate betweenness for all edges in the network
- 2. Remove the edge with the highest betweenness
- 3. Recalculate betweenness for all edges affected by the removal
- 4. Repeat from step 2 until no edges remain
- 5. From the resulting dendrogram (the hierarchical mapping produced by gradually removing these edges), select the partition that maximizes network modularity

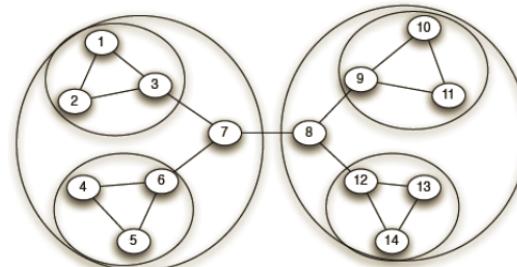
08

Louvain algorithm

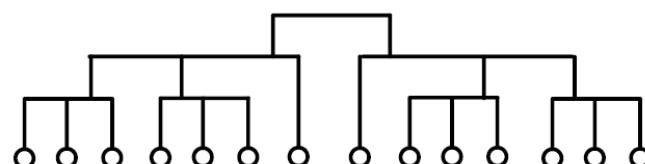
Louvain

- **Greedy algorithm** for community detection
 - $O(n \log n)$ run time
- Supports weighted graphs
- Provides hierarchical communities
- Widely utilized to **study large networks** because:
 - Fast
 - Rapid convergence
 - High modularity output
(i.e., “better communities”)

Network and communities:



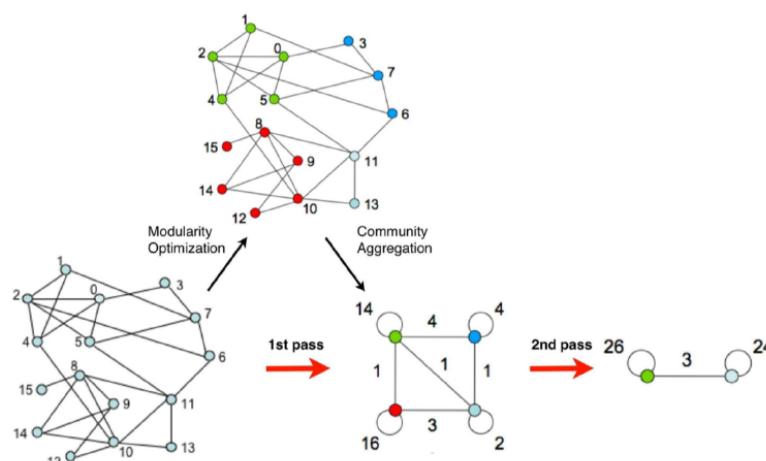
Dendrogram:



Louvain

- Louvain algorithm **greedily maximizes** modularity
- **Each pass is made of 2 phases:**
 - **Phase 1:** Modularity is **optimized** by allowing only local changes to node-communities memberships
 - **Phase 2:** The identified communities are **aggregated** into super-nodes to build a new network
 - **Goto Phase 1**

The passes are repeated **iteratively** until no increase of modularity is possible.



Louvain First Phase

- Put each node in a graph into a **distinct community** (one node per community)
- For each node i , the algorithm performs two calculations:
 - Compute the modularity delta (ΔQ) when putting node i into the community of some neighbor j
 - Move i to a community of node j that yields the largest gain in ΔQ
- **Phase 1 runs until no movement yields a gain**

This first phase stops when a local maxima of the modularity is attained, i.e., when no individual node move can improve the modularity.

Note that the output of the algorithm depends on the order in which the nodes are considered.

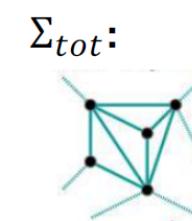
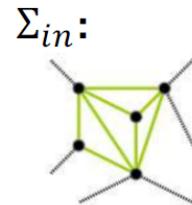
Research indicates that the ordering of the nodes does not have a significant influence on the overall modularity that is obtained.

Modularity Gain

What is ΔQ if we move node i to community C ?

$$\Delta Q(i \rightarrow C) = \left[\frac{\Sigma_{in} + k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2m} \right)^2 \right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right]$$

- where:
 - Σ_{in} ... sum of link weights between nodes in C
 - Σ_{tot} ... sum of all link weights of nodes in C
 - $k_{i,in}$... sum of link weights between node i and C
 - k_i ... sum of all link weights (i.e., degree) of node i
- Also need to derive $\Delta Q(D \rightarrow i)$ of taking node i out of community D .
- And then: $\Delta Q = \Delta Q(i \rightarrow C) + \Delta Q(D \rightarrow i)$



Modularity Gain

More in detail:

$$\Delta Q(i \rightarrow C) = \left[\frac{\Sigma_{in} + k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2m} \right)^2 \right] - \left[\underbrace{\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m} \right)^2}_{\text{Modularity of } C} - \underbrace{\left(\frac{k_i}{2m} \right)^2}_{\text{Modularity of } i} \right]$$

Self-edge weight Σ_{in}

Edge weight of the resulting super-node from merging C and i: $\Sigma_{tot} - \Sigma_{in} - (k_{i,in}/2)$ + $k_i - (k_{i,in}/2)$

rest of the graph (modeled as a single node)

Modularity contribution after merging node i

Modularity contribution before merging node i

By applying the Modularity definition:

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$$

Louvain Second Phase

- The communities obtained in the first phase are contracted into **super-nodes**, and the network is created accordingly:
 - Super-nodes are connected if there is at least one edge between the nodes of the corresponding communities
 - The weight of the edge between the two super-nodes is the sum of the weights from all edges between their corresponding communities
- **Phase 1 is then run on the super-node network**

Louvain Algorithm

Algorithm 1: Sequential Louvain Algorithm

Input: $G = (V, E)$: graph representation.
Output: C : community sets at each level;
 Q : modularity at each level.
Var: \hat{c} : vertex u 's best candidate community set.

```

1 Loop outer
2    $C \leftarrow \{\{u\}\}, \forall u \in V ;$ 
3    $\Sigma_{in}^c \leftarrow \sum w_{u,v}, e(u,v) \in E, u \in c \text{ and } v \in c ;$ 
4    $\Sigma_{tot}^c \leftarrow \sum w_{u,v}, e(u,v) \in E, u \in c \text{ or } v \in c ;$ 
5   // Phase 1.
6   Loop inner
7     for  $u \in V$  and  $u \in c$  do
8       // Find the best community for vertex  $u$ .
9        $\hat{c} \leftarrow \operatorname{argmax}_{\forall c', \exists e(u,v) \in E, v \in c'} \Delta Q_{u \rightarrow c'} ;$  Modularity gain
10      if  $\Delta Q_{u \rightarrow \hat{c}} > 0$  then
11        // Update  $\Sigma_{tot}$  and  $\Sigma_{in}$ .
12         $\Sigma_{tot}^{\hat{c}} \leftarrow \Sigma_{tot}^{\hat{c}} + w(u) ; \Sigma_{in}^{\hat{c}} \leftarrow \Sigma_{in}^{\hat{c}} + w_{u \rightarrow \hat{c}} ;$ 
13         $\Sigma_{tot}^c \leftarrow \Sigma_{tot}^c - w(u) ; \Sigma_{in}^c \leftarrow \Sigma_{in}^c - w_{u \rightarrow c} ;$ 
14        // Update the community information.
15         $\hat{c} \leftarrow \hat{c} \cup \{u\} ; c \leftarrow c - \{u\} ;$ 
16      if No vertex moves to a new community then
17        exit inner Loop;
  
```

Halting criterion for 1st Phase

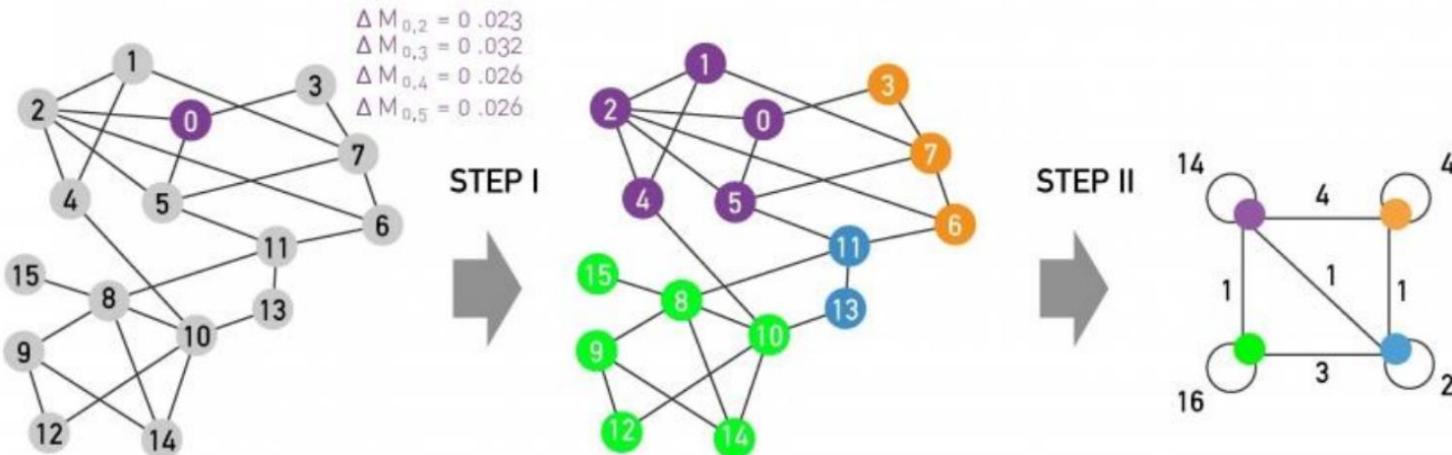
```

18   // Calculate community set and modularity.
19    $Q \leftarrow 0 ;$ 
20   for  $c \in C$  do
21      $Q \leftarrow Q + \frac{\Sigma_{in}^c}{2m} - \left(\frac{\Sigma_{tot}^c}{2m}\right)^2 ;$ 
22    $C' \leftarrow \{c\}. \forall c \in C : \text{print } C' \text{ and } Q ;$ 
23   // Phase 2: Rebuild Graph.
24    $V' \leftarrow C' ;$  Communities contracted into super-nodes
25    $E' \leftarrow \{e(c, c')\}, \exists e(u, v) \in E, u \in c, v \in c' ;$ 
26    $w_{c,c'} \leftarrow \sum w_{u,v}, \forall e(u, v) \in E, u \in c, v \in c' ;$ 
27   if No community changes then
28     | exit outer Loop;
29    $V \leftarrow V' ; E \leftarrow E' ;$  Halting criterion
    for 2nd Phase
  
```

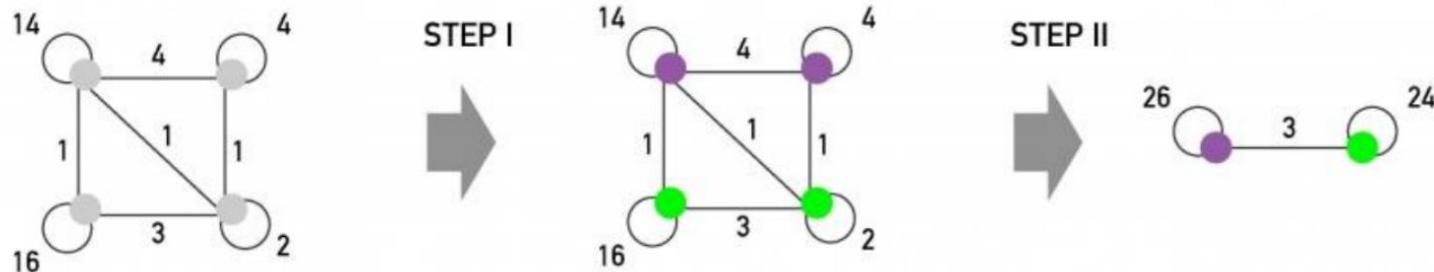
the weights of the edges between the new super-nodes are given by the **sum of the weights of the edges** between vertices in the corresponding two communities

Louvain Algorithm

1ST PASS



2ND PASS



Limitations

- Resolution limit: hard to detect smaller communities, may be merged into larger communities
 - Intermediate steps may have more intuitive structures
- Degeneracy problem: exponentially large number of possible community assignments with close to maximum modularity
 - Hard to find global maximum (NP-hard)
 - Is the global maximum better, more scientifically important than other community assignments with similar modularity?
- Locally optimal community assignments can have different structural properties

09

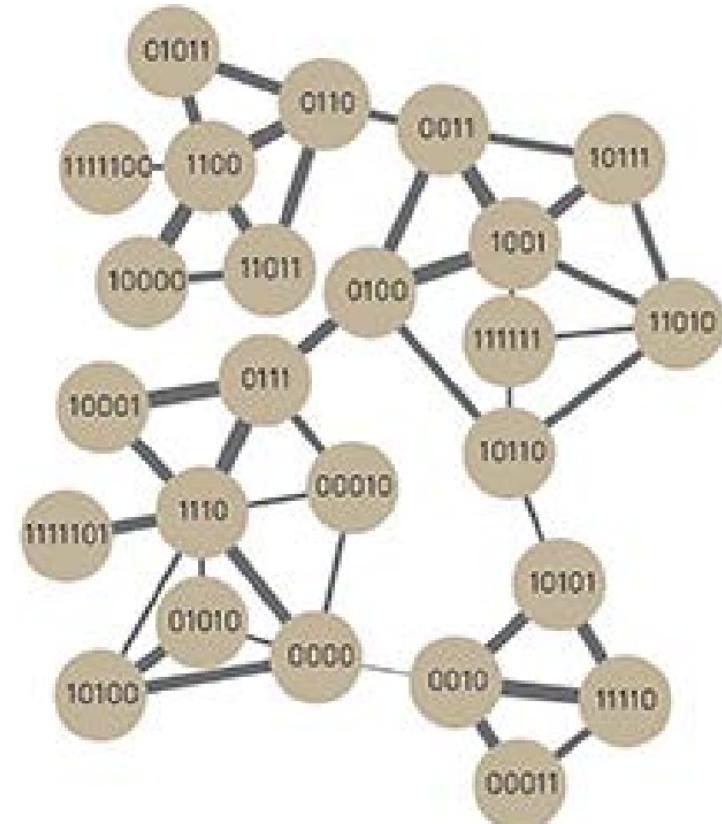
Infomap algorithm

Goal

- If we want to understand how network structure relates to system behavior,
- we need to understand the flow of information on the network
- A group of nodes among which information flows quickly and easily can be aggregated as a single well-connected module/community
- Succinctly describing information flow is a coding/compression problem
- Use a random walk as a proxy for information flow
 - Random walk uses all of the information in the network representation and nothing
 - more
- Finding community structure in networks is equivalent to solving a coding problem
 - Want a compressed description of a random walk, with unique names for important structures

Attempt 1

- Give a unique name to each node
- Derived using a Huffman encoding, using the estimated probability that the random walk visits that node
 - Huffman encoding assigns shorter names to more common nodes
- The random walk can be represented in 314 bits

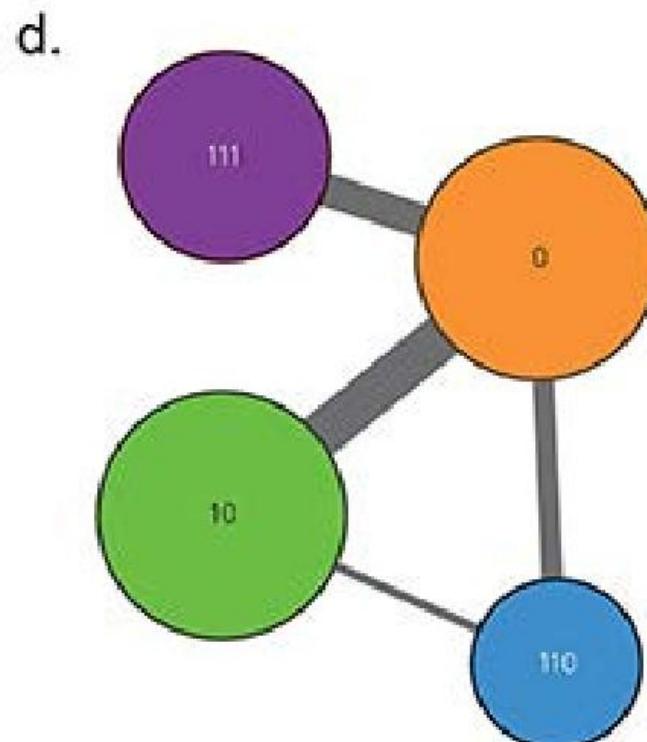
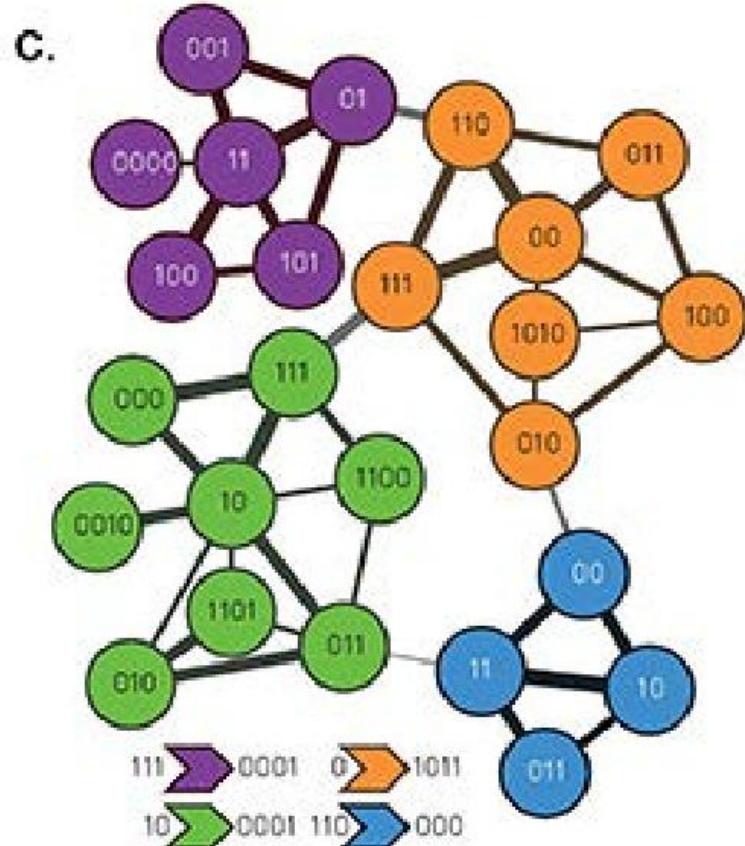


```
1111100 1100 0110 11011 10000 11011 0110 0011 10111 1001 0011  
1001 0100 0111 10001 1110 0111 10001 0111 1110 0000 1110 10001  
0111 1110 0111 1110 111101 1110 0000 10100 0000 1110 10001 0111  
0100 10110 11010 10111 1001 0100 1001 10111 1001 0100 1001 0100  
0011 0100 0011 0110 11011 0110 0011 0100 1001 10111 0011 0100  
0111 10001 1110 10001 0111 0100 10110 111111 10110 10101 11110  
00011
```

Attempt 2

- Use a two levels description, retaining unique names for each module, but use a different Huffman code for nodes within each cluster
- Start with name of the cluster, then the node names within the cluster
- Exit code is chosen as part of the within-cluster Huffman coding,
- indicating the walk is leaving the current cluster
 - Exit code is followed by the name of the next cluster

Illustration



Map Equation

Look for a module partition M of n nodes into m modules to minimize the expected description length of a random walk

Average description length is given by the map equation or L

$$L(M) = q_{\sim}H(Q) + \sum_{i=1}^m p_{\text{v}}^i H(P^i)$$

- First term: Average number of bits necessary to describe movement between communities
- Second term: Average number of bits necessary to describe movement within communities
 - Exiting is also considered a movement

Map Equation

$$L(\mathbf{M}) = q_{\sim} H(\mathcal{Q}) + \sum_{i=1}^m p_{\circlearrowleft}^i H(\mathcal{P}^i)$$

$$q_{\sim} = \sum_{i=1}^m q_{i\sim}$$

- Per-step probability that the random walker switches modules
- $q_{i\sim}$ is the per-step probability that the walker leaves module i

$$p_{\circlearrowleft}^i = q_{i\sim} + \sum_{\alpha \in i} p_{\alpha}$$

- Weights the entropy of movements within module i
- p_{α} is the ergodic node visit frequency at node α within the random walk

$$H(\mathcal{Q}) = \sum_{i=1}^m \frac{q_{i\sim}}{\sum_{j=1}^m q_{j\sim}} \log \left(\frac{q_{i\sim}}{\sum_{j=1}^m q_{j\sim}} \right)$$

- $H(\mathcal{Q})$ is the entropy of movements between modules

$$H(\mathcal{P}^i) = \frac{q_{i\sim}}{q_{i\sim} + \sum_{\beta \in i} p_{\beta}} \log \left(\frac{q_{i\sim}}{q_{i\sim} + \sum_{\beta \in i} p_{\beta}} \right) + \sum_{\alpha \in i} \frac{p_{\alpha}}{q_{i\sim} + \sum_{\beta \in i} p_{\beta}} \log \left(\frac{p_{\alpha}}{q_{i\sim} + \sum_{\beta \in i} p_{\beta}} \right)$$

- $H(\mathcal{P}^i)$ is the entropy of movements within module i

Minimizing the Map Equation

- Use the same optimization procedure as in the Louvain algorithm
 - Start with each node in a different community
 - Move nodes to neighboring communities for greatest decrease in L
 - Aggregate communities
 - Repeat until L is minimized

Complexity

- Computational complexity is determined by the procedure used to minimize the map equation
- So, it becomes the same as the Louvain algorithm

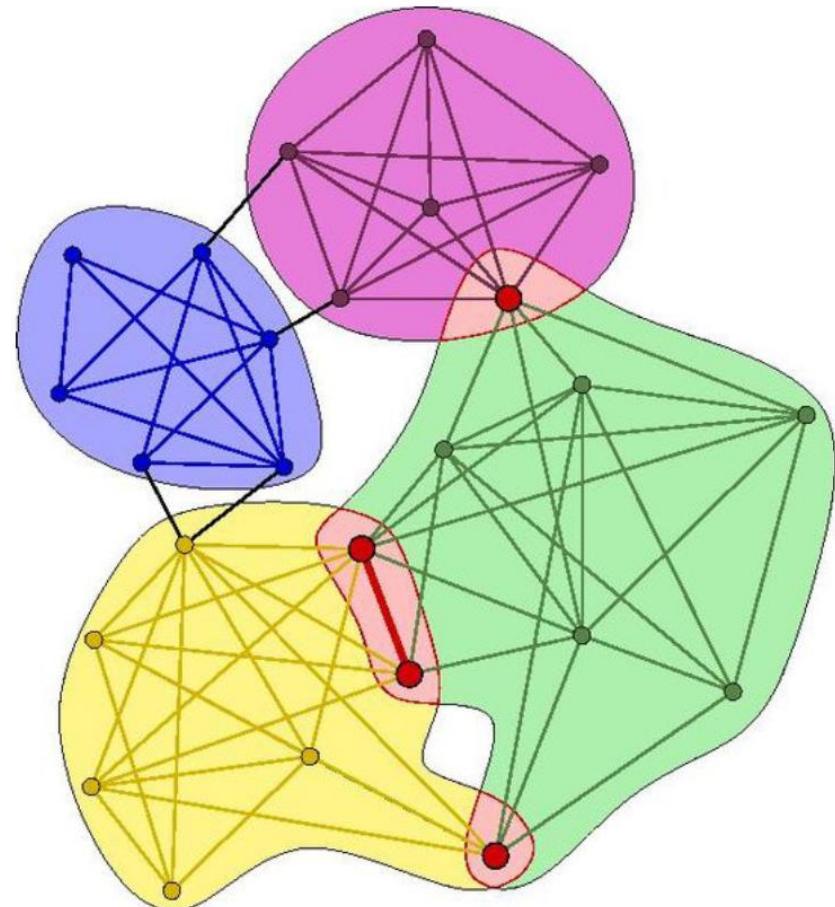
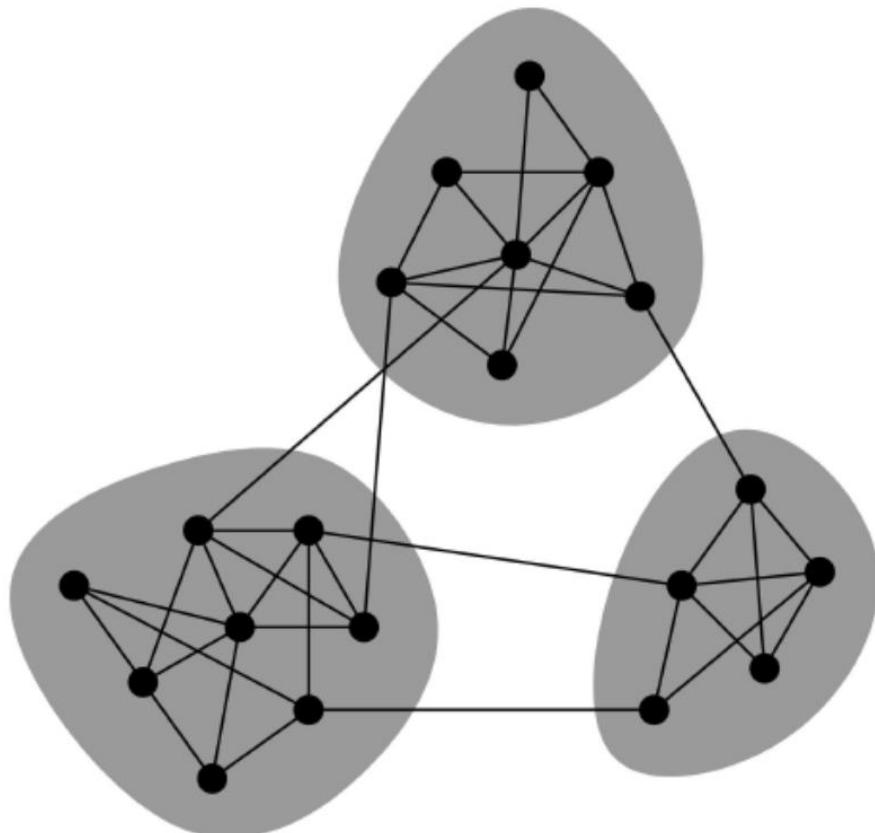
$O(L \log L)$ or $O(N \log N)$ for a sparse network

05

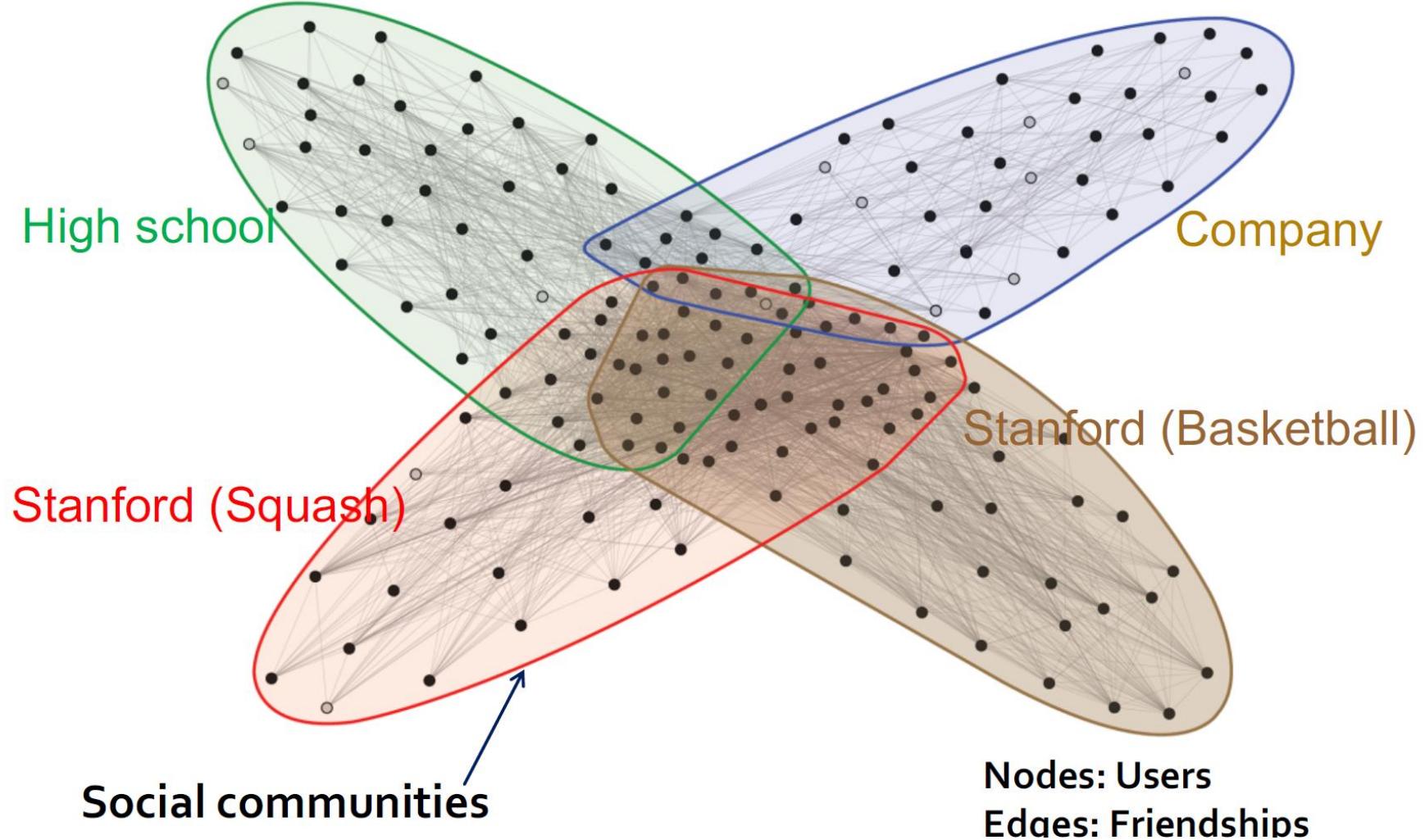
Overlapping Communities

Complexity

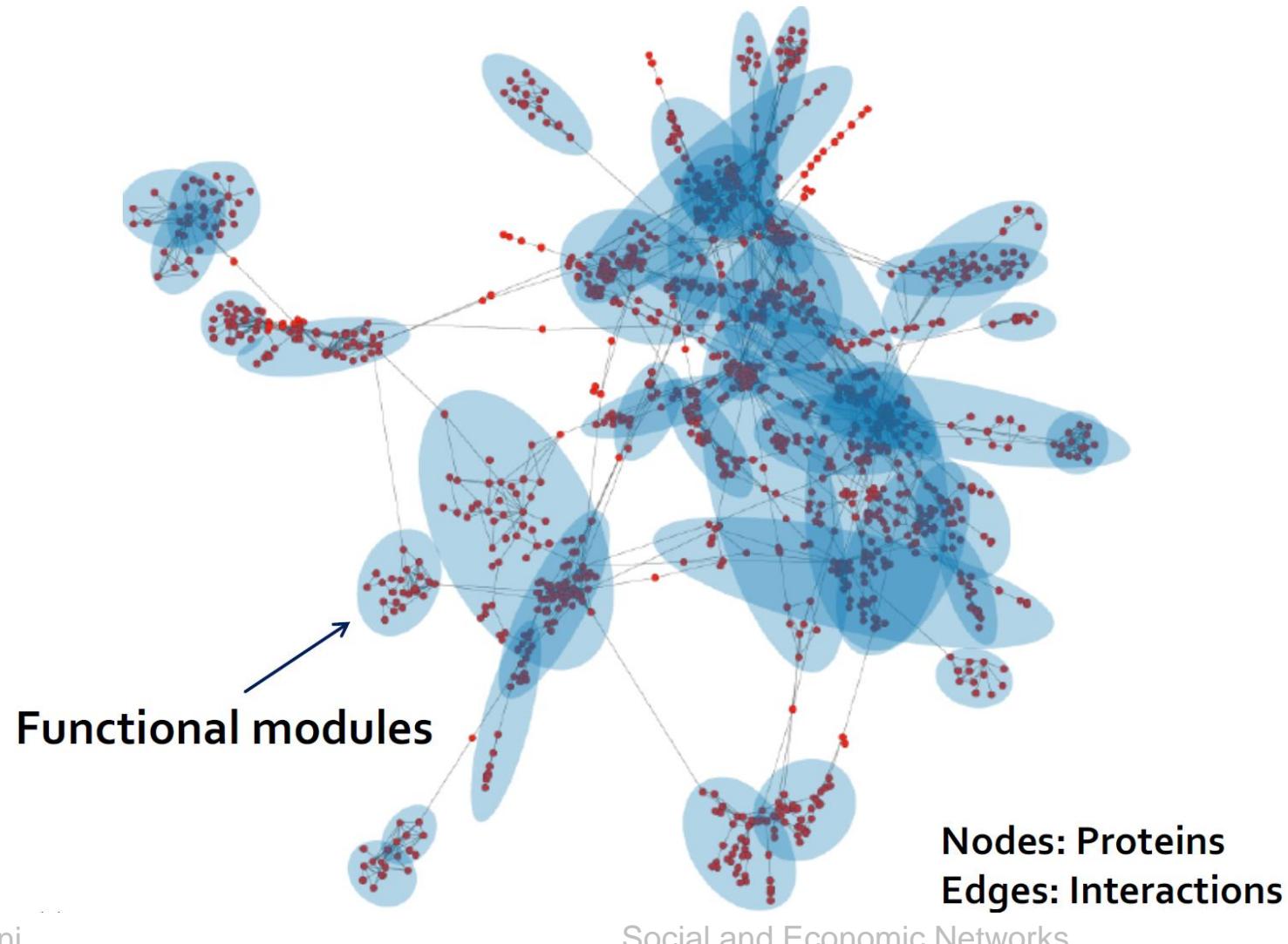
- Non-overlapping vs. overlapping communities



Facebook Network

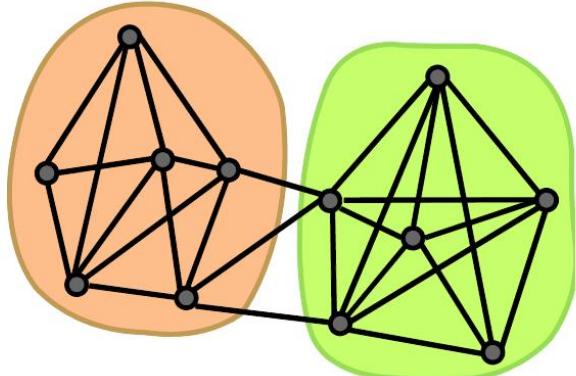


Protein-Protein Interaction



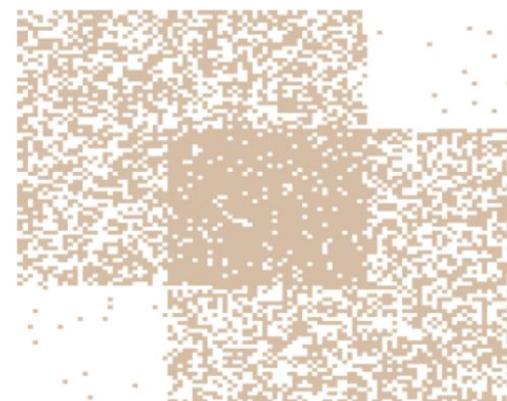
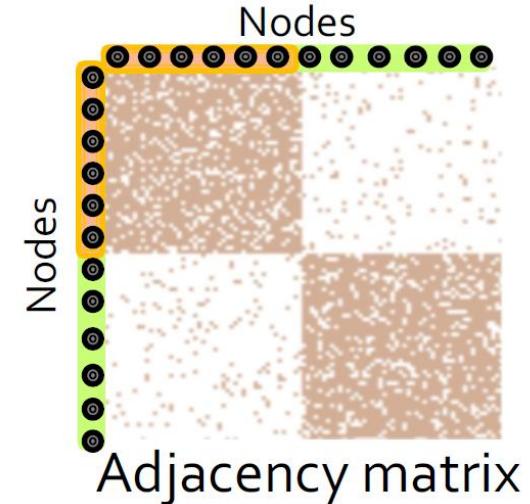
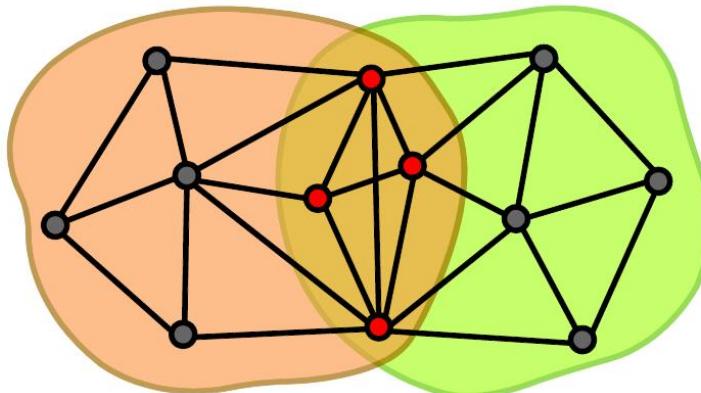
Communities

Non-overlapping



Network

Overlapping



06

BigCLAM

Steps

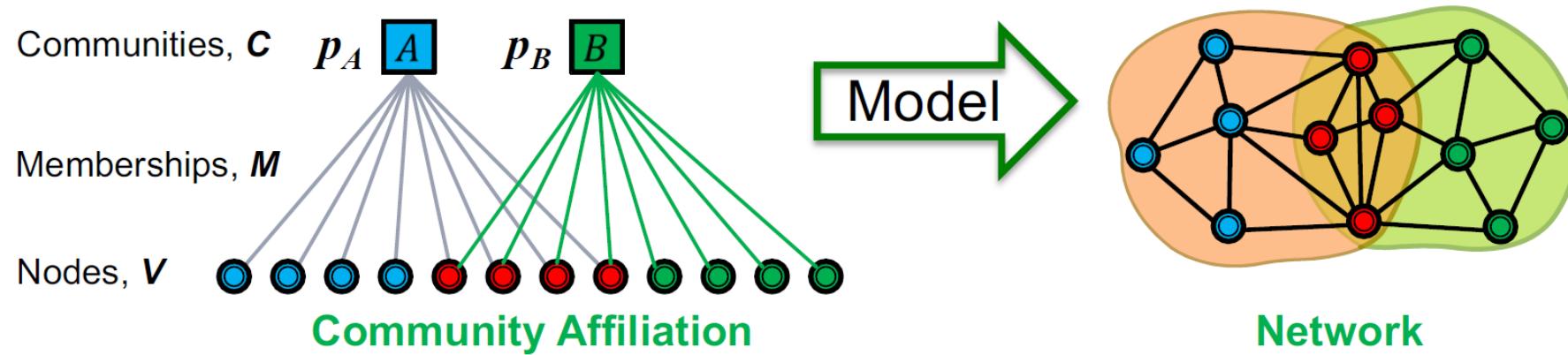
Step 1)

- Define a generative model for graphs that is based on node community affiliations
 - Community Affiliation Graph Model (AGM)

Step 2)

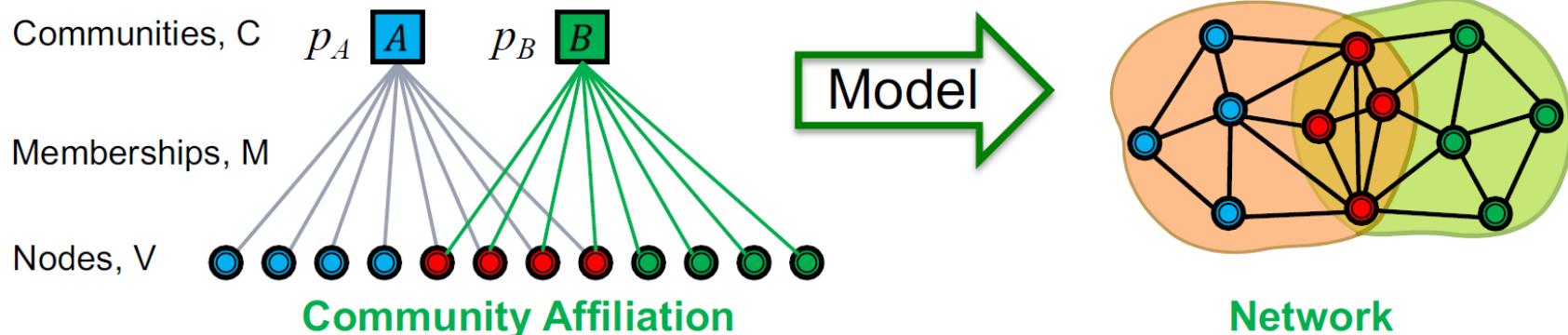
- Given graph G , make the assumption that G was generated by AGM
- Find the best AGM that could have generated G
- **And this way we discover communities**

Community Affiliation Graph Model (AGM)



- **Generative model:** How is a network generated from community affiliations?
- **Model parameters:**
 - Nodes V , Communities C , Memberships M
 - Each community c has a single probability p_c

Generative Process

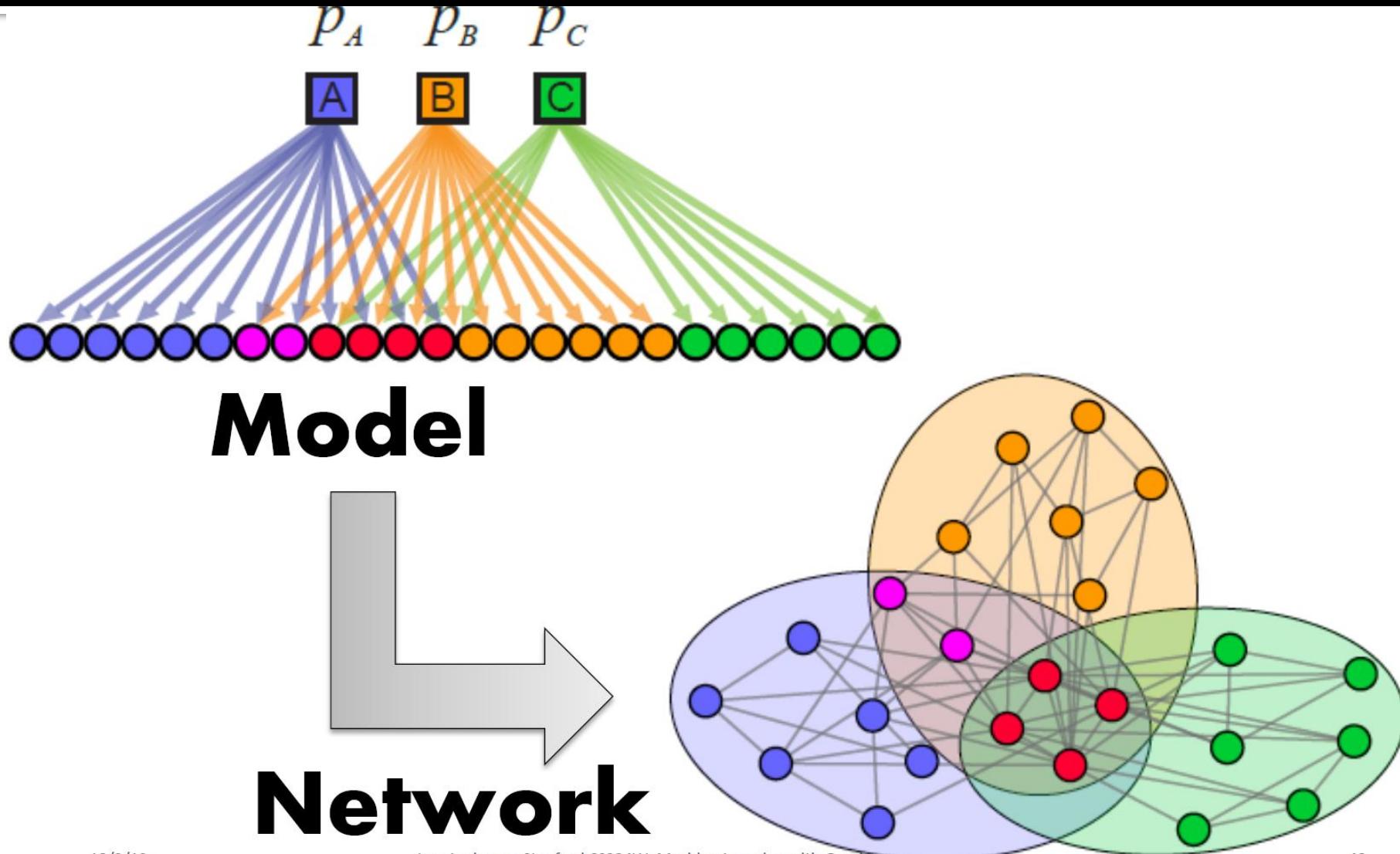


- Given parameters ($V, C, M, \{p_c\}$)
 - Nodes in community c connect to each other by flipping a coin with probability p_c
 - **Nodes that belong to multiple communities have multiple coin flips**
 - If they “miss” the first time, they get another chance through the next community

$$p(u, v) = 1 - \prod_{c \in M_u \cap M_v} (1 - p_c)$$

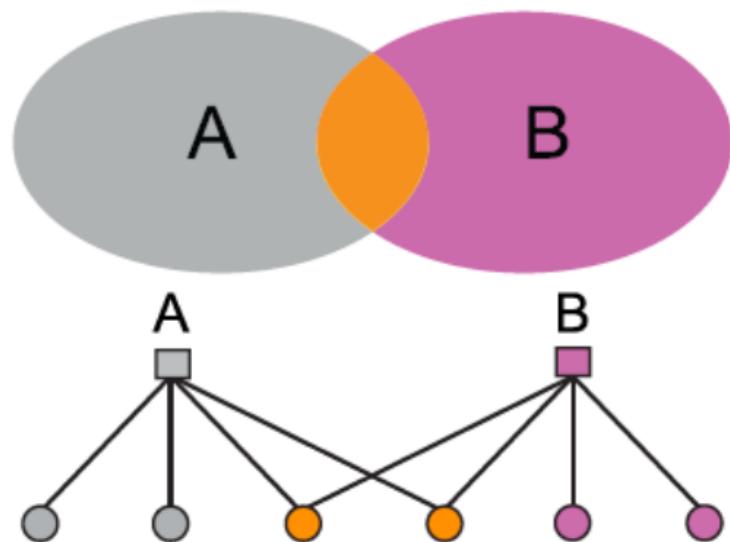
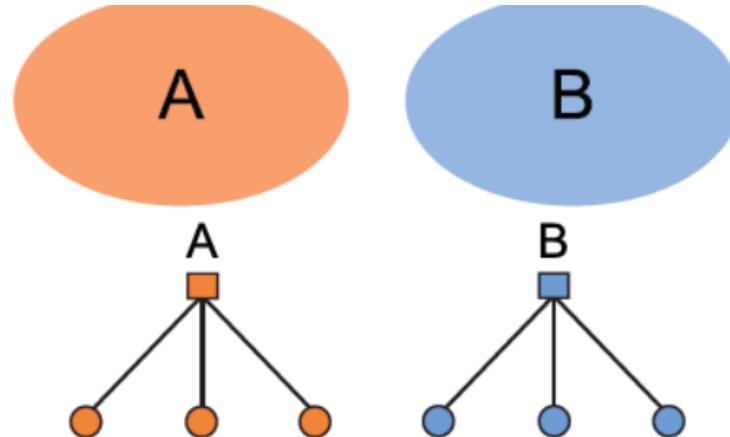
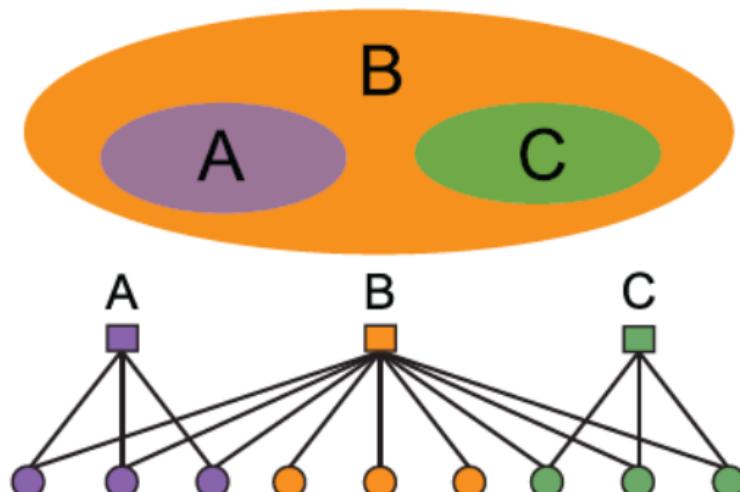
Note: If nodes u and v have no communities in common, then $p(u,v)=0$. We resolve this by having a background “epsilon” community that every node is a member of.

Dense Overlaps



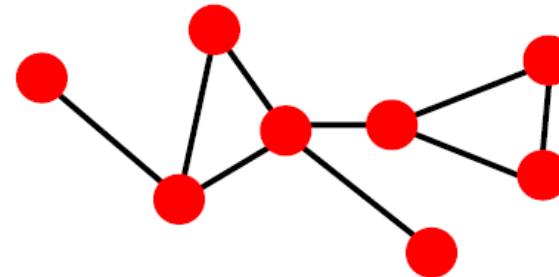
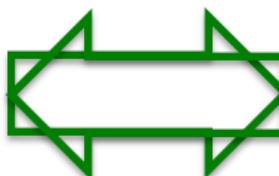
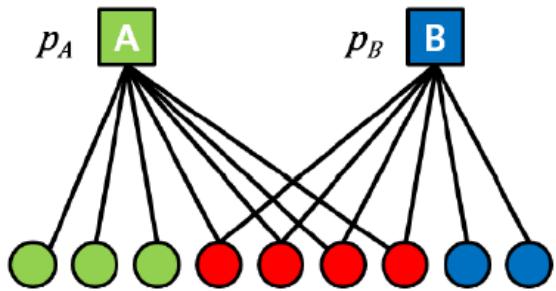
Flexibility

- AGM can express a variety of community structures:
Non-overlapping,
Overlapping, Nested



Algorithm

- Detecting communities with AGM:



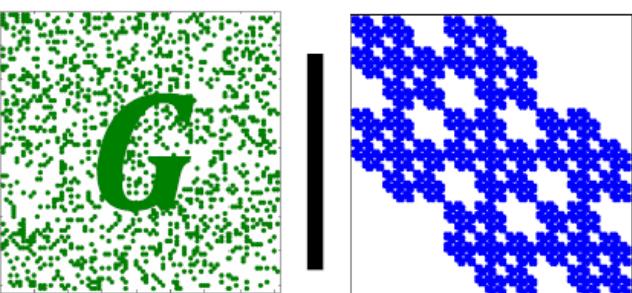
Given a Graph, find the model F

- 1) Affiliation graph M
- 2) Number of communities C
- 3) Parameters p_c

Graph Fitting

How to estimate model parameters F given a G ?

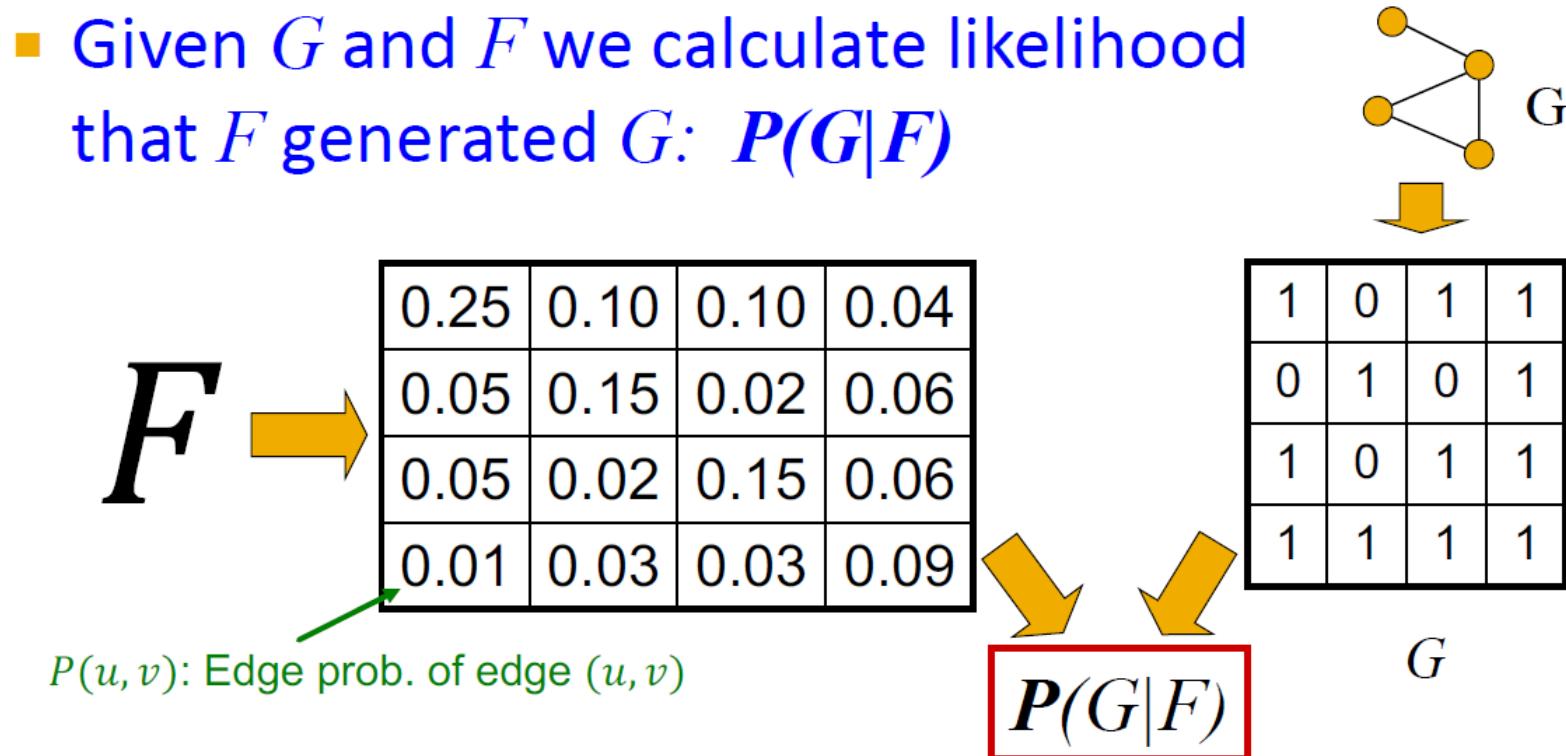
- Maximum likelihood estimation
- Given real graph G
- Find model/parameters F which

$$\arg \max_F P(G | \text{Model } F)$$


- To solve this we need to:
 - Efficiently calculate $P(G|F)$
 - Then maximize over F (e.g., using gradient descent)

Graph Likelihood

- Given G and F we calculate likelihood that F generated G : $P(G|F)$



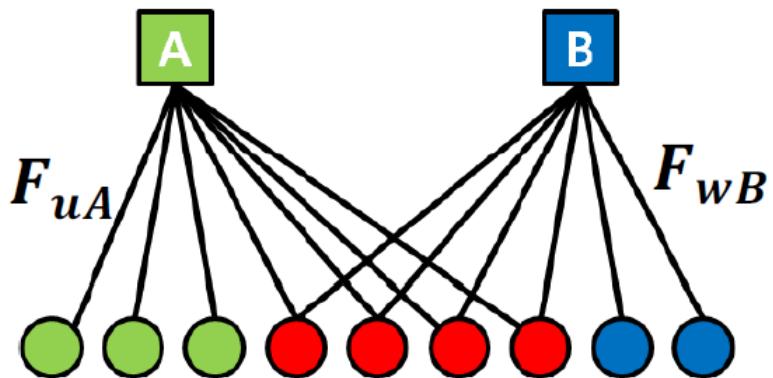
$$P(G|F) = \prod_{(u,v) \in G} P(u, v) \prod_{(u,v) \notin G} (1 - P(u, v))$$

Likelihood of edges in the graph

Likelihood of edges not in the graph

AGM Relaxing

- “Relax” the AGM: Memberships have strengths



- F_{uA} : The membership strength of node u to community A ($F_{uA} = \mathbf{0}$: no membership)
- F_u : A row vector of community memberships of node u

BigCLAM Model

- Prob. of nodes u, v linking is proportional to the strength of shared memberships:

$$P(u, v) = 1 - \exp(-\mathbf{F}_u \cdot \mathbf{F}_v^T)$$

- Given a network $G(V, E)$, we maximize $l(F)$

$$l(F) = \sum_{(u, v) \in E} \log(1 - \underbrace{\exp(-\mathbf{F}_u \mathbf{F}_v^T)}_{\text{Dot product}})) - \sum_{(u, v) \notin E} \mathbf{F}_u \mathbf{F}_v^T$$

This is log-likelihood of network G – total probability of all edges occurring and all non-edges not occurring.

- Optimization:

- Start with random F
- Update \mathbf{F}_{uC} for node u while fixing the memberships of all other nodes
- Updating takes linear time in the degree of u

BigCLAM Model

- **Gradient ascent:**

$$\nabla l(F_u) = \sum_{v \in \mathcal{N}(u)} F_v \frac{\exp(-F_u F_v^T)}{1 - \exp(-F_u F_v^T)} - \sum_{v \notin \mathcal{N}(u)} F_v$$

- Perform gradient ascent, where we make small changes to F that lead to increase in log-likelihood
- Pure gradient ascent is slow! **However:**

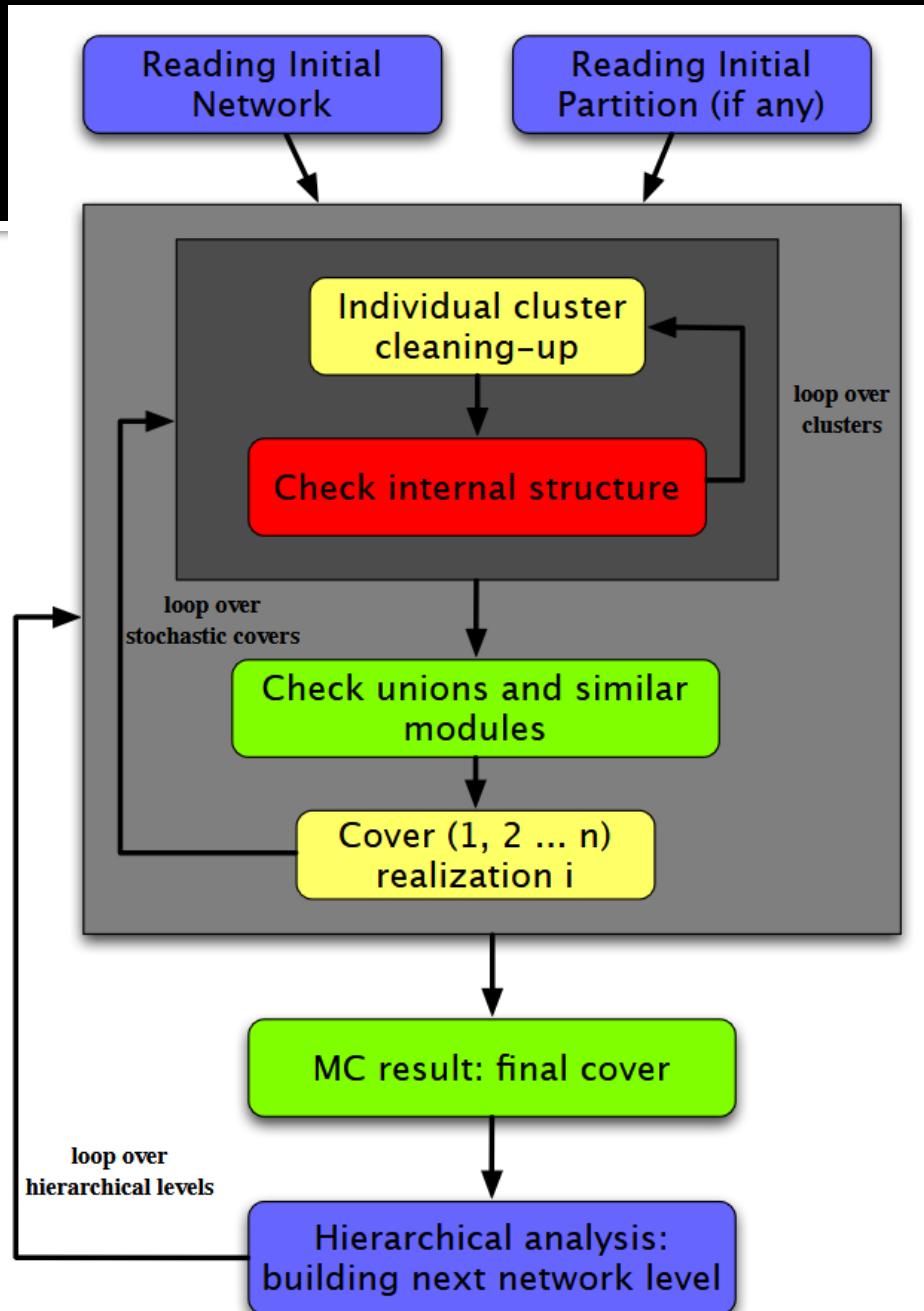
$$\sum_{v \notin \mathcal{N}(u)} F_v = (\sum_v F_v - F_u - \sum_{v \in \mathcal{N}(u)} F_v)$$

- By caching F_v , the gradient step takes **linear time** in the degree of u

07

OSLOM algorithm

OSLOM



<https://arxiv.org/pdf/1012.2363>



Any Question?