Community Detection

Prof. O-Joun Lee

Dept. of Artificial Intelligence,
The Catholic University of Korea
ojlee@catholic.ac.kr

네트워크 과학 연구실
NETWORK SCIENCE LAB

가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA

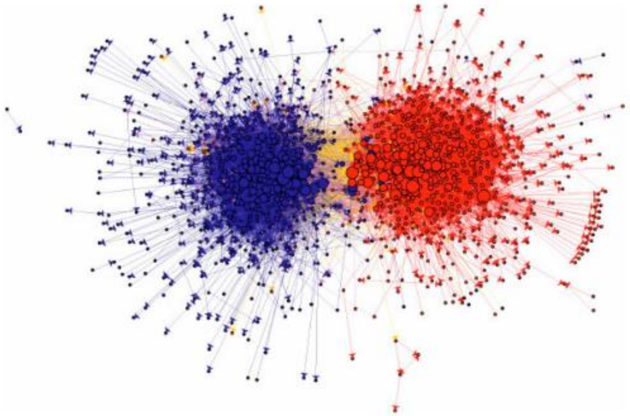
Contents

Learning Outcomes

- Understand why and how community detection and validation work:
  - Explain the connection to modularity.
- Distinguish methodologies used for overlapping and non-overlapping community detection.
- Contrast methodology used in networks.

Why community detection?

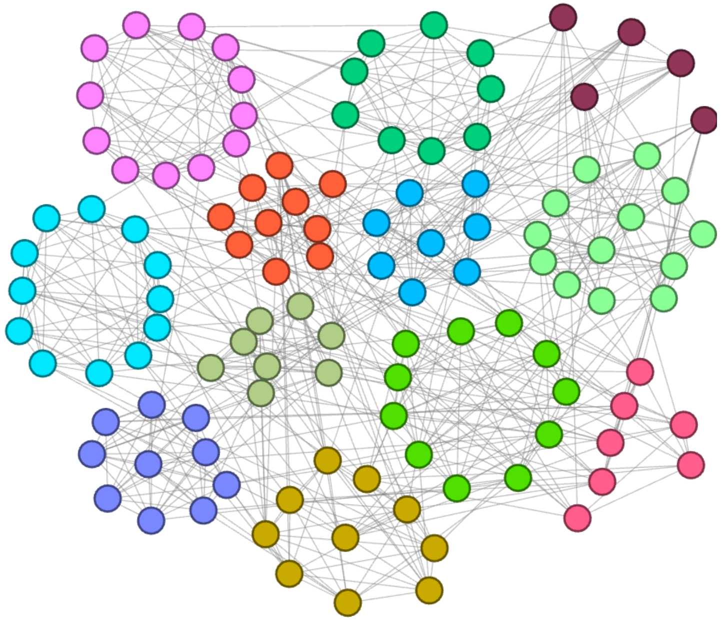
- Communities are features that appear in real networks
  - We generally try to identify them through the structural properties of the network: nodes tend to cluster based on common interests.
- Based on its usefulness, community detection became one of the most prominent directions of research in network science.
- It is one of the common analysis tools in understanding networks
- A community: a group of people with common characteristic or shared interests.



Adamic & Glance, 2005

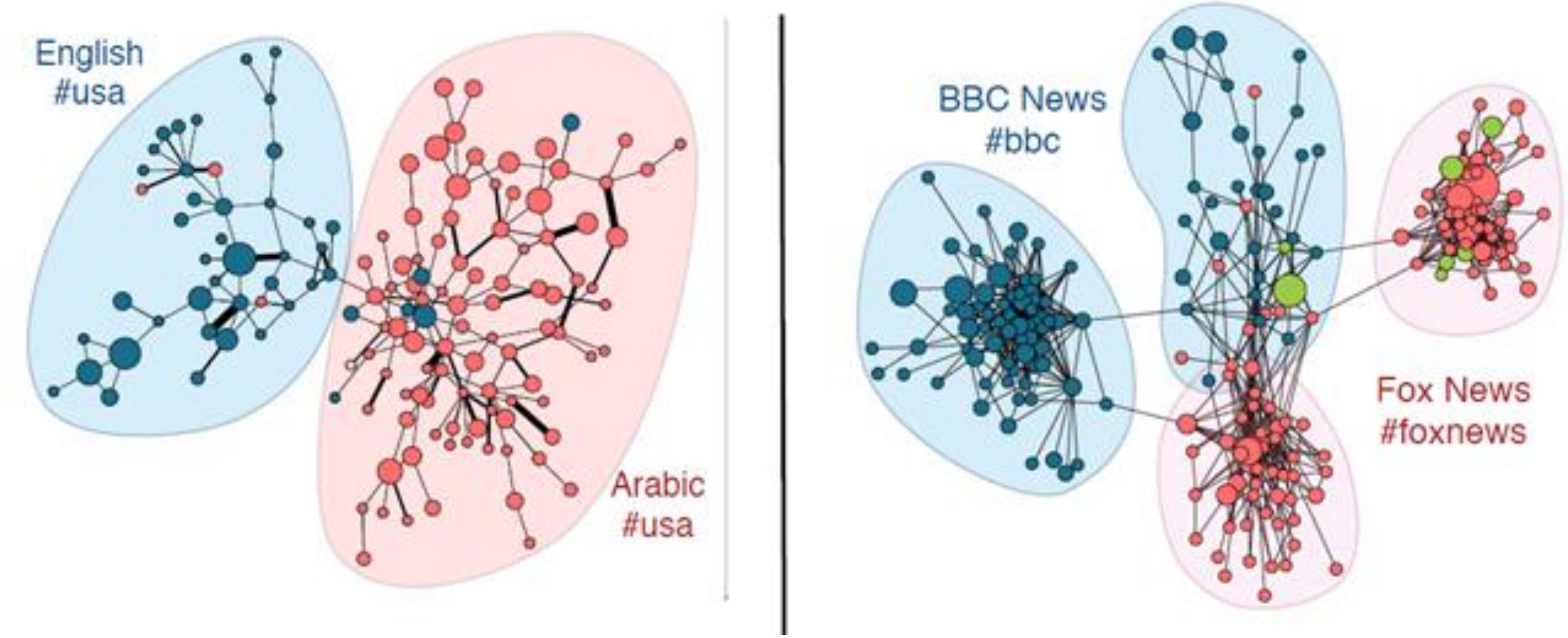
What is community?

- A community is a collection of nodes that have strong inter-connection within the community than what we would expect to occur at random.
- Given a graph G = (V, E), community detection consist in partitioning the set of vertices V into k subsets:
  - P = {C_1, C_2, …, C_k} such that:
  - {C_1 ∪ C_2 ∪…∪ C_k} = V and {C_1 ∩ C_2 ∩… ∩ C_k} = Ø.



What might influence a community?

- Homophily: similar nodes cluster together: for example based on Language (or based on degree for degree homophily)



Retweet network

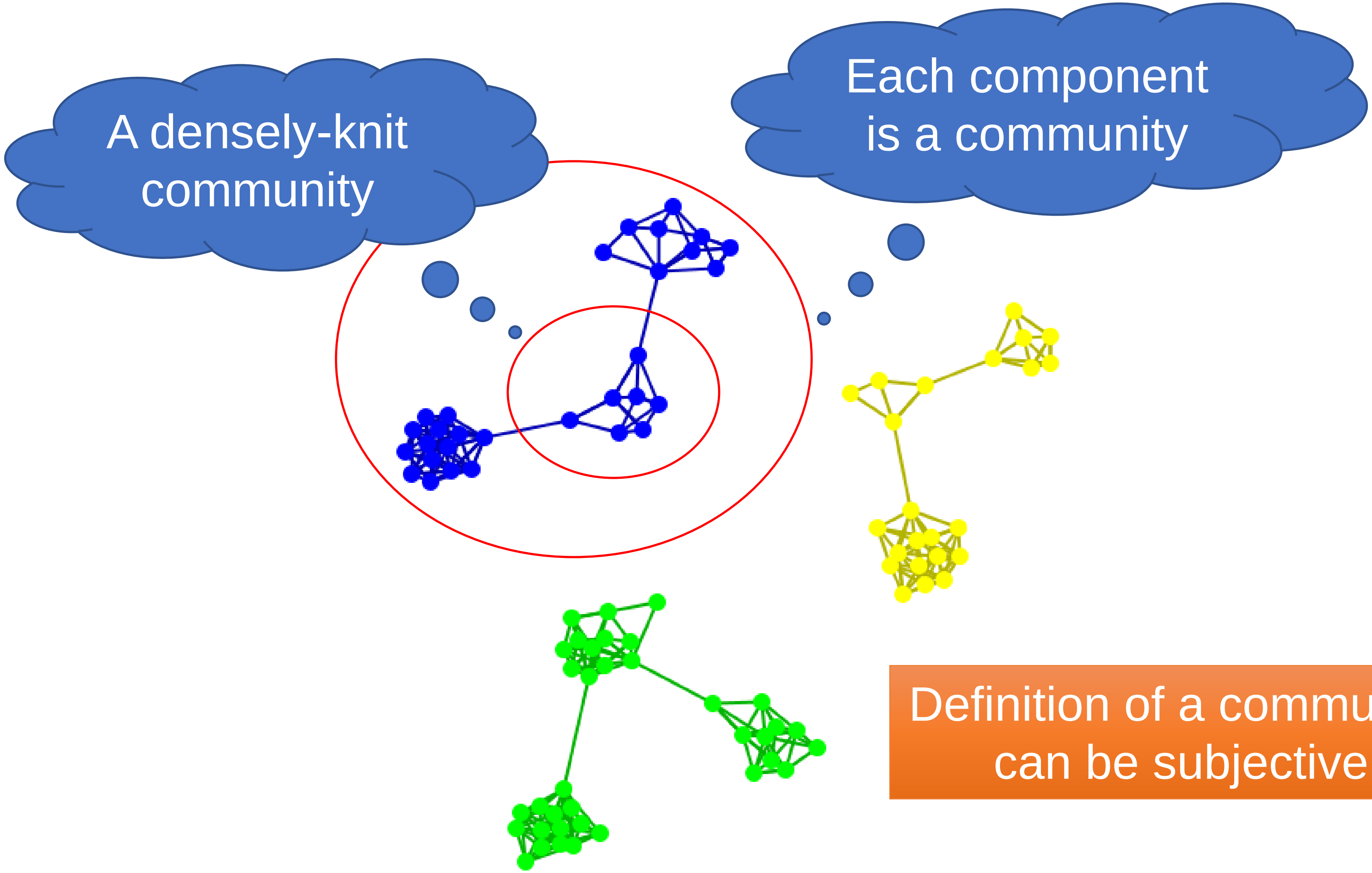
Follower network

Communities in Social Media: an example

- Two types of groups in social media
  - Explicit Groups: formed by user subscriptions
  - Implicit Groups: implicitly formed by social interactions

- Some social media sites allow people to join groups, is it necessary to extract groups based on network topology?
  - Not all sites provide community platform
  - Not all people want to make effort to join groups
  - Groups can change dynamically
- Network interaction provides rich information about the relationship between users
  - Can complement other kinds of information
  - Help network visualization and navigation
  - Provide basic information for other tasks

Subjectivity of Community Definition



A densely-knit community

Each component is a community

Definition of a community can be subjective.

Taxonomy of Community Criteria

- Criteria vary depending on the tasks.
- Roughly, community detection methods can be divided into 4 categories (not exclusive):

- 1. Node-Centric Community
  - Each node in a group satisfies certain properties
- 2. Group-Centric Community
  - Consider the connections within a group as a whole. The group has to satisfy certain properties without zooming into node-level
- 3. Network-Centric Community
  - Partition the whole network into several disjoint sets
- 4. Hierarchy-Centric Community
  - Construct a hierarchical structure of communities

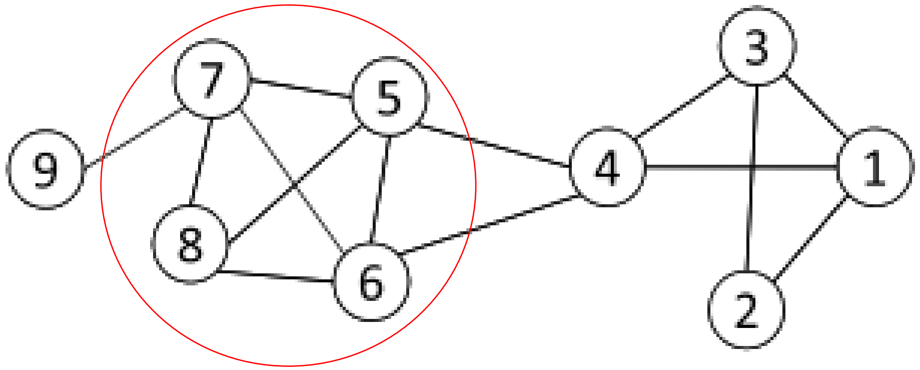
Node-Centric Community Detection

- Nodes satisfy different properties
    - Complete Mutuality
        - cliques
    - Reachability of members
        - k-clique, k-clan, k-club
    - Nodal degrees
        - k-plex, k-core
    - Relative frequency of Within-Outside Ties
        - LS sets, Lambda sets
- Commonly used in traditional social network analysis

We discuss some representative ones

Complete Mutuality: Cliques

- Clique: a maximum complete subgraph in which all nodes are adjacent to each other



Nodes 5, 6, 7 and 8 form a clique

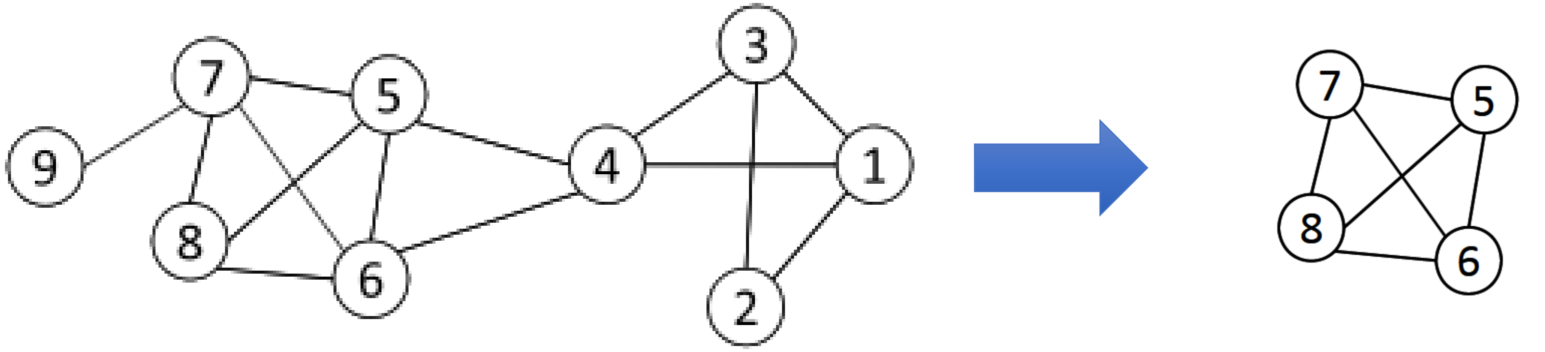
- NP-hard to find the maximum clique in a network
- Straightforward implementation to find cliques is very expensive in time complexity

Finding the Maximum Clique

- In a clique of size k, each node maintains degree >= k-1
- Nodes with degree < k-1 will not be included in the maximum clique
- Recursively apply the following pruning procedure:
  - Sample a sub-network from the given network, and find a clique in the sub-network, say, by a greedy approach
  - Suppose the clique above is size k, in order to find out a larger clique, all nodes with degree <= k-1 should be removed.
- Repeat until the network is small enough
- Many nodes will be pruned as social media networks follow a power law distribution for node degrees

Maximum Clique Example

- Suppose we sample a sub-network with nodes {1-5} and find a clique {1, 2, 3} of size 3
- In order to find a clique >3, remove all nodes with degree <= 3 – 1 = 2
    - Remove nodes 2 and 9
    - Remove nodes 1 and 3
    - Remove node 4

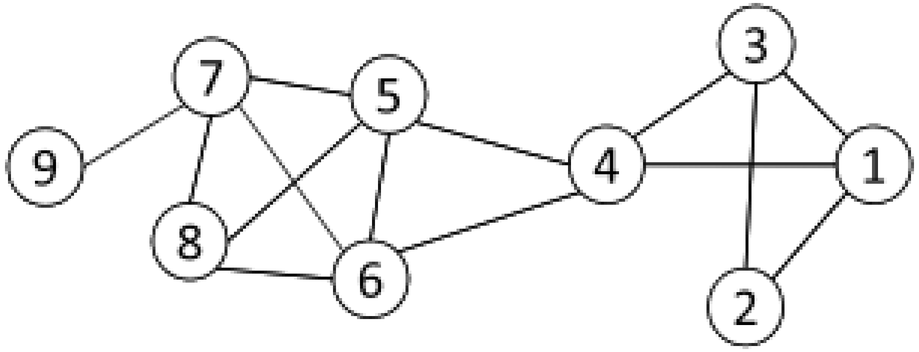


Clique Percolation Method (CPM)

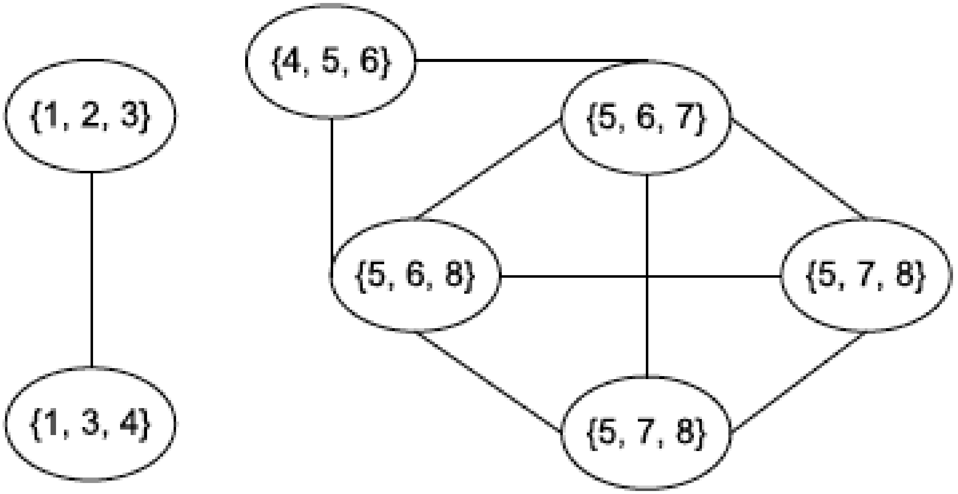
- Clique is a very strict definition, unstable
- Normally use cliques as a core or a seed to find larger communities

- CPM is such a method to find overlapping communities
- Input
  - A parameter k, and a network
- Procedure
  - Find out all cliques of size k in a 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 (CPM): Example



Cliques of size 3:
{1, 2, 3}, {1, 3, 4}, {4, 5, 6}, {5, 6, 7}, {5, 6, 8}, {5, 7, 8}, {6, 7, 8}

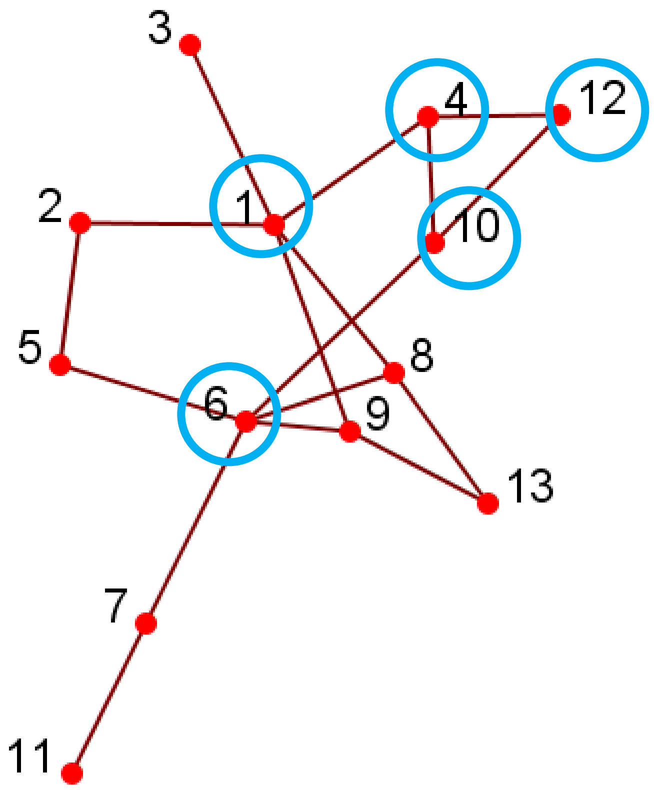


Communities:
{1, 2, 3, 4}
{4, 5, 6, 7, 8}

Reachability : k-clique, k-club

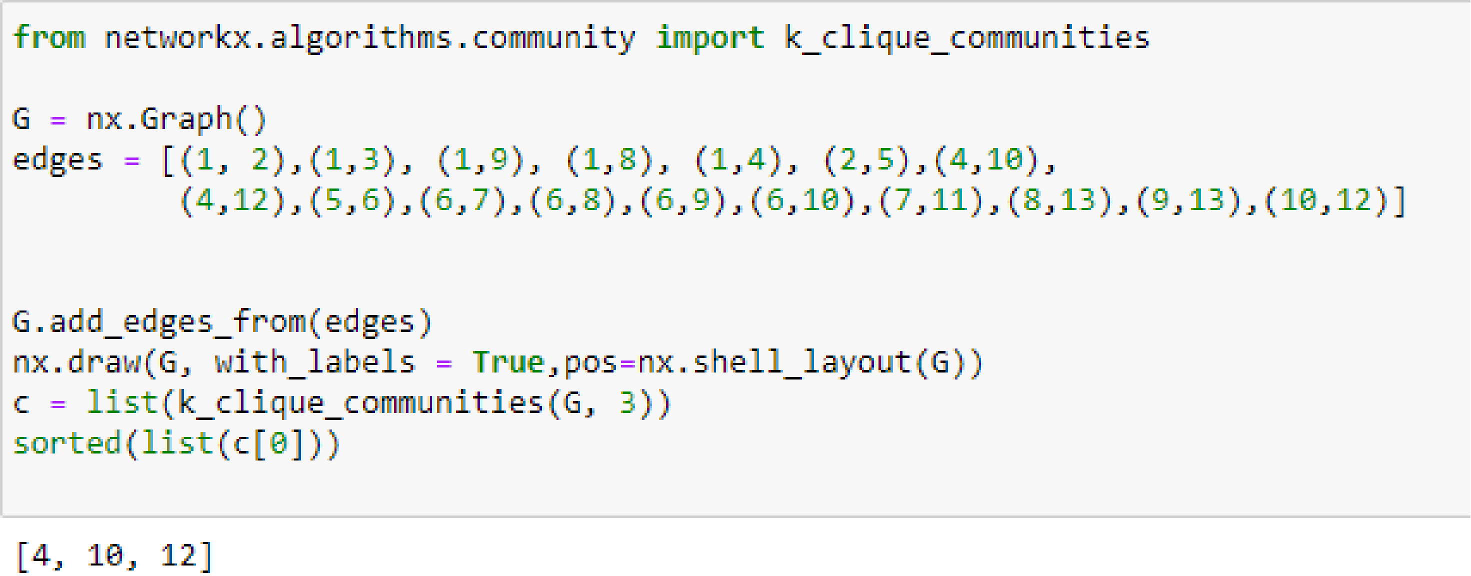
- Any node in a group should be reachable in k hops
- k-clique: a maximal subgraph in which the largest geodesic distance between any nodes <= k
- A k-clique can have diameter larger than k within the subgraph
    - e.g., 2-clique: {12, 4, 10, 1, 6}. In this clique, the distance between any two nodes <= 2-hop.

- k-club: a substructure of diameter <= k.
    - It means that every node pair is connected by at least one path with at most k edges.
    - e.g., {1,2,5,6,8,9}, {12, 4, 10, 1} are 2-clubs.



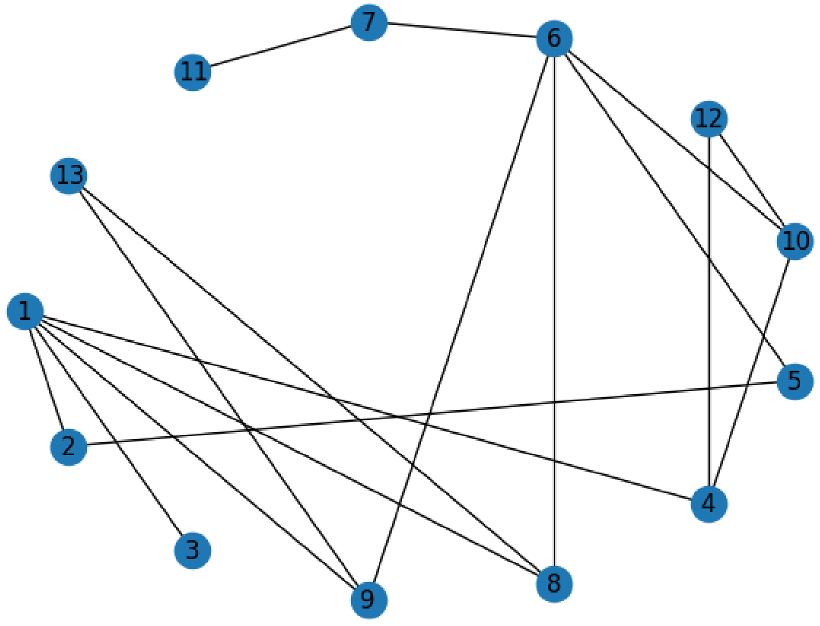
Find k-clique communities: sample code

- Find k-clique communities in graph using the percolation method

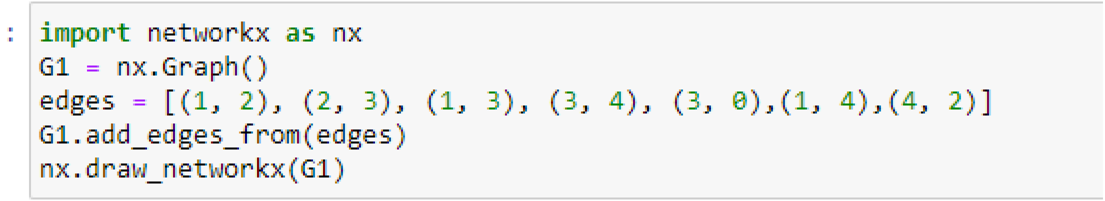


```
from networkx.algorithms.community import k_clique_communities

G = nx.Graph()
edges = [(1, 2),(1,3), (1,9), (1,8), (1,4), (2,5),(4,10),
         (4,12),(5,6),(6,7),(6,8),(6,9),(6,10),(7,11),(8,13),(9,13),(10,12)]


G.add_edges_from(edges)
nx.draw(G, with_labels = True,pos=nx.shell_layout(G))
c = list(k_clique_communities(G, 3))
sorted(list(c[0]))
```

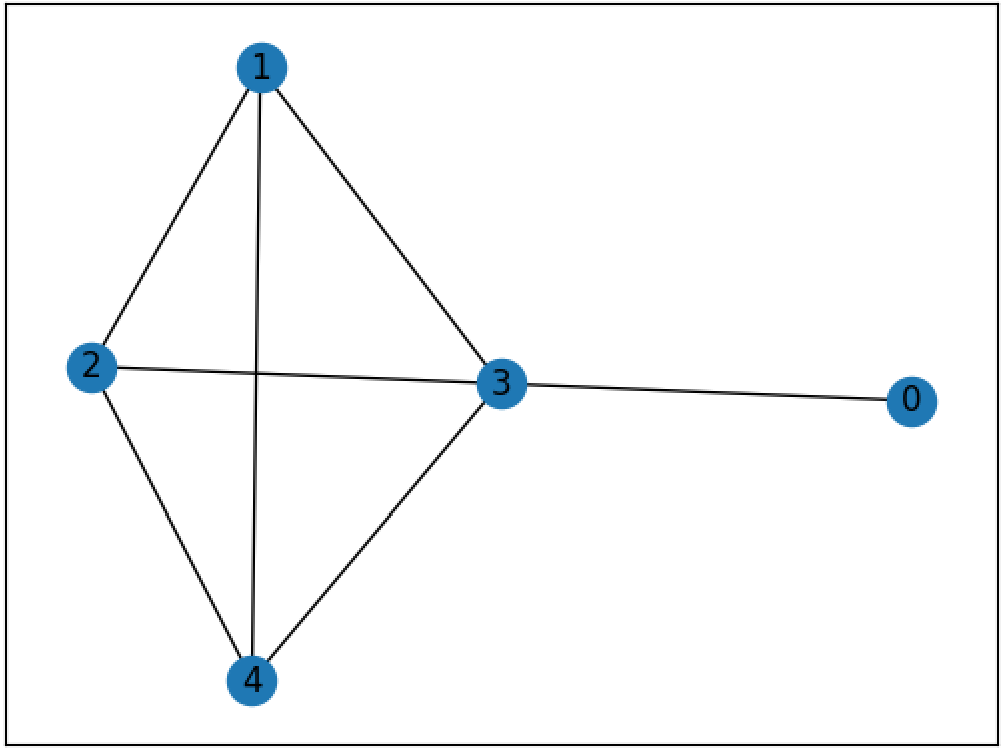
```
[4, 10, 12]
```

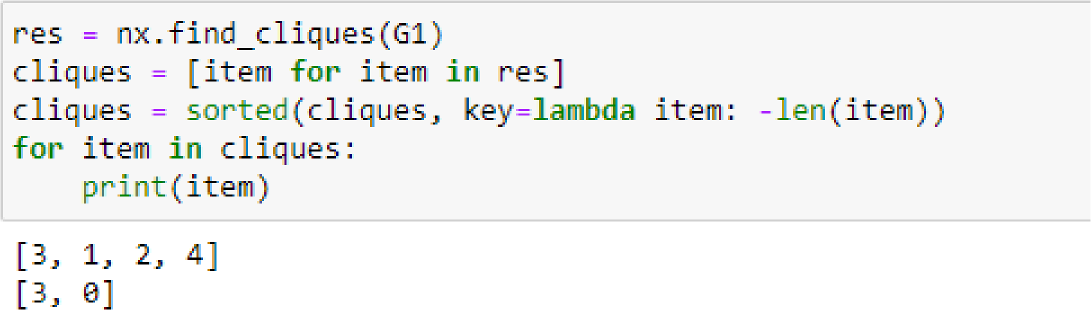


Maximum Clique: sample code



```
import networkx as nx
G1 = nx.Graph()
edges = [(1, 2), (2, 3), (1, 3), (3, 4), (3, 0),(1, 4),(4, 2)]
G1.add_edges_from(edges)
nx.draw_networkx(G1)
```





```
res = nx.find_cliques(G1)
cliques = [item for item in res]
cliques = sorted(cliques, key=lambda item: -len(item))
for item in cliques:
    print(item)
```

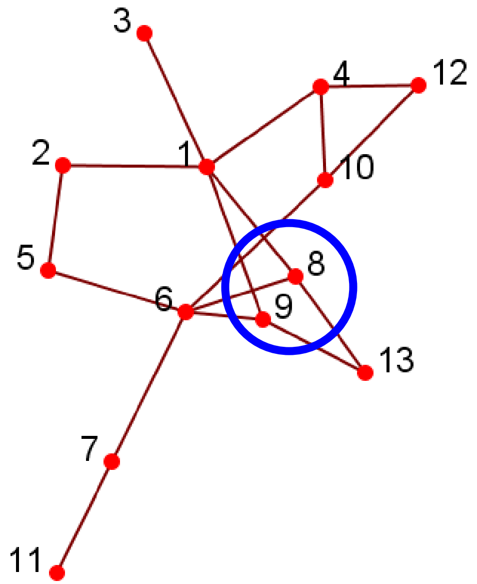
```
[3, 1, 2, 4]
[3, 0]
```

Network-Centric Community Detection

- To form a group, we need to consider the connections of the nodes globally.
- Goal: partition the network into disjoint sets
- Groups based on
  - Node Similarity
  - Cut Minimization
  - Louvain

Node Similarity

- Node similarity is defined by how similar their interaction patterns are
- Two nodes are structurally equivalent if they connect to the same set of actors
    - e.g., nodes 8 and 9 are structurally equivalent
- Groups are defined over equivalent nodes
    - Too strict
    - Rarely occur in a large-scale
    - Relaxed equivalence class is difficult to compute
- In practice, use vector similarity
    - e.g., cosine similarity, Jaccard similarity



Vector Similarity

a vector

structurally equivalent

	1	2	3	4	5	6	7	8	9	10	11	12	13
5		1				1							
8	1					1							1
9	1					1							1

Cosine Similarity:

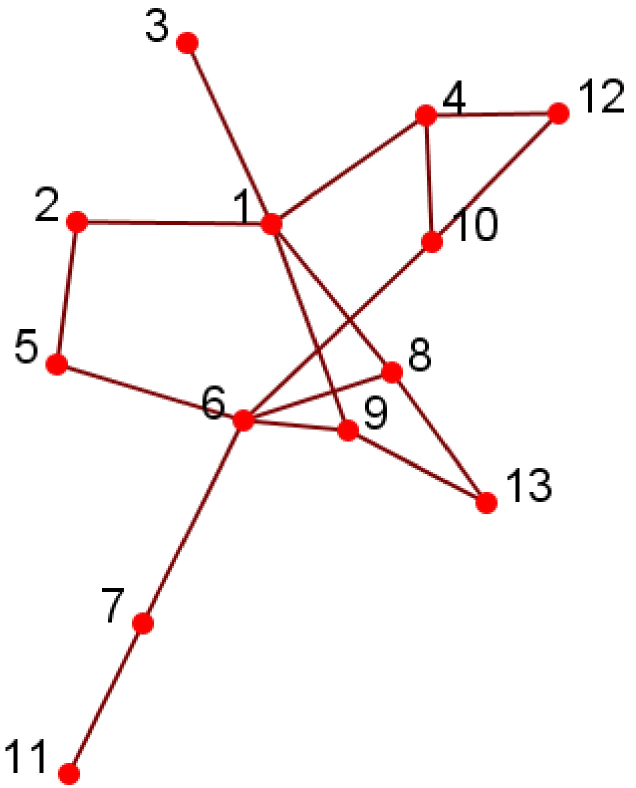
$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}.$$

$$sim(5,8) = \frac{1}{\sqrt{2} \times \sqrt{3}} = \frac{1}{\sqrt{6}}$$

Jaccard Similarity:

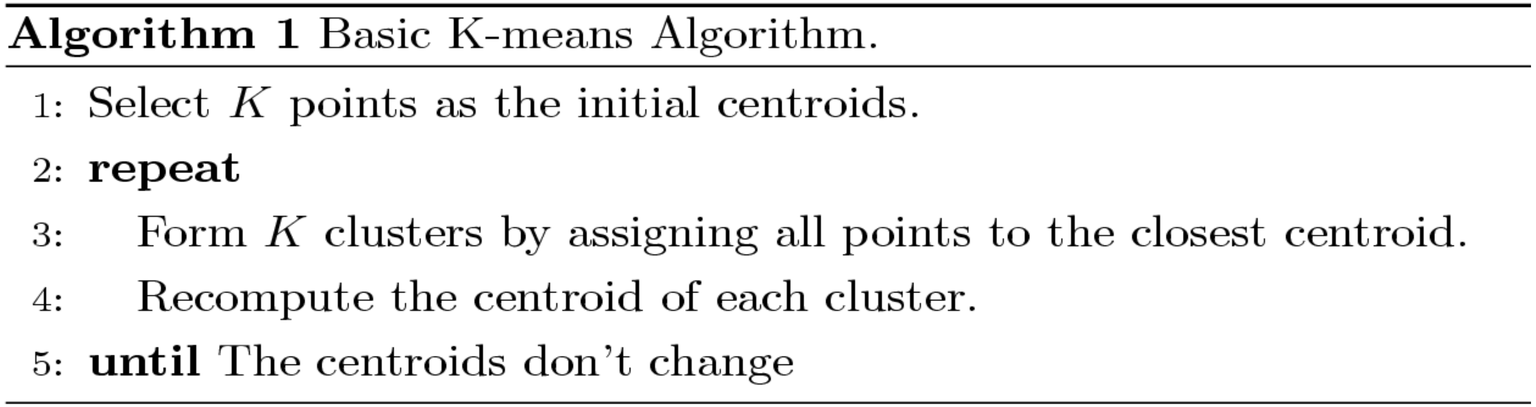
$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

$$J(5,8) = \frac{|\{6\}|}{|\{1,2,6,13\}|} = 1/4$$



Clustering based on Node Similarity

- For practical use with huge networks:
  - Consider the connections as features
  - Use Cosine or Jaccard similarity to compute vertex similarity
  - Apply classical k-means clustering Algorithm
- K-means Clustering Algorithm
  - Each cluster is associated with a centroid (center point)
  - Each node is assigned to the cluster with the closest centroid

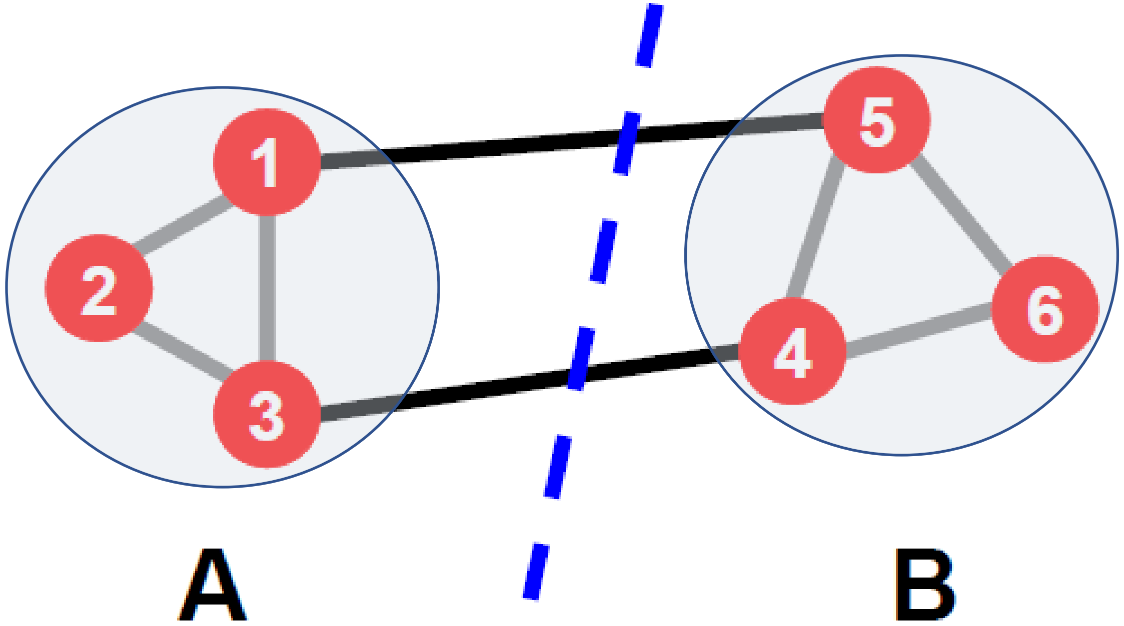


Algorithm 1 Basic K-means Algorithm.

```
Select $K$ points as the initial centroids.
repeat
    Form $K$ clusters by assigning all points to the closest centroid.
    Recompute the centroid of each cluster.
until The centroids don't change
```

Minimum-cut, Normalized-cut

- A Basic principle for graph partitioning
  - Minimize the number of between-group connections
  - Maximize the number of within-group connections



Graph partitioning : A & B

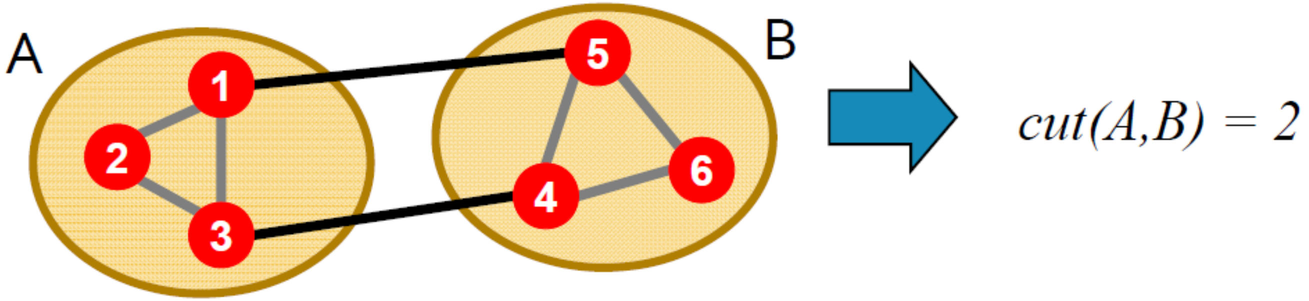
Criterion: Min-cut VS N-cut

- A Basic principle for graph partitioning
  - Minimize the number of between-group connections
  - Maximize the number of within-group connections

	Min-cut	**N-cut**
Minimize: between group connections	✓	✓
Maximize : within-group connections	X	✓

Mathematical expression: Cut (A,B)

- For considering between-group:



$cut(A,B) = 2$

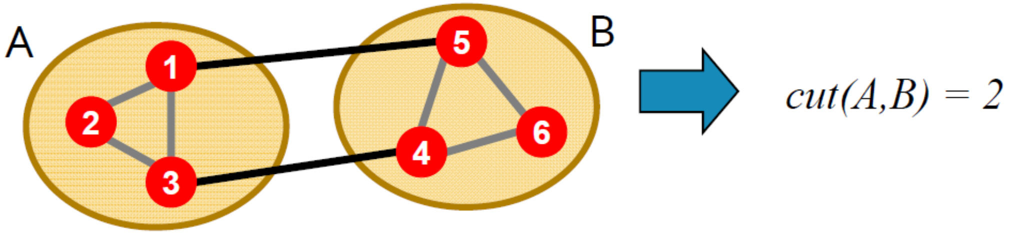
Cut: Set of edges with only one vertex in a group: $cut(A,B) = \sum_{i \in A, j \in B} w_{ij}$

Mathematical expression: Normalized-cut (Ncut)

- For considering within-group
- Assoc(A,V): the total connection from nodes in A to all nodes in the graph

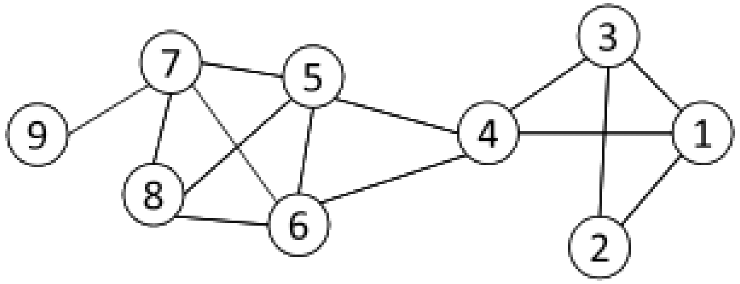
$$assoc(A,V) = \sum_{u\in A,t\in V} w(u,t)$$

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)}$$



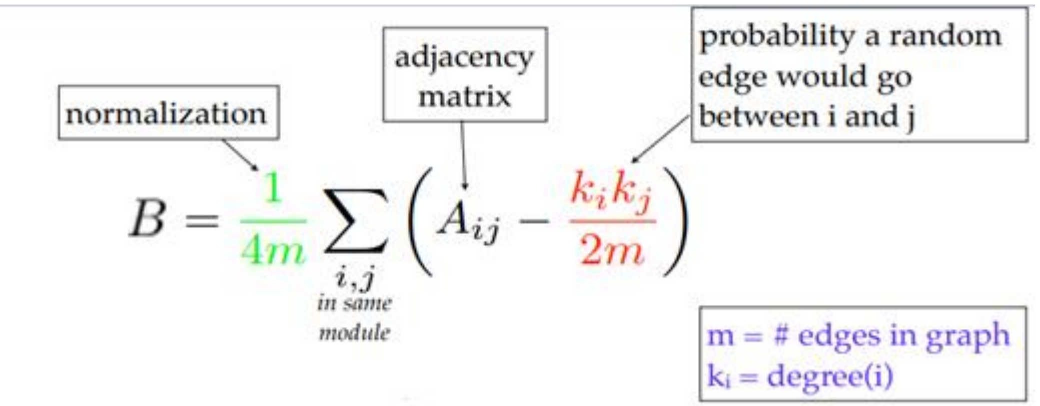
Modularity Maximization

- Modularity measures the strength of a community partition by taking into account the degree distribution
- Given a graph with m edges, the expected number of edges between two nodes with degree d_i and d_j is $d_i d_j / 2m$



The expected number of edges between nodes 1 and 2 is 3*2/ (2*14) = 3/14

- Modularity:



$$B = \frac{1}{4m} \sum_{\substack{i,j \\ \text{in same} \\ \text{module}}} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$$

Modularity Matrix

- We have modularity matrix:

$$B = A - \mathbf{d}\mathbf{d}^T/2m \qquad (B_{ij} = A_{ij} - d_i d_j/2m)$$

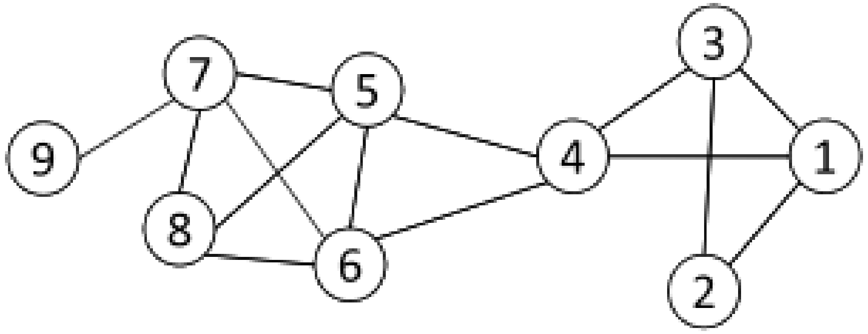
Where: d is node degree.

- Similar to spectral clustering, modularity maximization can be reformulated as

$$\max Q = \frac{1}{2m} Tr(S^T BS) \quad s.t.\ S^T S = I_k$$

- Optimal solution: top eigenvectors of the modularity matrix
- Apply k-means to S as a post-processing step to obtain community partition

Modularity Maximization Example



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

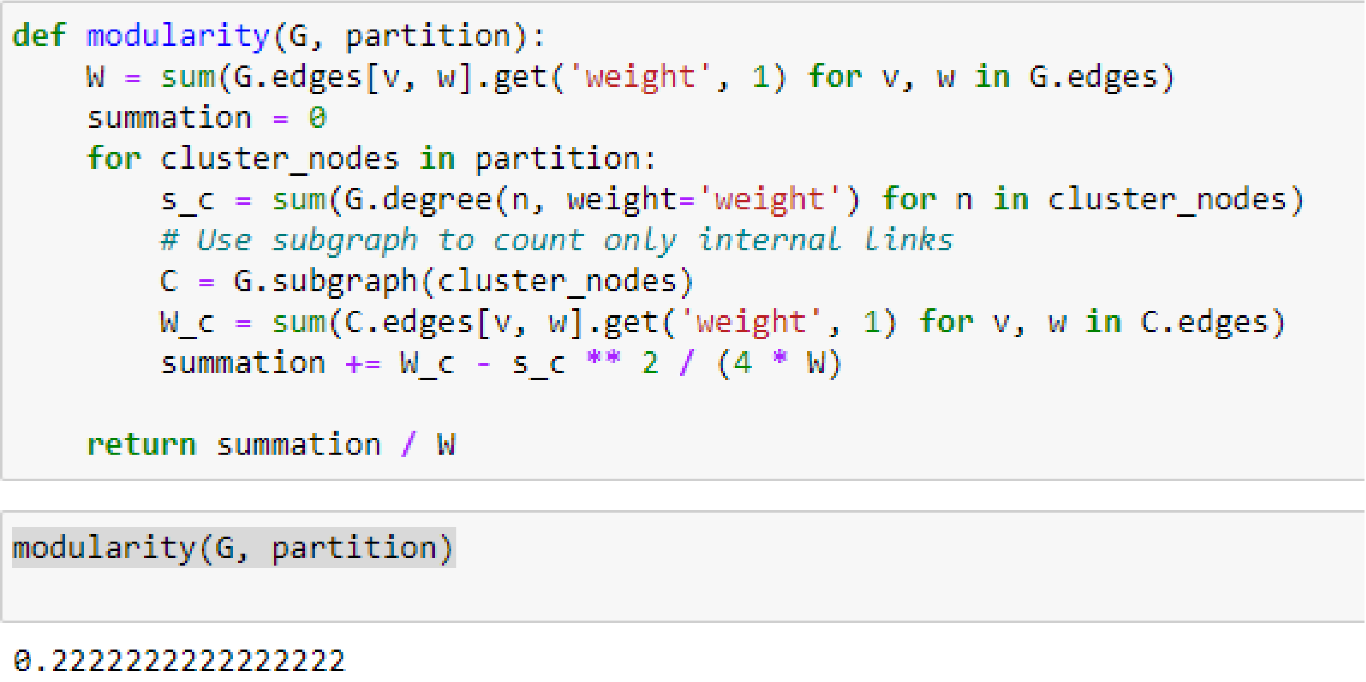
Modularity Matrix

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

k-means

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

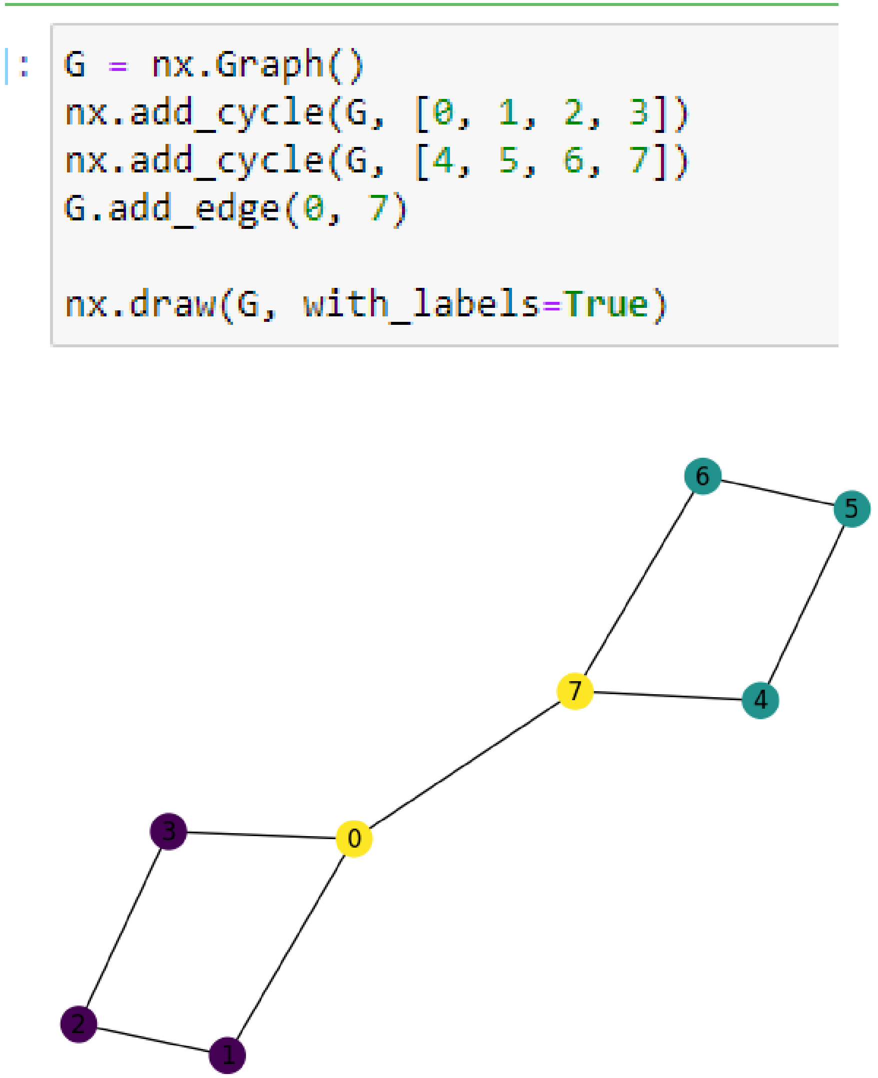
Modularity: sample code



```
def modularity(G, partition):
    W = sum(G.edges[v, w].get('weight', 1) for v, w in G.edges)
    summation = 0
    for cluster_nodes in partition:
        s_c = sum(G.degree(n, weight='weight') for n in cluster_nodes)
        # Use subgraph to count only internal links
        C = G.subgraph(cluster_nodes)
        W_c = sum(C.edges[v, w].get('weight', 1) for v, w in C.edges)
        summation += W_c - s_c ** 2 / (4 * W)

    return summation / W
```

```
modularity(G, partition)
```

```
0.2222222222222222
```



```
G = nx.Graph()
nx.add_cycle(G, [0, 1, 2, 3])
nx.add_cycle(G, [4, 5, 6, 7])
G.add_edge(0, 7)

nx.draw(G, with_labels=True)
```

Hierarchy-Centric 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

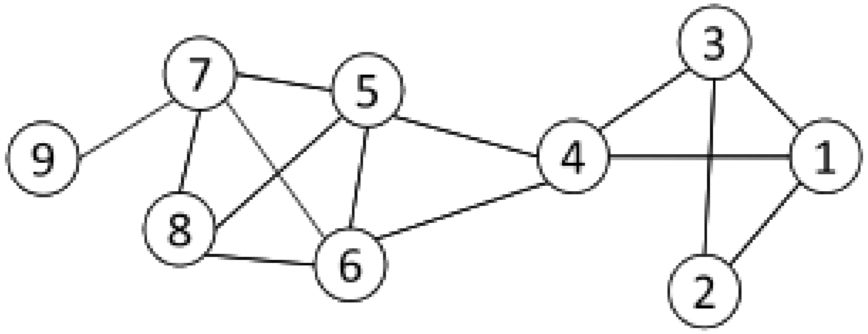
Divisive Hierarchical Clustering

- 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: recursively remove the “weakest” tie
  - Find the edge with the least strength
  - Remove the edge and update the corresponding strength of each edge
- Recursively apply the above two steps until a network is discomposed into desired number of connected components.
- Each component forms a community

Edge Betweenness

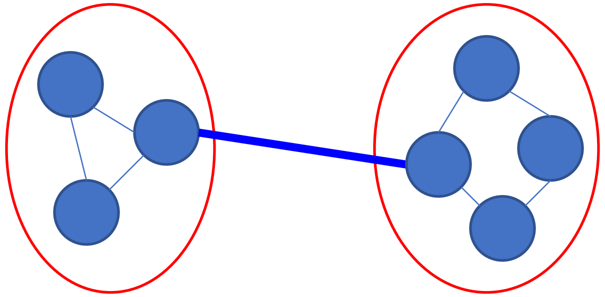
- The strength of a tie can be measured by edge betweenness
- Edge betweenness: the number of shortest paths that pass along with the edge

$$\text{edge-betweenness}(\mathrm{e}) = \Sigma_{s<t} \frac{\sigma_{st}(e)}{\sigma_{s,t}}$$

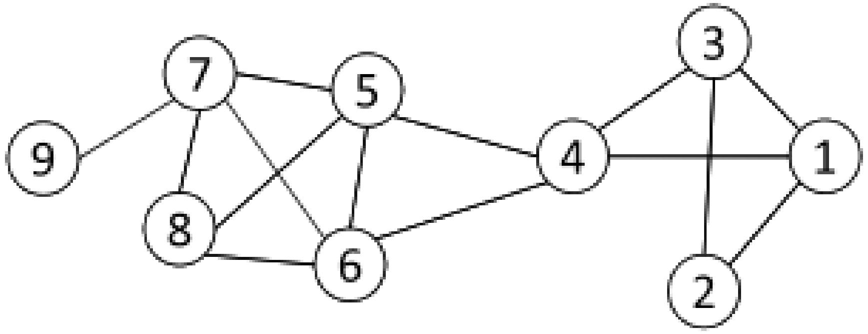


The edge betweenness of e(1, 2) is 4, as all the shortest paths from 2 to {4, 5, 6, 7, 8, 9} have to either pass e(1, 2) or e(2, 3), and e(1,2) is the shortest path between 1 and 2

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



Divisive clustering based on edge betweenness



Initial betweenness value

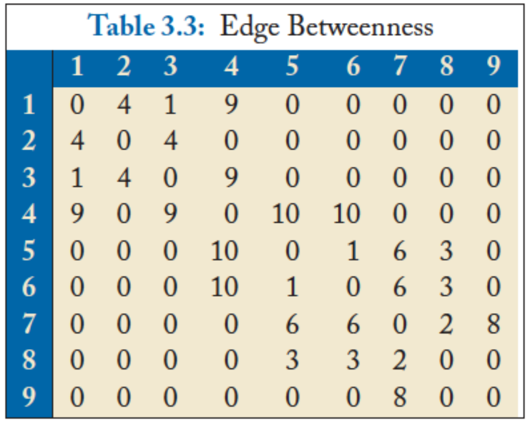
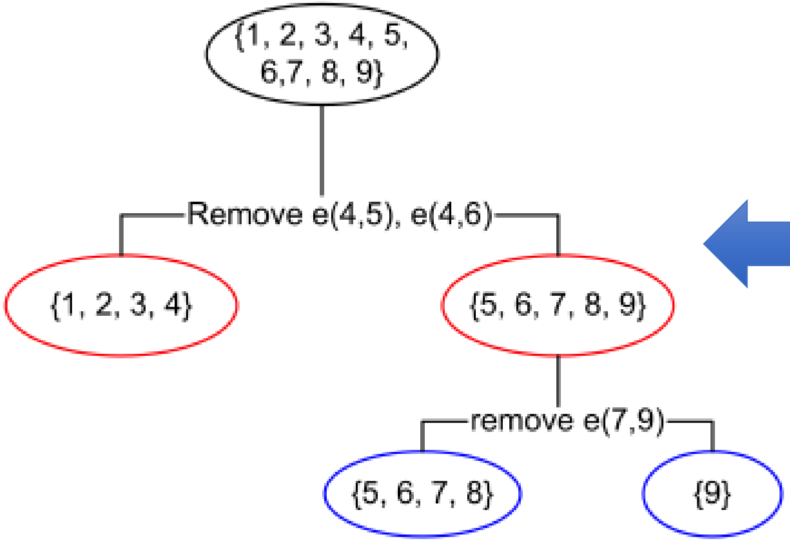


Table 3.3: Edge Betweenness

	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



After remove e(4,5), the betweenness of e(4, 6) becomes 20, which is the highest;

After remove e(4,6), the edge e(7,9) has the highest betweenness value 4, and should be removed.

Louvain method

- Louvain algorithm is an efficient hierarchical clustering algorithm based on graph theory.
- Its principle is to make the modularity of community partition result reach the maximum value through continuous iteration of mobile nodes and then obtain the optimal community partition.

Main steps of the Louvain algorithm

Step 1:

- Initialize the community and set each node as a separate community, namely, community 1: (node1), community 2: (node2), and so on.

Step 2:

- Find out all the communities connected to node 1, and calculate the change of modularity after moving node 2 to each neighbor community. Move node 1 to the community, which can increase the modularity to the maximum.

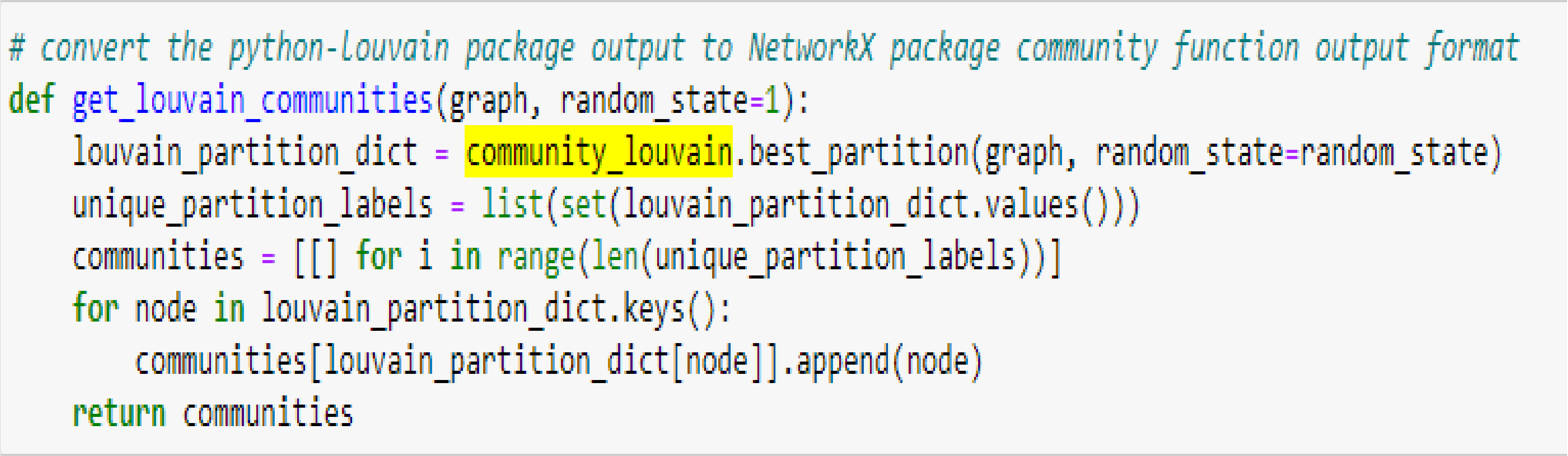
Step 3:

- Iterate over all the nodes and execute step 2 until there are no nodes to move and get a layer of community partition.

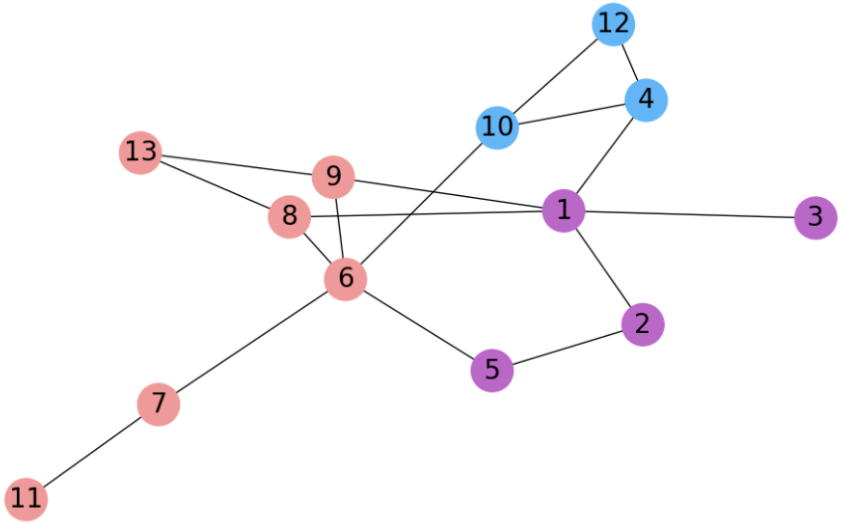
Step 4:

- Merge each community in step 3 into a new node. The relationship between new nodes is the relationship between the original communities. Return to step 1 until all nodes are finally merged into one community.

Louvain algorithm: sample code

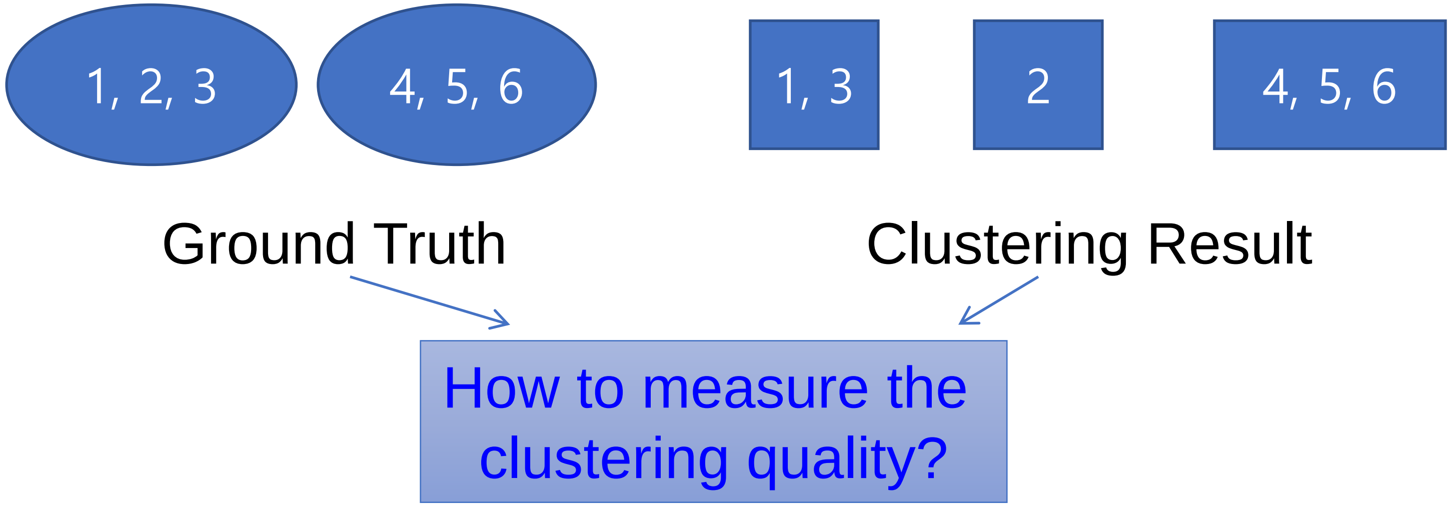


```
# convert the python-louvain package output to NetworkX package community function output format
def get_louvain_communities(graph, random_state=1):
    louvain_partition_dict = community_louvain.best_partition(graph, random_state=random_state)
    unique_partition_labels = list(set(louvain_partition_dict.values()))
    communities = [[] for i in range(len(unique_partition_labels))]
    for node in louvain_partition_dict.keys():
        communities[louvain_partition_dict[node]].append(node)
    return communities
```



Evaluating Community Detection

- The number of communities after grouping can be different from the ground truth
- No clear community correspondence between clustering result and the ground truth
- Normalized Mutual Information can be used



1, 2, 3

4, 5, 6

Ground Truth

1, 3

2

4, 5, 6

Clustering Result

How to measure the clustering quality?

Normalized Mutual Information

- Entropy: the information contained in a distribution

$$H(X) = \sum_{x \in X} p(x) \log p(x)$$

- Mutual Information: the shared information between two distributions

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p_1(x)p_2(y)} \right)$$

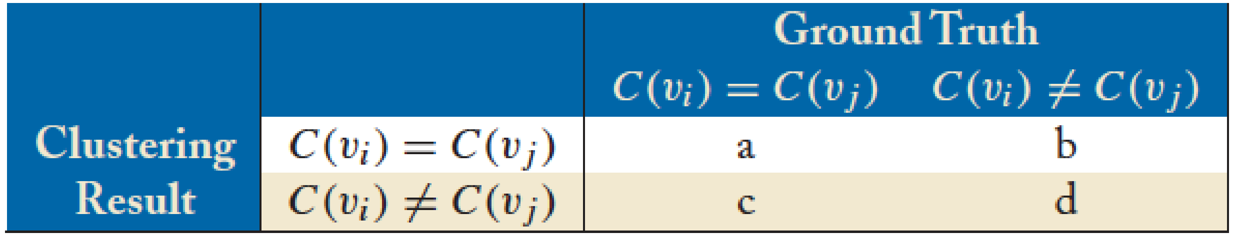
- Normalized Mutual Information (between 0 and 1)

$$NMI(X;Y) = \frac{I(X;Y)}{\sqrt{H(X)H(Y)}}$$

- Consider a partition as a distribution (probability of one node falling into one community), we can compute the matching between two clusterings

Accuracy of Pairwise Community Memberships

- Consider all the possible pairs of nodes and check whether they reside in the same community
- An error occurs if
  - Two nodes belonging to the same community are assigned to different communities after clustering
  - Two nodes belonging to different communities are assigned to the same community
- Construct a contingency table



		Ground Truth	
		$C(v_i) = C(v_j)$	$C(v_i) \neq C(v_j)$
Clustering Result	$C(v_i) = C(v_j)$	a	b
	$C(v_i) \neq C(v_j)$	c	d

$$accuracy = \frac{a+d}{a+b+c+d} = \frac{a+d}{n(n-1)/2}$$

Accuracy Example

1, 2, 3

4, 5, 6

1, 3

2

4, 5, 6

Ground Truth

Clustering Result

		Ground Truth	
		$C(v_i) = C(v_j)$	$C(v_i) != C(v_j)$
Clustering Result	$C(v_i) = C(v_j)$	4	0
	$C(v_i) != C(v_j)$	2	9

Accuracy = (4+9)/ (4+2+9+0) = 13/15

네트워크 과학 연구실
NETWORK SCIENCE LAB

가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA