CS-7280: Network Science

Fiona Ding

OMSCS Summer 2022

Contents

1 Lesson 1: What is network science?

Objectives:

- Define "network science"
- Underestand its history and connections with other disciplines
- Applications of network science

Reading:

- Chapter-1 from A-L. Barabási, Network Science.
- Recommended papers:
  - Networks in Epidemiology: An Integrated Modeling Environment to Study the Co-evolution of Networks, Individual Behavior and Epidemics by Chris Barrett et al.
  - Networks in Biology: Network Inference, Analysis, and Modeling in Systems Biology by Reka Albert
  - Networks in Neuroscience: Complex brain networks: graph theoretical analysis of structural and functional systems by Ed Bullmore and Olaf Sporns
  - Networks in Social Science: Network Analysis in the Social Sciences by Stephen Borgatti et al.
  - Networks in Economics: Economic Networks: The New Challenges by Frank Schweitzer et al.
  - Networks in Ecology: Networks in Ecology by Jordi Bascompte
  - Networks and the Internet: Network Topologies: Inference, Modelling and Generation by Hamed Haddadi et al.

1.1 Lecture Notes

1.1.1 Definition of network science

Definition: The study of complex systems focusing on their architecture, i.e., on the network, or graph, that shows how the system components are interconnected.

1.1.2 Complex Systems

Fundamental properties of the complex systems that network science is interested in:

- Many and heterogeneous components
- Components that interact with each other through a non-trivial network (not connected randomly)
- Non-linear interactions between components

In contrast to complex networks:

- **Regular networks** have the same interconnection pattern for all nodes; regular networks have been studied extensively by mathematicians.
- **Random networks** are networks where the connections between nodes are determined randomly.

Most networks in practice are not regular or random, they have interaction patterns that are highly variable.

1.1.3 Complex Systems and Their Network Representation

Note that mapping an actual complex system to a graph representation is only a model. Some information about the system is lost in the model. Thus, it's always important to ask whether the network representation of a given system has enough detail to answer the questions you are interested in about the system.

1.1.4 Premise of Network Science

The network architecture of a system provides valuable information about the system's function, capabilities, resilience, evolution, etc. We don't need to know all the details about the system and its components to draw conclusions about the system.

1.1.5 Examples of Network Science

- **a** -
- **b** -

1.1.6 Network Centrality

The centrality of a node/edge is a measure of the "importance". There are different metrics that can be used to measure the centrality, eg:

- number of other nodes that - causes largest disruption if removed -

1.2 Book: Chapter 1

2 Lesson 2: Graph Theory

Objectives:

- Review relevant concepts from graph theory and linear algebra
- Review basic graph algorithms
- Explain how these concepts, math, algorithms relate to real-world networks

Reading:

- Chapter-2 from A-L. Barabási, Network Science(Links to external site), 2015.
- Chapter-2 from D. Easley and J. Kleinberg, Networks, Crowds and Markets
- (Recommended) Fibonacci Heap. Wikipedia. `https://en.wikipedia.org/wiki/Fibonacci_heap`
- (Recommended) Kosaraju's Algorithm. Wikipedia. `https://en.wikipedia.org/wiki/Kosaraju%27s_algorithm`

2.1 Lecture Notes

2.1.1 Undirected Graphs

Graph or Network: collection of dyadic relations between a set of nodes (aka vertices). Relations are called edges or links. Represented by $G(V, E)$, where

Undirected Graph
Graphs are often represented by an Adjacency Matrix or an Adjacency List. An adjacency matrix only needs a single memory access to check if an edge exists, but requires n^2 space; an adjacency list requires $n + 2 * m$ where n is the number of nodes and m is the number of edges. For sparse graphs, where the number of edges is closer to n than the max of $n - choose - 2$, the size of an adjacency matrix vs list can differ by a lot.

2.1.2 Walks, Paths, Cycles

A *walk* can visit a node more than once.

A *path* is a walk where the intermediate nodes are distinct.

A *cycle* is a path that starts and ends on the same node.

The number of walks between two nodes of length k is given by A^k, where A is the adjacency matrix.

2.1.3 Connected Components

An undirected graph is **connected** if there exists a path between any two nodes. Many real-world networks are not connected, but consist of multiple **connected components**. BFS can be used to obtain all the nodes in a connected component.

2.1.4 Regular Networks

Trees are connected graphs with no cycles.

A **k-regular graph** is a network where every vertex has the same degree k.

A **complete graph** , or a **clique**, is a special case of a k-regular network where every vertex is connected to every other vertex ($k = n - 1$).

2.1.5 Directed Graphs

Each edge in directed graphs has a starting node and an ending node, ie the edge has a direction. Thus, the adjacency matrix may no longer be symmetric.

The definition of node degree for directed graphs now needs to be split into **in-degree** and **out-degree**.

2.1.6 Weighted Graphs

When edges can have different strengths, we can represent the strenth of the edge with a weight. For undirected graphs, the strength of a node is the sum of weights of all edges adjacent to the node. For directed graphs, we can define both an **in-strength**, the weight of all incoming edges, and an **out-strength**, the weight of all outgoing edges.

Signed graphs can have negative edge weights, which represent competitive interactions.

2.1.7 Strongly Connected Components

In directed graphs, the notion of *connectedness* is different, since there may exist a path from node s to node t but not vice versa.

A directed graph is **strongly connected** if there is a path between all pairs of vertices. A **strongly connected component** of a directed graph is a maximal strongly connected subgraph.

How to determine if a directed graph is strongly connected? The graph is strongly connected iff any node s can reach every other node AND any other node can reach s.

1. Pick any node s.
2. Run BFS from s on the original graph, G and see if the traversal reaches all nodes in G. This checks that s can reach all nodes in the graph.
3. Run BFS from s on the reverse graph, G', with all edges in the opposite direction of G, and see if the traversal reaches all nodes in G'. This checks that all other nodes have a path to s.

Note: see Tarjan's algorithm and Kosaraju's algorithm to compute the set of strongly connected components in a directed graph.

2.1.8 Directed Acyclic Graphs (DAG)

A **DAG** is a directed graph that does not contain any cycles. They are common because they represent generalized hierarchies and dependency networks.

A graph has a **topological ordering** if we can run its nodes such that every edge points from a node of lower rank to a node of higher rank.

Properties of DAGs:

1. If a directed network has a topological ordering it must be a DAG.
2. A DAG must have at least one source node, ie a node that does not contain any incoming edges.

3. If a graph is a DAG it must have a topological ordering.

The first and third properties can also be stated: a directed graph has a topological ordering if and only if it is a DAG.

How to show the above properties:

Topological ordering $\implies$ DAG If the graph had a cycle, there would be an edge from a higher-rank node to a lower-rank node, which violates the topological order property.

At least one source node Start from any node and start moving backwards. Since there are no cycles and the graph has a finite number of nodes, we must eventually reach a source node.

DAG $\implies$ Topological ordering Start from a source node. Remove the node from the graph and decrement the in-degree of all nodes it pointed to. The graph is still a DAG. Choose a new source node and repeat until all nodes are removed.

2.1.9 Dijkstra's Algorithm

The shortest path(s) between a pair of nodes is often useful to know, since they represent the most effiecient way to move within a network.

In unweighted networks, the shortest path can be found in linear time using BFS. If the network is weighted, Dijkstra's algorithm can be used to determine the shortest path.

2.1.10 Random Walks

2.1.11 Min-Cut Problem

A $cut(s,t)$ is a set of edges in a graph such that if those edges were removed it would disrupt all paths between s and t.

The **minimum cut** is the cut with the minimum sum of edge weights.

2.1.12 Max-Flow Problem

The flow from a source to a target node

2.1.13 Bipartite Graphs

A **bipartite graph** is a graph where the set of nodes can be partitioned into two subsets such that every edge connects a node from one subset to the other. No vertices within each subset are connected to each other.

Theorem: A graph is bipartite if and only if it does not include any odd-length cycles.

Bipartite graphs can be used for recommendation systems: eg given a bipartite graph mapping users to the items they bought.

- To find users with similar preferecnces, compute the **one-mode projection** of the bipartite graph onto the set of users. The projection contains only the user nodes, and an edge between two users if they have purchased at least one common item.
- To find items that are often purchased together, compute the **one-mode projection** onto the set of items. Two items are connected with a weighted edge representing the number of users that purchased both items.

The **one-mode projections** can be computed using the adjacency matrix of the bipartite graph, A.

- The **co-citation metric**, $C_{i,j}$, between two nodes i and j, is the number of nodes that have outgoing edges to both i and j, ie, if i and j are items, $C_{i,j}$ is the number of users who purchased both i and j. It can be computed: $C = A^T A$.
- The **bibliographic coupling metric**, $B_{i,j}$, between two nodes i and j, is the number of nodes that receive incoming edges from both i and j, ie, if i and j are users, $B_{i,j}$ is the number of items purchased by both i and j. It can be computed: $B = AA^T$

2.2 Reading Notes

3 Lesson 3: Degree Distribution and the "Friendship Paradox"

Objectives:

- Measure and interpret the degree distribution of a network

- Understand the “friendship paradox” to illustrate the importance of the degree distribution
- Learn the G(n,p) model as the most basic type of random graph
- Degree correlations and assortative networks

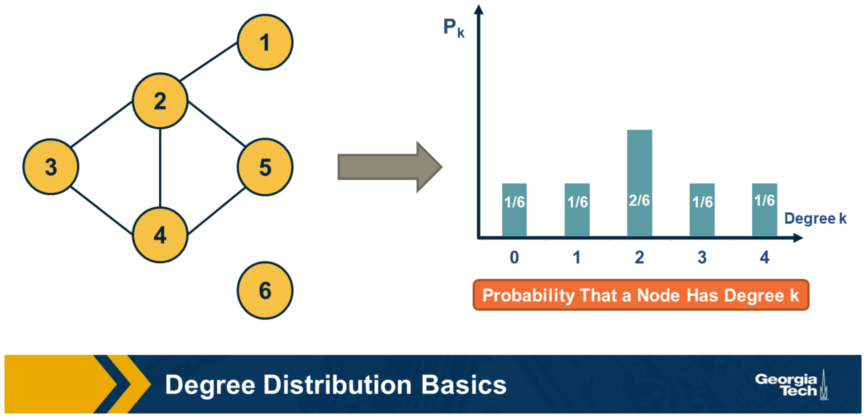
Reading:

- Chapter 3 from A-L. Barabási, Network Science.
- Chapter 7, Sections 7.1, 7.2, 7.3, 7.5 from A-L. Barabási, Network Science.
- (Recommended) Simulated Epidemics in an Empirical Spatiotemporal Network of 50,185 Sexual Contacts

3.1 Lecture Notes

3.1.1 Degree Distribution

The degree distribution shows the fraction of nodes with degree k.


1
2
3
4
5
6
P_k
1/6
1/6
2/6
1/6
1/6
Degree k
0
1
2
3
4
Probability That a Node Has Degree k
Degree Distribution Basics
Georgia Tech

3.1.2 Degree Distribution Moments

Given the probability distribution of a random variable, we can compute:

- **First Moment (aka mean):**

$$\bar{k} = \sum_{k=0}^{maxdegree} k p_k$$

- **Second Moment:**
$$\bar{k} = \sum_{k=0}^{maxdegree} k^2 p_k$$
- **Variance:**
$$\sigma_k^2 = \overline{k^2} - (\bar{k})^2$$

3.1.3 Complementary Cumulative Distribution Function

For larger networks, we usually use the Complementary Cumulative Distribution Function (C-CDF) instead of the empirical probability density function, p_k, which shows the probability that the degree is at least k, for any $k > 0$.

$$\overline{P_k} = Prob(degree >= k) = \sum_{x=k}^{\infty} p_x$$

C-CDF plots ($\overline{P_k}$ vs k)are often shown with log or log-log scales, since for many networks, the C-CDF decays exponentially (ie $\overline{P_k} = e^{-k\lambda}$) or decays with a power-law of k (ie $\overline{P_k} = ck^{-\alpha}$).

3.1.4 Friendship Paradox

Definition: The average degree of a node's neighbor is higher than the average node degree of the network. Informally, on average, your friend has more friends than you.

Proof: probability that a random edge connects to a node of degree k Suppose we pick a random edge and randomly select one of the two stubs of the edge. The probability q_k that a randomly chosen stub belongs to a node of degree k is:

$$\begin{aligned} q_k &= (\text{\# of nodes of degree k} \times (\text{probability an edge connects to a specific node of degree k})) \\ &= (np_k)(\frac{k}{2m}) \\ &= \frac{kp_k}{2m/n} \\ &= \frac{kp_k}{\bar{k}} \end{aligned}$$

where n is the number of nodes and m is the number of edges.

The probability that a randomly chosen stub connects to a node of degree k is proportional to both k and the probability that a node has degree k.

For nodes with degree $k > \bar{k}$, it is more likely to sample one of their stubs than the nodes themselves, and vice versa for nodes with degree $k < \bar{k}$.

Proof: expected value of a neighbor's degree

$$
\begin{aligned}
\overline{k_{nn}} &= \sum_{k=0}^{\text{max degree}} k \cdot q_k \\
&= \sum_k k \frac{k p_k}{\bar{k}} \\
&= \frac{\sum_k k^2 p_k}{\bar{k}} \\
&= \frac{\overline{k^2}}{\bar{k}} \\
&= \frac{(\bar{k})^2 + (\sigma_k)^2}{\bar{k}} \\
&= \bar{k} + \frac{\sigma_k^2}{\bar{k}}
\end{aligned}
$$

As long as the variance of the degree distribution is positive, and given our assumption that neighboring nodes have independent degrees, the average neighbor's degree is higher than the average node degree.

3.1.5 Edge Cases of The Friendship Paradox

Infinitely large regular network All nodes have the same degree, and thus the degree variance is 0. The average neighbor's degree is equal to the average node degree.

Infinitely large star network One hub node at the center, and all peripheral nodes connecting only to the hub. The degree variance diverges as n increases, as does the difference between the average node degree and the average neighbor degree.

3.1.6 Practical Application of Friendship Paradox

Suppose we have an epidemic, where a virus is spreading through a social network. Assume there is no natural immunity. The virus would eventually spread through the entire network.

Say there is a vaccine: if we can't vaccinate everyone, who should we choose to vaccinate to minimize the spread? We could choose the people with the highest degrees, but that's not always easy to determine in a real-world network. Instead, we could use the "acquaintance immunization" strategy, based on the Friendship Paradox:

1. Choose a set of random nodes v
2. Ask each node in v to identify "most connected" neighbor u
3. Immunize u

If the network contains any hubs, they will likely be connected to some of the randomly selected nodes.

3.1.7 The G(n,p) model (ER Graphs)

The G(n,p) model (aka Gilbert Model, aka Erdos-Renyi (ER) Model) is the simplest random graph model. It is a network with n nodes, where the probability that any two nodes are connected is p.

Assuming no self-edges allowed:

- **Expected number of edges:** $p \cdot \frac{n(n-1)}{2}$
- **Average node degree:** $p \cdot (n-1)$
- **Density of network:** p
- **Degree variance:** $p \cdot (1-p) \cdot (n-1)$

The degree distribution of the G(n,p) model follows the Binomial(n-1, p) distribution since each node can be connected to $n-1$ other nodes with probability p.

The degrees of neighboring nodes are not correlated, so the average neighbor degree of a G(n, p) network is $\overline{k_{nn}} = \bar{k} + (1-p)$, using the binomial distribution. This is $\overline{k_{nn}} = \bar{k} + 1$ using Poisson's approximation when $p \ll 1$ (true for sparse networks). In other words, if we reach a node v by following an edge from another node, the expected value of v's degree is one more than the average node degree.

Connected Components in G(n, p) There is no guarantee that the G(n, p) model will give a connected network. Note: connection probability is $p \approx \frac{\bar{k}}{n}$.

If p is close to zero, the network consists of many small components.

When the average degree approaches 1, the largest connected component starts covering a significant fraction of all network nodes, in one "giant component".

Derivation: size of LCC in G(n, p) as a function of p Let S be the probability that a node belongs to the LCC, aka the expected value of the fraction of nodes that belong in the LCC.

Then, $\overline{S} = 1 - S$ is the probability a node is not in the LCC.

That probability can be written:

$$\overline{S} = ((1-p) + p \cdot \overline{S})^{n-1}$$

The first term is for the case that a node v is not connected to another node, and the second term for the case when v is connected to another node that is *not* in the LCC.

Since $p = \frac{\bar{k}}{n-1}$:

$$\overline{S} = (1 - \frac{\bar{k}}{n-1}(1-\overline{S}))^{n-1}$$

$$\begin{aligned}\ln \overline{S} &= (n-1)\ln(1 - \frac{\bar{k}}{n-1}(1-\overline{S})) \\ &\approx -(n-1)\frac{\bar{k}}{n-1}(1-\overline{S})\end{aligned}$$

$$S = 1 - e^{-\bar{k}S}$$

Phase transition at $\bar{k} = 1$

When does G(n,p) have a Single Connected Component? How large does p (or $\bar{k}$) need to be so that the LCC covers all nodes?

Suppose that S is the probability that a node belongs in the LCC.

Then, the probability that a node does NOT connect to ANY node in the LCC:

$$(1-p)^{Sn} \approx (1-p)^n$$

if $S \approx 1$.

The expected number of nodes not connecting to LCC:

$$\overline{k_0} = n(1-p)^n = n(1-\frac{np}{n})^n$$

Recall that $(1-\frac{x}{n})^n \approx e^{-x}$ when $x \ll n$. We assume that the network is sparse ($p \ll 1$):

$$\overline{k_0} \approx ne^{-np}$$

If we set $\overline{k_0}$ to less than one node:

$$ne^{-np} \leq 1$$

$$-np \leq \ln(\frac{1}{n}) = -\ln n$$

$$p \geq \frac{\ln n}{n}$$

$$\bar{k} = np \geq \ln n$$

which means that when the average degree is higher than the natural logarithm of the network size ($\bar{k} > \ln n$) we expect to have a single connected component.

3.1.8 Degree Correlations

So far this lesson, we assumed that the degree of a node does not depend on the degree of its neighbors (ie **no degree correlations**, aka **neutral networks**). In mathematical terms, if nodes u and v are connected, we have assumed:

$$P(\text{degree}(u)|\text{degree}(v) = k') = P(\text{degree}(u) = k|u \text{ connects to another node}) = q_k = p_k\frac{k}{\bar{k}}$$

Note: this probability does not depend on the degree k' of neighbor v.

In general, however, there *are* correlations between the degrees of neighboring nodes, and they are described by the conditional probability distribution:

$$P(k'|k) = P(\text{a neighbor of a k-degree node has degree k'})$$

The expected value of this distribution is referred to as the **average nearest-neighbor degree**, $k_{nn}(k)$, of degree-k nodes:

$$k_{nn}(k) = \sum_{k'} k' \cdot P(k'|k)$$

Recall that we already derived that for a neutral network, $k_{nn}(k)$ is independent of k:

$$k_{nn}(k) = \bar{k} + \frac{\sigma_k^2}{\bar{k}} = \bar{k}_{nn}$$

In most real networks, $k_{nn}(k)$ *does* depend on k.

How To Measure Degree Correlations One way to quantify the degree correlations in a network is by modeling (i.e., approximating) the relationship between the average nearest neighbor degree $k_{nn}(k)$ and the degree k with a power law of the form:

$$k_{nn}(k) \approx a \cdot k^{\mu}$$

Then we can estimate the exponent, μ, from the data.

1. **Assortative:** If $\mu > 0$, the network is assortative (higher-degree nodes tend to have higher-degree neighbors and lower-degree nodes tend to have lower-degree neighbors).
2. **Disassortative:** If $\mu < 0$, the network is assortative (higher-degree nodes tend to have lower-degree neighbors).
3. **Neutral:** If μ is statistically not significantly different from zero, we say that the network is neutral.

3.1.9 Assortative, Neutral, Disassortative Networks

Examples:

- **Assortative:** Scientific collaboration: nodes (scientists) are connected if they have written at least one research paper together.
- **Disassortative:** Metabolic network: nodes (metabolites) are connected if they appear on opposite sides of the same chemical reaction in a biological cell.
- **Neutral:** Power grid

3.2 Reading Notes

3.2.1 subsection

4 Lesson 4: Random Networks

Objectives:

- Examples of highly skewed degree distributions
- Understand the math of power-law distributions and the concept of "scale-free" networks
- Learn about models that can generate networks with power-law degree distribution

Reading:

- Chapter 4 (sections 4.1, 4.2, 4.3, 4.4., 4.7, 4.8, 4.12) from A-L. Barabási, Network Science.
- Chapter 5 (sections 5.1, 5.2, 5.3) from A-L. Barabási, Network Science.

4.1 Lecture Notes

4.1.1 Power Law Degree Distribution

Most real-life networks are highly-skewed, very different from the G(n, p) networks which follow a binomial distribution. Instead, a power-law distribution is a better model.

- **Degree distribution:** $p_k = ck^{-\alpha}$ where $\alpha > 0$
- **Proportionality coefficient:** calculated so that the sum of all degree probabilities is equal to 1 for a given α and a given minimum degree k_{min} .

$$c = (\alpha - 1)k_{min}^{\alpha-1}$$

- Thus, the complete power-law distribution is:

$$p_k = \frac{\alpha - 1}{k_{min}}(\frac{k}{k_{min}})^{-\alpha}$$

- **C-CDF:**

$$P\left[\text{degree} \geq k\right] = (\frac{k}{k_{min}})^{\alpha-1}$$

4.1.2 Max Degree in Power Law Network

$$k_{max} = k_{min} n^{\frac{1}{\alpha-1}}$$

This means that the maximum degree in a power-law network increases as a power-law of the network size n. If $\alpha = 3$ the maximum degree increases with the square-root of n.

4.1.3 Failures in Power Law Network

Random Failures: a fraction of randomly selected nodes are removed from the network

Targeted Attacks: a fraction of nodes with the highest degree are removed from the network

Power-law networks are more robust to random failures than Poisson networks – but they are also more sensitive to targeted attacks than Poisson networks.

4.1.4 Degree-Preserving Randomization

To determine whether an interesting property P of a network G is because of its degree distribution or some other property of the network, we can perform randomization of G but preserve the degree of all nodes, then check for property P.

4.1.5 Average Degree of Nearest Neighbor in Power Law Network

Recall from lesson 3: for a neutral network (no correlation between the degrees of two connected nodes), the average neighbor degree is:

$$\overline{k_n} = \bar{k} + \frac{\sigma_k^2}{\bar{k}}$$

Since power law networks can have very high degree variability (ie $\sigma_k \gg \bar{k}$), then the average neighbor degree can be much higher than the network's average degree (intensifying friendship paradox).

4.1.6 Generation Models

1. **Configuration model**: Given the number of nodes, n, and degree k_i of each node i, the model generates the n nodes, where node i has k_i available "edge stubs". It then continually selects two random available stubs and connects them with an edge until there are no more available stubs. Note that this might create self-loops or multi-edges.

2. **Preferential Attachment Model**: (aka PA model aka Barabasi-Albert model) The network grows by one node at each time step. Each time a new node is added, it connects to m existing nodes, chosen randomly but with a non-uniform probability distribution. The probability that the new node at time t will connect to node i is:

   $$\Pi_i(t) = \frac{k_i(t)}{2mt}$$

   . New nodes are more likely to connect to nodes with higher degrees.

3. **Link Selection Model** Each time a new node is introduced, select a random link; the new node will connect to one of the two end-points of that link, chosen randomly.

4.2 Reading Notes

4.2.1 subsection

5 Lesson 5: Network Paths, Clustering, and the "Small World" Property

Objectives:

- **Network efficiency**: concept + metrics
- **Network clustering**: concept + metrics
- Difference between "small-world" networks and random/regular, power-law networks
- **Network motifs**: concept + metrics
- Case studies of small-world networks

Reading:

- Chapter 3, Sections 8-9 from A-L. Barabási, Network Science.
- Sections 3.1, 3.2, 20.1, 20.2 - D. Easley and J. Kleinberg, Networks, Crowds and Markets

5.1 Lecture Notes

5.1.1 Clustering Coefficient

In social networks, if A is a friend of B and C, then B and C are also likely to be friends with each other. In other words, A, B, and C form a **"friendship triangle"**.

To quantify the presence of such triangles, we can use the **Clustering Coefficient**, defined for node-i with at least two neighbors as the fraction of its neighbors' pairs that are connected.

For an undirected, unweighted network described by adjacency matrix A, the clustering coefficient for node-i is:

$$\begin{aligned} C_i &= \frac{1/2 \sum_{j,m} A_{i,j} A_{j,m} A_{m,i}}{k_i(k_i-1)/2} \\ &= \frac{\sum_{j,m} A_{i,j} A_{j,m} A_{m,i}}{k_i(k_i-1)} \end{aligned}$$

The denominator is the number of neighbor pairs of node-i and the numerator is the number of those pairs that form triangles with node-i.

The clustering coefficient is not well-defined for nodes of degree zero or one.

The clustering coefficient is maximized (1) if node-i and its neighbors form a clique, while the clustering coefficient is minimized (0) if node-i and its neighbors form a star.

To describe the clustering coefficient of the whole network, rather than one specific node, we can plot the average clustering coefficient average clustering coefficient as a function of the degree, k, for all nodes with degree $k > 1$.

5.1.2 Average Clustering and Transitivity Coefficient

To quantify the degree of clustering in the entire network with a single number, there are two options:

- **Average Clustering Coefficient** for all nodes with degree > 1.

- **Transitivity (aka Global Clustering Coefficient)**: the fraction of the connected triplets of nodes that form triangles. A **connected triplet** is an ordered set of three nodes (ABC) such that A connects to B and B connects to C. An A, B, C triangle corresponds to three triplets: ABC, BCA, and CAB. The transitivity of the network is defined mathematically as:

$$T = \frac{3 \times \text{number of triangles}}{\text{number of connected triplets}}$$

Note: the Transitivity and the Average Clustering Coefficient are two different metrics; while often they can be close, there are certain cases where the two metrics give very different answers. For example: consider a network in which two nodes, A and B, are connected to each other as well as all other nodes, with no other links. For a large number of nodes, n, the average clustering coefficient approaches 1: since all nodes other than A and B are only connected to A and B, the clustering coefficient of those nodes are 1. The transitivity, however, is much closer to zero, since any pair of nodes other than A or B forms a connected triplet but not a triangle.

5.1.3 Clustering in Weighted Networks

The definition of clustering coefficient can also be generalized for weighted networks as follows. Suppose that $w_{i,j}$ is the weight of the edge between nodes i and j.

First, in weighted networks, instead of the degree of a node we often talk about its "strength", defined as the sum of all weights of the node's connections:

$$s_i = \sum_j A_{i,j} w_{i,j}$$

Then, the weighted clustering coefficient of node i is defined as:

$$C_w(i) = \frac{1}{s_i(k_i - 1)} \sum_{j,h} \frac{w_{i,j} + w_{i,h}}{2} A_{i,j} A_{i,h} A_{j,h}$$

The normalization term $\frac{1}{s_i(k_i-1)}$ is such that the maximum value of the weighted clustering coefficient is one. The product of the three adjacency matrix elements at the right is one only if the nodes i, j, h form a triangle. In that case, that triangle contributes to the clustering coefficient of node-i based on the average weight of the two edges that connect node i with j and h, respectively. Note that the weight between nodes j and h does not matter.

5.1.4 Clustering in G(n,p) networks

How large is the expected clustering coefficient at a random ER network?

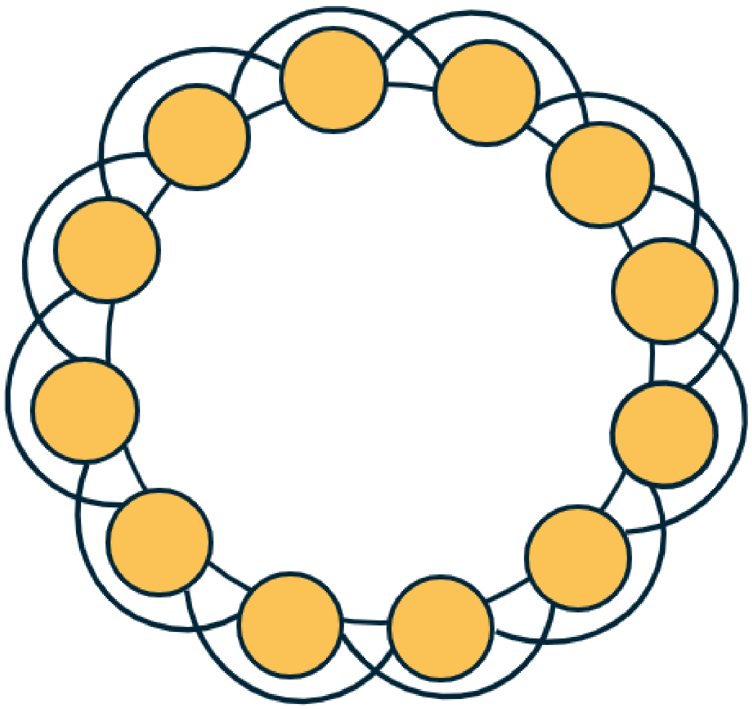
Recall that any two nodes in that model are connected with the same probability p. So, the probability that a connected triplet A-B-C forms a triangle (A-B-C-A) is also p. Thus, the expected value of the clustering coefficient for any node with more than one connection is p. Similarly, the transitivity (and the average clustering coefficient) of the whole network is also expected to be p.

In $G(n, p)$, the average degree is $\bar{k} = p(n-1)$. If the average degree remains constant, we expect p to drop inversely proportional with n as the network size grows.

Real networks do not show a decreasing trend between the clustering coefficient tand the network size, the way we'd expect if they followed the $G(n, p)$ model. The $G(n, p)$ model predicts negligible clustering, expecially for large and sparse networks. On the contrary, real-world networks have a much higher clustering coefficient than $G(n, p)$, and its magnitude does not seem to depend on the network size.

5.1.5 Clustering in Regular Networks

Regular networks typically have a locally clustered topology. The exact value of the clustering coefficient depends on the specific network type but in general, it is fair to say that "regular networks have strong clustering". Further, the clustering coefficient of regular networks is typically independent of their size.



To see that, let's consider the common regular network topology shown at the visualization. The n nodes are placed in a circle, and every node is connected to an even number c of the nearest neighbors (c/2 at the left and c/2 at the right). If c=2, this topology is simply a ring network with zero clustering (no triangles). For a higher (even) value of c however, the transitivity coefficient is:

$$T = \frac{3(c-2)}{4(c-1)}$$

Note this does not depend on the network size. As c increases the transitivity approaches 3/4.

5.1.6 Diameter, Characteristic Path Length, and Network Efficiency

The notion of small-world networks depends on two concepts: how clustered the network is (see above) and how short the paths between network nodes are.

If a network forms a connected component, we can compute a shortest-path length, $d_{i,j}$ between any two nodes i and j. One way to summarize these distances for the entire network is to simply take the average such distance across all distinct node pairs. This metric L is referred to as **Average (shortest) Path Length (APL)** or **Characteristic Path Length (CPL)** of the network. For an undirected and connected network of n nodes, we define L as:

$$L = \frac{2}{n(n-1)} \sum_{i<j} d_{i,j}$$

A related metric is the harmonic mean of the shortest-path lengths across all distinct node pairs, referred to as the **efficiency** of the network:

$$E = \frac{2}{n(n-1)} \sum_{i<j} \frac{1}{d_{i,j}}$$

The efficiency varies between 0 and 1.

Another metric that is often used to quantify the distance between network nodes is the diameter, which is defined as the maximum shortest-path distance across all node pairs:

$$D = max_{i<j} d_{i,j}$$

A more informative description is the distribution of the shortest-path lengths across all distinct node pairs.

5.1.7 Diameter and CPL of G(n,p) Networks

Approximation of diameter We can derive an approximate expression for the diameter of the $G(n, p)$ network as follows:

Suppose the average degree is:

$$\bar{k} = (n-1)p > 1$$

so that the network has a giant connected component. We will further assume that the topology of the network is a tree.

Start from node i. Within one hop away from that node, we expect to visit $\bar{k}$ nodes. Within two hops, we expect to visit approximately $\bar{k}^2$ nodes. Similarly, after s hops, we expect to visit approximately $(\bar{k})^s$ nodes.

The total number of nodes in the network is n, and we expect to visit all of them with the maximum number of hops that is possible, which is the network diameter D. So, $n \approx (\bar{k})^D$. Solving for D:

$$D \approx \frac{\ln n}{\ln \bar{k}}$$

In a random network, even the longest shortest-paths are expected to grow very slowly (logarithmically) with the size of the network.

More accurate estimates of diameter for a sparse G(n,p) network As k approaches 1 from above (ie the network is still expected to have a large connected component), the diameter is expected to be:

$$D \approx 3\frac{\ln n}{\ln \bar{k}}$$

More accurate estimates of diameter for a dense G(n,p) network For dense networks, a good approximation for the diameter is:

$$D \approx \frac{\ln n}{\ln \bar{k}} + \frac{2\ln n}{\bar{k}} + \ln n \frac{\ln \bar{k}}{(\bar{k})^2}$$

Note the diameter is still increasing with the logarithm of the network size, so the main qualitative conclusion remains: the diameter of G(n,p) networks increases very slowly (logarithmically) with the number of nodes – and so the CPL cannot increase faster than that either.

5.1.8 Diameter and Efficiency of G(n,p) Versus Regular Networks

In one-dimensional lattices, each node has two neighbors, and the lattice is a line network of n nodes – so the diameter increases linearly with n.

In two dimensions, each node has 4 neighbors, and the lattice is a square with $\sqrt{n}$ nodes on each side, so the diameter increases as $O(n^{1/2})$.

In lattice networks, the diameter and CPL grows as a power law of the network size, which grows much faster than a logarithmic function.

5.1.9 What is a "small-world" network?

Real world networks often have:

- Strong clustering: similar to that of lattice networks with the same average degree
- Short paths: similar to the CPL and diameter of G(n,p) networks with the same size n and density p

These are referred to as **small-world networks**. They are small in the sense that the shortest path increases logarithmically as a function of the size of the network.

Watts Strogatz Model Start with regular network. With small probability p, choose an edge and reassign one of the stubs to a new randomly chosen node. As rewiring probability approaches 1, the network becomes more random.

5.1.10 Clustering in PA Model

Preferential attachment model: TODO

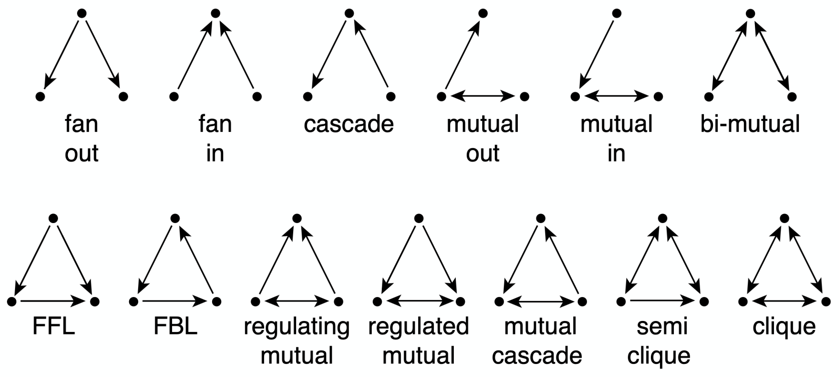
5.1.11 Length of Shortest Paths in PA Model

TODO

5.1.12 Path Lengths in Power-law Networks

TODO

5.1.13 Directed Subgraph Connectivity Patterns



This figure shows all 13 types of connection patterns between three weakly connected network nodes. Each of these patterns will be referred to as **network motifs**.

5.1.14 Statistical Test For The Frequency of a Network Motif

What does it mean when a specific network motif occurs very frequently? Or the opposite, what does it mean when a network motif occurs much less frequently than expected based on chance?

TODO

5.1.15 Frequent Motifs and Their Function

5.2 Reading Notes

5.2.1 subsection

5.2.2

6 Lesson 6: Centrality Metrics and Network-Core Metrics and Algorithms

Objectives:

- Understand the problem of finding individually important nodes/edges or groups of important nodes/edges
- Define and analyze common centrality metrics
- Define and analyze common "network core" metrics
- Learn how to apply centrality metrics and core identification algorithms through case studies

Reading:

- Chapter 7, Sections 1-8 from M.E.J. Newman, Networks: An Introduction.
- Section 14.3 - D. Easley and J. Kleinberg, Networks, Crowds and Markets
- (Recommended) Sabrin, K, Dovrolis, C. The Hourglass Effect in Hierarchical Dependency Networks, Journal of Network Science, (2017)
- (Recommended) Faskowitz, J., Yan, X., Zuo, X. et al. Weighted Stochastic Block Models of the Human Connectome across the Life Span. Sci Rep 8, 12997 (2018).

6.1 Lecture Notes

6.1.1 Degree, Eigenvector and The Katz Centrality

Degree and Strength Centrality The simplest way to define the importance of a network node is based on its number of connections - the more connections a node has, the more important it is. So, the **degree centrality** of a node is simply the degree of that node.

For weighted networks, the corresponding metric is the sum of the weights of all edges of that node, i.e., the **strength** of that node.

For directed networks, we can separate between in-degree and out-degree centrality.

One problem with this definition of centrality is that it only captures the "local role" of a node in the network. A node might have many connections in an isolated cluster, or you could have a very important node that only has a few direct connections but connects two large groups that are otherwise disconnected. This centrality is also easy to manipulate.

Eigenvector Centrality A better centrality metric is to consider not only the number of neighbors of a node – but also the centrality of those neighbors. A node is more central when its neighbors are also more central.

Suppose that we are given an undirected network with adjacency matrix A. The centrality of node i is defined:

$$v_i = \frac{1}{\lambda} \sum_j A_{i,j} v_j$$

The $\frac{1}{\lambda}$ term can be thought of as a normalization constant for now. Note that the centrality of a node is the (normalized) sum of the centralities of all its neighbors.

This can also be written in matrix form as

$$\lambda v = Av$$

where v is the vector of all node centralities. Notice that this is the same as the definition of the eigenvector of A: λ is the corresponding eigenvalue for eigenvector v. Thus, we refer to this centrality metric as "eigenvector centrality".

We wish to have non-negative centralities. In other words, the eigenvector v corresponding to eigenvalue λ should consist of non-negative entries. It can be shown (using the Perron–Frobenius theorem) that using the largest eigenvalue of A satisfies this requirement.

The Katz Centrality The Katz centrality metric is a variation of eigenvector centrality that is more appropriate for directed networks.

The eigenvector centrality may be zero even for nodes with non-zero in-degree and out-degree, and so Katz starts from the same equation as eigenvector centrality but it also assigns a small centrality β to every node, so the definition of the Katz centrality of node i is:

$$v_i = \beta + \frac{1}{\lambda} \sum_j A_{i,j} v_j$$

where the summation is over all nodes j that connect with i.

Given that we are only interested in the relative magnitude of the centrality values, we can arbitrarily assign $\beta = 1$.

Rewriting the definition in matrix form:

$$v = (I - \frac{1}{\lambda})^{-1} \cdot \mathbf{1}$$

where $\mathbf{1}$ is an $n \times 1$ vector of all ones.

The value of λ controls the relative magnitude between the constant centrality β we assign to each node and the centrality that each node derives from its neighbors. If λ is very large, then the former term dominates and all nodes have roughly the same centrality. If λ is too small, on the other hand, the Katz centralities may diverge. This is the case when the determinant of the matrix $(I - \frac{1}{\lambda} A)$ is zero, which happens when λ is equal to an eigenvalue of A. To avoid this divergence, the value of λ is typically constrained to be larger than the maximum eigenvalue of A.

6.1.2 PageRank Centrality

PageRank is a famous centrality metric because it was the main algorithm that made Google the most successful search engine back in the late 1990s.

PageRank is only a slight modification of the Katz centrality metric. It is based on the following idea: if a node j points to a node i (thus $A_{i,j} = 1$), and node j has $k_{j,\text{out}}$ outgoing connections, then the centrality of node j should be "split" amoung those $k_{j,\text{out}}$ neighbors. In other words, the "wealth" of node j should not be just inherited by all nodes it points to, but rather, the "wealth" of node j should be split among those nodes.

So, the PageRank centrality becomes:

$$v_i = \beta + \frac{1}{\lambda} \sum_j A_{i,j} \frac{v_j}{k_{j,\text{out}}}$$

where the summation is over all nodes j that point to i (and thus, $k_{j,\text{out}}$ is non-zero).

In matrix form when $\beta = 1$:

$$v = (I - \frac{1}{\lambda} AD)^{-1} \cdot \mathbf{1}$$

where D is a diagonal $n \times n$ matrix in which the jth element is $\frac{1}{k_{j,\text{out}}}$ if $k_{j,\text{out}}$ is non-zero.

Undirected networks are typically transformed to directed networks by replacing each undirected edge with two directed edges.

In practice, the computation of both Katz and PageRank centralities is performed numerically, using a power-iteration method that iterates the computation of the centrality values until those values converge. Typical values for $\frac{1}{\lambda}$ and β are 0.85 and $(1 - \frac{1}{\lambda})/n$, respectively. It is theoretically possible that the Katz and PageRank centrality computations do not converge if λ is too low.

6.1.3 Closeness Centrality and Harmonic Centrality

The previous centrality metrics are all based on the direct connections of a node. Another group of centrality metrics focuses on network paths. Paths represent "routes" over which something is transferred through a network. Consequently, another way to think about the centrality of a node is based on how good the routes are of that node to the rest of the network.

Closeness Centrality A commonly used path-based metric is closeness centrality. It is based on the length (number of hops) of the shortest-path between a node i and a node j, denoted by $d_{i,j}$. The closeness centrality of node i is defined as the inverse of the average shortest path length $d_{i,j}$, across all nodes j that i connects to:

$$v_i = \frac{n-1}{\sum_j d_{i,j}}$$

where j is any node in the same connected component with i, and n is the number of nodes in that connected component (including i). If the network is directed, then we typically focus on shortest-paths from any node j to node i (i.e., incoming paths to i). The range of closeness centrality values is limited between 0 and 1.

The closeness centrality has some shortcomings, including the fact that it does not consider all network nodes – only nodes that are in the same connected component with i. So an isolated cluster of nodes can have high closeness centrality values (close to 1) even though those nodes are not even connected to most other nodes.

Harmonic Centrality An improved metric is harmonic centrality, defined as:

$$v_i = \sum_{j \neq i} \frac{1}{d_{i,j}}$$

Here, if nodes i and j cannot reach other, the corresponding distance can be thought of as infinite, and thus the term $\frac{1}{d_{i,j}}$ is 0.

Sometimes, the harmonic centrality is normalized by $\frac{1}{n-1}$, but that does not affect the relative ordering of node centralities.

6.1.4 Betweenness Centrality Variants

In some network analysis applications, the importance of a node is associated with how many paths go through a node: the more routes go through a node, the more central that node is.

Shortest-path Betweenness Centrality Consider any two nodes s (source) and t (target) in the same connected component, and let us define the number of shortest-paths between these two nodes as $n_{s,t}$. Also, suppose that the subset of these paths that traverses node i is $n_{s,t}(i)$ where i is different from s and t.

The shortest-path betweenness centrality of node i is:

$$v_i = \sum_{s,t \neq i} \frac{n_{s,t}(i)}{n_{s,t}}$$

If s and t are the same, then $n_{s,t} = 1$.

The previous metric is often normalized by its maximum possible value so that the centrality values are between 0 and 1. This is not necessary however given that we only care about the relative magnitude of centralities.

For weighted networks, the shortest-paths can be computed using Dijkstra's algorithm for weighted networks.

There are many variants on the betweenness centrality metric, depending on the kind of paths used.

- **Flow Betweenness Centrality**: compute the max-flow from any source node s to any target node t, and then replace the fraction $\frac{n_{s,t}(i)}{n_{s,t}}$ with the fraction of the max-flow that traverses node i.

- **Random-Walk Betweenness Centrality**: number of random walks from node s to node t that traverse node i.

Edge Centrality Metrics In some cases, we are interested in the centrality of edges, rather than nodes. One such metric is the **edge betweenness centrality**. The definition is the same as for node betweenness centrality – but instead of considering the fraction of paths that traverse a node i, we consider the fraction of that traverse an edge (i,j). These paths can be shortest-paths or any other well-defined set of paths, as we discussed on the previous page.

Another way to define the centrality of an edge is to quantify the impact of its removal. For instance, one could measure the increase in the Characteristic Path Length (CPL, see Lesson-5) after removing edge (i,j) – the higher that CPL increase is, the more important that edge is for the network.

6.1.5 Path Centrality For Directed Acyclic Graphs

In directed acyclic graphs (DAGs), we can consider all paths from the set of sources to the set of targets. Each source-target path (ST-path) represents one **"dependency chain"** through which that target depends on the corresponding source.

The path centrality of a node (including sources or targets) is defined as the total number of source-target paths that traverse that node.

It can be easily shown that the path centrality of node-i is the product of the number of paths from all sources to node-i, times the number of paths from node-i to all targets. The former term can be thought of as the **"complexity"** of node-i, while the second term can be thought of as the **"generality"** of node-i.

6.1.6 The Notion of "Node Importance"

We've defined a number of centrality metrics now – which metric should we use for each network analysis application? To choose the right metric, it is important to understand the notion of **"node importance"** that each of these centrality metrics focuses on.

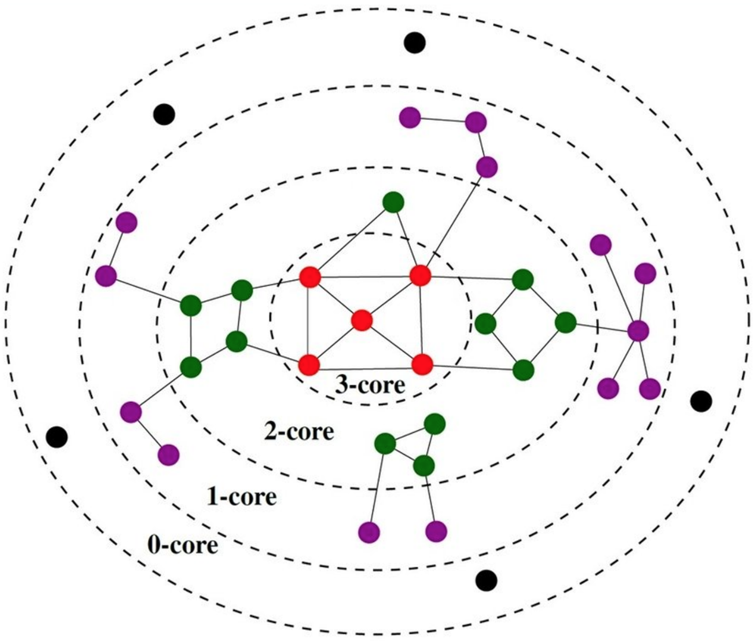
- **Degree (or strength) centrality**: appropriate when we are interested in the number or weight of direct connections of each node, eg finding the person with the most friends in a social network.

- **Eigenvector/Katz/PageRank centrality**: appropriate when we are mostly interested in the number of connections with other well-connected nodes. For undirected networks, it is better to use Eigenvector centrality because it does not depend on any parameters. For directed networks, use Katz or PageRank depending on whether it makes sense to split the centrality of a node among its outgoing connections.
- **Closeness (or harmonic) centrality**: appropriate when we are interested in how fast a node can reach every other node. If the network includes multiple connected components, it is better to use harmonic centrality.
- **Betweenness centrality**: appropriate in problems that involve some form of transfer through a network, and the importance of a node relates to whether that node is in the route of such transfers. You should use a betweenness centrality that captures correctly the type of routes used in that network. For instance, if the network uses shortest-path routes, it makes sense to use shortest-path betweenness centrality. If however, the network uses equally all possible routes between each pair of nodes, the path centrality may be a more appropriate metric.

6.1.7 k-core Decomposition

Sometimes instead of trying to rank individual nodes in terms of centrality, we are interested in identifying the most important group of nodes in the network. There are different ways to think about the importance of groups of nodes. One of them is based on the notion of **k-core**.

A k-core (or "core of order-k") is a maximal subset of nodes such that each node in that subset is connected to at least k others in that subset.



The visualization shows a network in which the red nodes belong to a core of order-3 (the highest order in this example), the green and red nodes form a core of order-2, while all nodes (except the black) form a core of order-1.

Note that there is a purple node with degree=5, but its highest order is 1 because all its connections except one are to nodes of degree-1. Similarly, there are green nodes of degree-3 that are in the 2-core set.

k-core decomposition is a simple algorithm which can associate each node with the highest k-core order that that node belongs to.

Algorithm:

1. Iteratively remove all nodes of degree k, for $k = 0, 1, 2....$
2. If, during the removal of nodes of degree k, some higher-degree nodes lose connections and become degree k, remove those nodes too and add them to the k-core.
3. Terminate when all nodes have been removed.

We can think about k-cores as successive layers of an onion, where $k = 1$ corresponds to the external layer of the onion, and the highest k corresponds to the "heart" of the onion.

The k-core decomposition process gradually peels off the network layer by layer until it reaches its most internal group of nodes.

The nodes in the maximum core order are considered as the most well-connected group of nodes in the network.

k-shell: the subgraph induced by nodes in the k-core that are not in the (k+1)-core. For example, the green nodes in the visualization form the 2-shell.

6.1.8 Core-Periphery Structure

Networks with a core-periphery structure can be partitioned in two groups:

- core nodes
- periphery nodes

Core nodes are very densely connected to each other and the rest of the network. Periphery nodes are well-connected only to core nodes.

6.1.9 Rich-Club Set of Nodes

A common approach for the detection of core-periphery structure is referred to as the **rich-club** of a network.

For undirected networks, suppose that the number of nodes with degree greater than k is n_k. Let e_k be the number of edges between those nodes. The maximum number of possible edges between those nodes is $\frac{n_k(n_k-1)}{2}$. The "rich-club coefficient" for degree k is defined as:

$$\phi(k) = \frac{2e_k}{n_k(n_k-1)}$$

and quantifies the density of connections between nodes of degree greater than k.

To determine whether the value of $\phi(k)$ is statistically significant for a given k, generate an ensemble of random networks with the same number of nodes and degree distribution (see lesson 4). These random networks represent our **null model**. Compute the average value of $\phi(k)$ across all random networks for each k.

If the rich-club coefficient for degree k in the given network is much larger than the corresponding coefficient in the null model, we can mark that value of k as statistically significant for the existence of a rich-club. If there is at least one such statistically significant value of k, we conclude that the network includes a rich-club – the set of nodes with degree greater than k. If there are multiple such values of k, the rich-club nodes can be identified based on the value of k for which we have the largest difference between the rich-club coefficient in the real network versus the null model (even though there is some variation about this in the literature).

6.1.10 Core Set of Nodes in DAGs

A path-based approach to identify a group of core nodes in a network is referred to as the "τ-core".

We define that a node v **"covers"** a path p when v is traversed by p.

The τ-core is defined: given a set of network paths P we are interested in, what is the minimal set of nodes that can cover at least a fraction τ of all paths in P?

In the context of DAGs, the set P can be the ST-paths from the set of all sources to all targets.

The rationale for covering only a fraction of paths (say 90%) instead of all paths is because in many real networks there are some ST-paths that do not traverse any intermediate nodes.

The problem of computing the the "τ-core" has been shown to be NP-Complete in the following paper: Sabrin, Kaeser M., and Constantine Dovrolis. "The hourglass effect in hierarchical dependency networks."

6.1.11 Applications

1. **GOT Network**: interactions of characters in Game of Thrones.

6.2 Reading Notes

6.2.1 subsection

7 Lesson 7: Modularity and Community Detection

Objectives:

- Understand the community detection problem

- Utilize the modularity metric and appreciate its limitations
- Analyze and deploy algorithms for community detection
- Understand the notion of hierarchical modularity

Reading:

- Chapter-9 from A-L. Barabási, Network Science.

7.1 Lecture Notes

7.1.1 The Graph Partitioning Problem

Graph Partitioning is a classical problem in computer science. Given a graph, how would we partition the nodes in two non-overlapping sets of the same size so that we minimize the number of edges between the two sets? This is also known as the **Minimum Bisection Problem**.

We can also state more general versions of this problem in which we partition the network into K non-overlapping sets of the same size, where $K > 2$ is a given constant.

The graph partitioning problem is NP-complete, so we only have approximate solutions. The **Kerninghan-Lin Algorithm** iteratively switches a node from one partition with a node from the other partition, selecting whichever pair causes the largest reduction in the number of edges that cut across the partition.

7.1.2 Network Community Detection

In the **Community Detection Problem**, we also need to partition the network into a set of non-overlapping clusters or communities, but this time we do not know a priori the number of sets, and they do not necessarily need to be the same size.

What are the key properties that define a community? Loosely speaking, the nodes should form a densely connected sub-graph. But this is not a mathematically precise definition, since we did not define how dense the sub-graph should be.

One extreme is to use a maximally sized clique, ie the largest subgraph that is still complete. This is a stringent definition, doesn't capture the fact that sometimes there might be missing edges between nodes in a community.

Instead, we could think of a community as an approximate clique: a subgraph in which the number of internal edges (edges between nodes of the subgraph) is much greater than the number of external edges (edges between a node of the subgraph and a node in the rest of the graph). This is still a loose definition.

Another way to think about network communities is through the adjacency matrix. Suppose a network has k non-overlapping groups of nodes of different sizes, such that the density of internal connections is much greater than the density of external connections. If we reorder the adjacency matrix so that the nodes of its group appear in consecutive rows, we will observe that the adjacency matrix contains much denser submatrices for each community.

Let's define communities better:

- Is every node required to belong to a community?
- Are communities necessarily non-overlapping? Can a node belong to more than one community?
- What if there are no real communities in the network? What if the network is randomly formed?

7.1.3 Community Detection Based on Edge Centrality Metrics

A family of algorithms for community detection is based on hierarchical clustering. The goal of such approaches is to create a tree structure, or "dendrogram", that shows how the nodes of the network can iteratively split in a top-down manner into smaller and smaller communities (**"divisive hierarchical clustering"**) - or how the nodes of the network can be iteratively merged in a bottom-up manner into larger and larger communities (**"agglomerative hierarchical clustering"**).

Starting with divisive algorithms: we first need a metric to select edges that fall between communities - the iterative removal of such edges will gradually result in smaller and smaller communities.

One such metric is the edge (shortest path) betweenness centrality, ie the fraction of shortest paths, across all pairs of terminal nodes, that traverses a given edge (see lesson 6). Removing the edge with the maximum centrality value will partition the network into two communities. We can then recompute the betweenness centrality for each remaining edge, and repeat the process to identify the next smaller communities. This algorithm is referred to as **"Girvan-Newman"** in the literature.

Another edge centrality metric that can be used for the same purpose is the random walk betweenness centrality. Here, instead of following shortest paths from a node u to a node v, we compute the probability that a random walk that starts from u and terminates at v traverses each edge e. The edge with the highest such centrality is removed first.

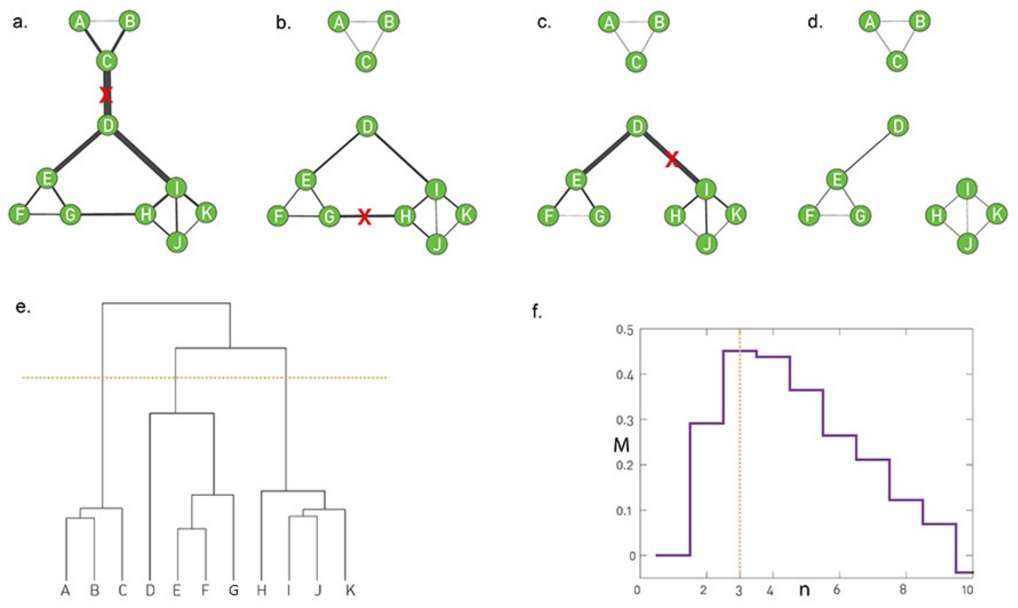
Note that the computational complexity of such algorithms depends on the algorithm we use for the computation of the centrality metric.

- For betweenness centrality, that computation can be performed in $O(LN)$, where L is the number of edges and N is the number of nodes.
- Given that we remove one edge each time, and need to recompute the centrality of the remaining edges in each iteration, the computational complexity of the Girvan-Newman algorithm is $O(L^2N)$. In sparse networks the number of edges is in the same order with the number of nodes, and so the Girvan-Newman algorithm runs in $O(N^3)$.

7.1.4 Divisive Hierarchical Community Detection

To illustrate the iterative top-down process followed by divisive hierarchical community detection algorithms, let us focus on the edge betweenness centrality metric. The Girvan-Newman algorithm uses that metric to remove a single edge in each iteration – the edge with the highest edge betweenness centrality.

The value of the edge betweenness centrality of the remaining edges has to be re-computed because the set of shortest paths changes in each iteration.



Note that a hierarchical clustering process does NOT tell us what is the best set of communities – each horizontal cut of the dendrogram corresponds to a different set of communities. So we clearly need an additional objective or criterion to select the best such set of communities. Such a metric, called modularity M, is shown in the visualization (f), which suggests we cut the dendrogram at the point of three communities (yellow line).

This algorithm always detect communities in a given network. So, even a random network can be split in this hierarchical manner, even though the resulting communities may not have any statistical significance.

7.1.5 Agglomerative Hierarchical Clustering Approaches - Node Similarity Metric

Agglomerative hierarchical clustering algorithms are bottom-up approaches. We start at the dendogram leaves, one for each node. In each iteration, we decide which nodes to merge and build the dendrogram up. Ideally, the merged nodes should belong to the same community, so we need a metric that quantifies the likelihood of two nodes belonging to the same community.

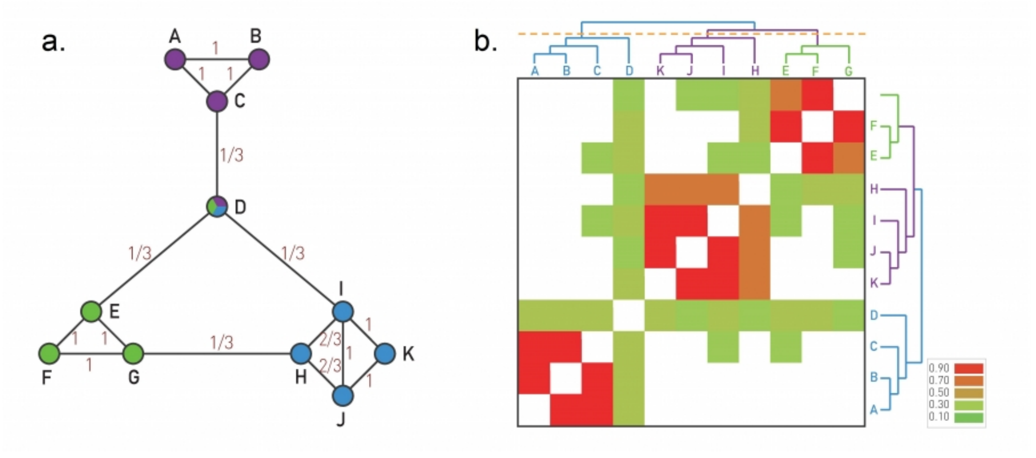
If two nodes, i and j, belong to the same community, we expect they will both be connected to several other nodes of the same community, so we expect them to have a large number of common neighbors, relative to their degree.

We define a similarity metric between any pair of nodes i and j:

$$S_{i,j} = \frac{N_{i,j} + A_{i,j}}{\min\{k_i, k_j\}}$$

where $N_{i,j}$ is the number of common neighbors of i and j, $A_{i,j}$ is the adjacency matrix element for the two nodes (1 if connected, 0 otherwise), and k_i is the degree of node i.

Note that $S_{i,j} = 1$ if the two nodes are connected with each other and every neighbor of the lower-degree node is also a neighbor of the other node. On the other hand, $S_{i,j} = 0$ when the two nodes are not connected to each other, and do not have any common neighbors.



The visualization on the left shows the node similarity value for each pair of connected nodes. The visualization on the rights shows the color-coded node similarity matrix, for every pair of nodes (even if they are not connected). Note that three groups of nodes emerge with higher similarity values: (A, B, C), (H,I,J,K) and (E,F,G). Node D on the other hand has a lower similarity with any other node, and it appears to be a "connector" between the three communities.

7.1.6 Hierarchical Clustering Approaches - Group Similarity

Three ways to compute the similarity between two groups of nodes:

- **Single Linkage:** the similarity of groups 1 and 2 is determined by the minimum distance (i.e., maximum similarity) across all pairs of nodes in groups 1 and 2.

- **Complete Linkage:** the similarity of groups 1 and 2 is determined by the maximum distance (i.e., minimum similarity) across all pairs of nodes in groups 1 and 2.
- **Average Linkage:** the similarity of groups 1 and 2 is determined by the average distance (i.e., average similarity) across all pairs of nodes in groups 1 and 2.

Average linkage is the most commonly used metric.

Where To Cut The Dendrogram Now we can design an iterative algorithm, known as the **Ravasz algorithm**, to compute a hierarchical clustering dendrogram in a bottom-up manner:

1. Start with each node represented as a leaf of the dendrogram. Compute the matrix of N^2 pairwise node similarities.
2. In each step:
   a) Select the two nodes, or two groups of nodes, with the highest similarity value.
   b) Merge them into a new group of nodes.
3. Continue until all nodes belong in the same group (the root of the dendrogram).

The computational complexity of the Ravasz algorithm is $O(N^2)$, since it requires:

1. $O(N^2)$ for the initial calculation of the pairwise node similarities
2. $O(N)$ for updating the similarity between the newly formed group of nodes and all other groups; this is performed each iteration, resulting in $O(N^2)$
3. $O(N \log N)$ for constructing the dendrogram

Note: depending on where we cut the dendrogram, we will end up with a different set of communities.

7.1.7 Modularity Metric

So far, our approaches for community detection cannot answer the following two questions:

1. Which partition of nodes into a set of communities is the best?
2. Are these communities statistically significant?

The **modularity metric** gives us a way to answer these questions. The idea is that randomly wired networks should not have communities, so a given community structure is statistically significant if the number of internal edges within the given communities is much larger than the number of internal edges if the network was randomly rewired, preserving the degree of each node.

In more detail:

- Given:
  - A network with N nodes and L edges. For now, we assume the network is undirected and unweighted.
  - The degree of node i is k_i.
  - A partitioning of all nodes to a set of C hypothetical communities, $C_1, C_2, ..., C_C$.
- Goal: evaluate this community structure – how good is it, and is it statistically significant?
- Let A be the network adjacency matrix.
- The number of internal edges between all nodes of community C_C is:

$$\frac{1}{2}\sum_{(i,j)\in C_C} A_{i,j}$$

- If the connections between nodes are randomly made, the expected number of internal edges between all nodes of that community is:

$$\frac{1}{2}\sum_{(i,j)\in C_i} \frac{k_i k_j}{2L}$$

  because node i has k_i stubs, and the probability that any of those stubs connect to a stub of node j is $\frac{k_j}{2L}$
- Define the modularity metric based on the difference between the actual number of internal edges and the expected number of internal edges, across all C communities:

$$M = \frac{1}{2L}\sum_{C=1}^{C}\sum_{(i,j)\in C_C}\left(A_{i,j} - \frac{k_i k_j}{2L}\right)$$

  . Note that we divide by the total number of edges, L, so M is always less than 1.

To determine whether a given community structure is statistically significant, we can compare the modularity of a network with 0, which is what we expect for a randomly wired network.

Alternatively, we can generate an ensemble of random networks using degree-preserving randomization and estimate the distribution of modularity values in that ensemble. Then use a one-sided hypothesis test to evaluate whether the modularity of the original network is larger than that distribution.

7.1.8 Modularity Metric - Derivation

Derivation of a more useful formula for the Modularity Metric Starting from the earlier definition of the modularity metric, we can derive a more useful expression for it:

$$M = \frac{1}{2L} \sum_{C=1}^{C} \sum_{(i,j) \in C_C} \left(A_{i,j} - \frac{k_i k_j}{2L} \right)$$

Rewrite the first term as

$$\sum_{C=1}^{C} \frac{L_C}{L}$$

where L_C is the number of internal edges in community C_C: $L_C = \frac{1}{2} \sum_{(i,j) \in C_C} A_{i,j}$.

The second term can be rewritten as:

$$\begin{aligned} \frac{1}{2L} \sum_{C=1}^{C} \sum_{(i,j) \in C_C} \frac{k_i k_j}{2L} &= \frac{1}{4L^2} \sum_{C=1}^{C} \sum_{(i,j) \in C_C} k_i k_j \\ &= \frac{1}{4L^2} \sum_{C=1}^{C} \left(\sum_{i \in C_C} k_i \right) \left(\sum_{j \in C_C} k_j \right) \\ &= \sum_{C=1}^{C} \left(\frac{\kappa_C}{2L} \right)^2 \end{aligned}$$

where κ_C is the total number of stubs at the nodes of community C, ie: $\kappa_C = \sum_{i \in C_C} k_i$.

Substituting these two terms back to the original definition, we get an equivalent definition of the modularity metric:

$$M = \sum_{C=1}^{C} \left(\frac{L_C}{L} - \left(\frac{\kappa_C}{2L} \right)^2 \right)$$

The modularity of a given community assignment is expressed as the summation, across all communities, of the fraction of network edges that are internal to community C_C MINUS the squared fraction of total edge stubs that belong to that community.

7.1.9 Modularity Metric: Selecting the Community Assignment And Directed and/or Weighted Networks

Selecting the Community Assignment Now that we have a way to compare different community assignments, we can return to the hierarchical clustering approaches we saw earlier (both divisive and agglomerative) and add an extra step, after the construction of the dendrogram:

Each branching point of the dendrogram corresponds to a different community assignment, i.e., a different partition of the nodes in a set of communities. So, we can calculate the modularity at each branching point, and select the branching point that leads to the highest modularity value.

Directed and/or Weighted Networks The modularity definition can be easily modified for directed and/or weighted networks.

- **Directed and Unweighted:** Suppose the out-degree of node i is $k_{i,out}$, the in-degree of node j is $k_{j,in}$, and $A_{i,j} = 1$ when there is an edge from node i to node j and $A_{i,j} = 0$ otherwise. The modularity definition can be rewritten as:

$$M = \frac{1}{L} \sum_{C=1}^{C} \sum_{(i,j) \in C_C} \left(A_{i,j} - \frac{k_{i,out} k_{j,in}}{L} \right)$$

- **Directed and Weighted:** Suppose the "out-strength" (sum of all outgoing edge weights) of node i is $s_{i,out}$, the "in-strength" of node j is $s_{j,in}$, and $A_{i,j}$ is the weight of the edge from node i to node j (and 0 when there is no such edge). Then the modularity metric is:

$$M = \frac{1}{S} \sum_{C=1}^{C} \sum_{(i,j) \in C_C} \left(A_{i,j} - \frac{s_{i,out} s_{j,in}}{S} \right)$$

  where S is the sum of all edge weights.

7.1.10 Modularity After Merging Two Communities

The modularity of a network increases when we merge two communities into a new community.

Derivation Suppose that community A has a number of internal links L_A and a total degree κ_A, and similarly for community B.

If we merge A and B into a single community called AB while all other communities remain the same:

- Number of internal links in AB: $L_{AB} = L_A + L_B + l_{AB}$ where l_{AB} is the number of external links between communities A and B.
- Total degree of community AB: $\kappa_{AB} = \kappa_A + \kappa_B$
- Thus, the difference in modularity after merging A and B:

$$\Delta M_{AB} = \left(\frac{L_{AB}}{L} - \left(\frac{\kappa_{AB}}{2L}\right)^2\right) - \left(\frac{L_A}{L} - \left(\frac{\kappa_A}{2L}\right)^2 + \frac{L_B}{L} - \left(\frac{\kappa_B}{2L}\right)^2\right)$$

  This can be simplified:

$$\Delta M_{AB} = \frac{l_{AB}}{L} - \frac{\kappa_A \kappa_B}{2L^2}$$

The merging step results in higher modularity only when the number of links between A and B is sufficiently large for the first term to be larger than the second term. The larger the communities A and B are, the larger the second term is. Otherwise, this merging operation causes a reduction in the modularity of the new community assignment.

7.1.11 Greedy Modularity Maximization

Maximizing modularity is an NP hard problem. It cannot be solved efficiently unless P=NP.

One of the most commonly used algorithms is this greedy heuristic:

1. Assign each node to a community, starting with N communities of single nodes.
2. Inspect each community pair connected by at least one link, and compute the modularity difference ΔM obtained if we merge those two communities. Merge the pair with the largest ΔM.
3. Repeat the above step until all nodes merge into a single community, recording the value of M at each iteration.
4. Select the iteration/partition for which M is maximal.

Computational Complexity The calculation of each ΔM can be done in constant time, so the second step requires $O(L)$ computations. After deciding which communities to merge, the update of the matrix can be done in worst-case time $O(N)$. Since the algorithm requires $N-1$ community mergers, its complexity is $O[(L+N)N]$, or $O(N^2)$ for a sparse graph.

Optimized Greedy Algorithm Using data structures for sparse matrices can decrease the complexity to $O(N \log^2 N)$: see paper Finding community structure in very large networks by Clauset, Newman and Moore.

7.1.12 Louvain Algorithm

The Louvain algorithm is a more computationally efficient modularity maximization algorithm than the previous greedy algorithm.

Even if the original network is unweighted, the Louvain algorithm creates a weighted network, so all modularity calculations use the formula for weighted networks.

The Louvain consists of multiple passes of the following two steps:

Step 1 Start with a network of N nodes, initially assigning each node to a different community. For each node i, evaluate the modularity change if we place node i in the community of any of its neighbors j.

Move node i in the neighboring for which the modularity difference is the largest, but onoly if this difference is positive (if we cannot increase the modularity by moving i, then it stays in its original community).

The modularity change by merging a single node i with the community A of a neighboring node is:

$$\Delta M_{i,A} = \left(\frac{W_{A,int} + 2W_{i,A}}{2W} - \left(\frac{W_A + W_i}{2W} \right)^2 \right) - \left(\frac{W_{A,int}}{2W} - \left(\frac{W_A}{2W} \right)^2 - \left(\frac{W_i}{2W} \right)^2 \right)$$

where W_A is the sum of weights of all links in A, $W_{A,int}$ is the sum of weights of all internal links in A, W_i is the sum of weights of all links of node i, $W_{i,A}$ is the sum of weights of all links between node i and any node in A, and W is the sum of weights of all links in the network. This is similar to the formula derived earlier for merging two communities, but adjusted for weighted networks.

This process is applied to every node until no further improvement can be achieved.

Step 2 Now we construct a new weighted network, whose nodes are the communities identified in Step 1. The weight of the link between two nodes in this network is the sum of weights of all links between the nodes in the corresponding two communities. Links between nodes of the same community lead to weighted self-loops.

Steps 1 and 2 can be repeated in successive passes. The number of communities decreases with each pass. The passes are repeated until there are no more changes and maximum modularity is attained.

Computational Complexity This algorithm is $O(L)$.

7.1.13 Modularity Resolution

Does the modularity metric every fail in choosing a community assignment? Yes - the modularity metric cannot be used to discover very small communities, relative to the size of the given network.

The modularity maximization algorithm will always merge two communities if the total degree of each of those communities is smaller than the critical value $\sqrt{2L}$. This critical value is referred to as the **modularity resolution** because it is the smaller community size (in terms of total degree) that a modularity maximization algorithm can detect.

Derivation: Suppose a network consists of multiple communities, with only a single link between any two communities. A good community detection algorithm should not merge any of these communities.

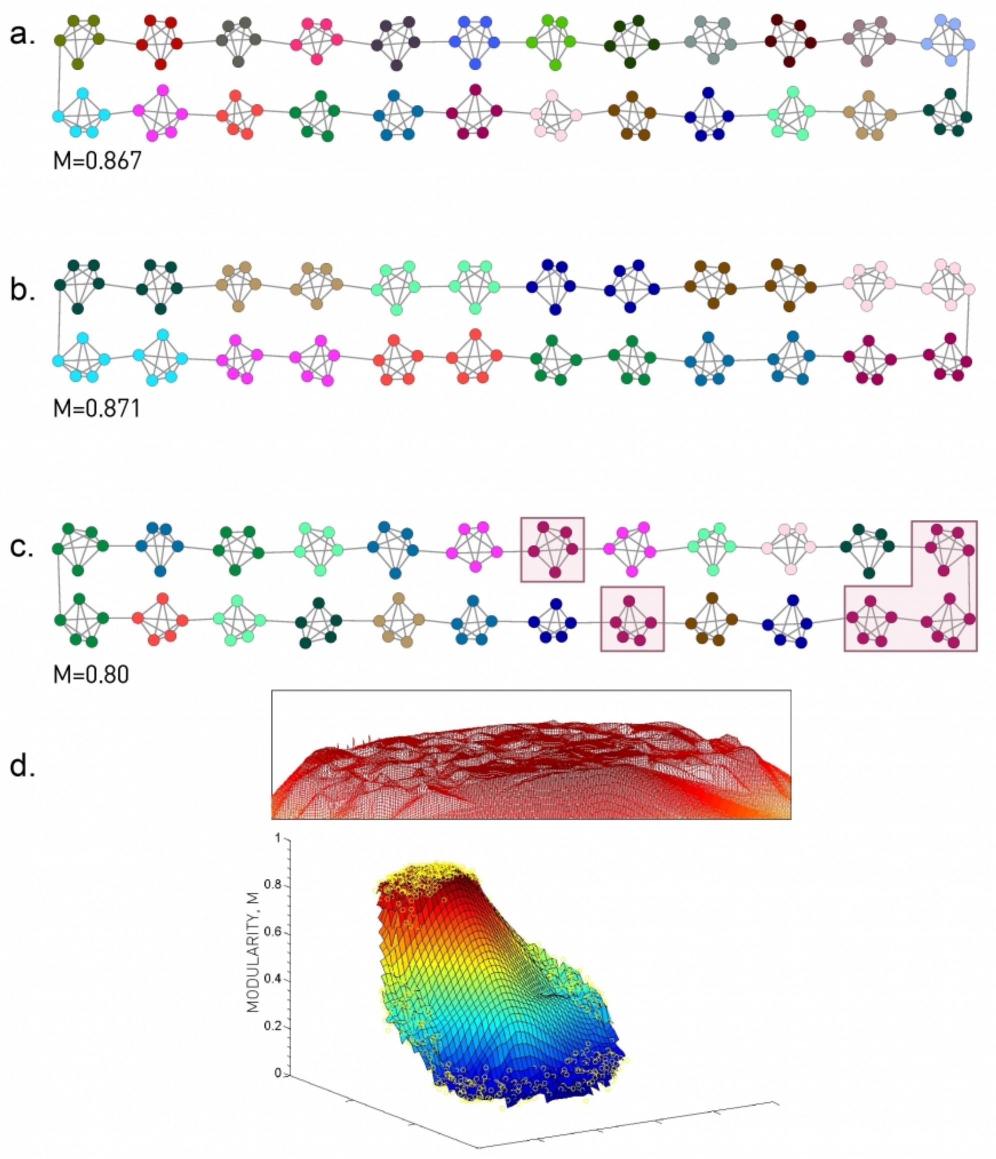
Suppose that the total degree of each community is κ_l. Then if we merge any two connected communities, the modularity difference will be:

$$\begin{aligned}\Delta M_{AB} &= \frac{l_{AB}}{L} - \frac{\kappa_A \kappa_B}{2L^2} \\ &= \frac{1}{L}\left(1 - \frac{\kappa_l^2}{2L}\right)\end{aligned}$$

Two small communities will be merged if $\Delta M_{AB} > 0$, or $\kappa_l < \sqrt{2L}$.

7.1.14 The Modularity Search Landscape

It is common that the most intuitive community assignment (Fig (a)) is not the maximal modularity value (Fig (b)), but often the maximum resides in a plateau (Fig (d)) and is not much higher than other local maxima. Thus, even though we rarely know the optimal solution, the heuristics can usually compute reasonable solutions that reside in this modularity plateau.



Network Science Book by Barabasi, Figure 9.18

7.1.15 Communities Within Communities

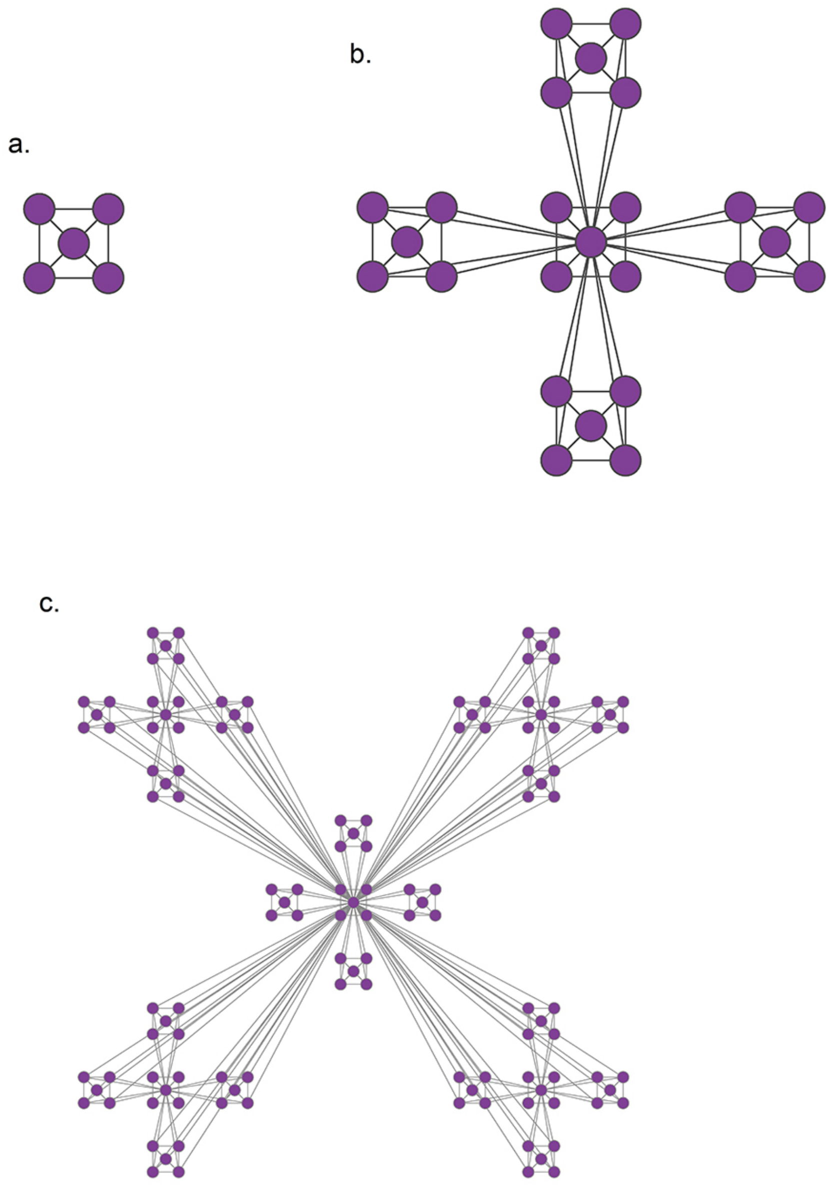
In many real-world networks, we observe communities within communities within communities, etc.

This is referred to as **Hierarchical Modularity**. Each module corresponds to a community of nodes. Smaller, more tightly connected communities can be members of larger but less tightly connected larger communities. This recursive process can continue for multiple levels until we reach a level in which the group of nodes is so loosely connected that they do not form a community anymore.

A prototypical example: the most elementary community is a five-node clique module. In Fig b, we see a similar structure composed of five of these cliques. Note that the network at this level is NOT a clique, so it is less tightly connected than the five smaller communities it's composed of.

Figure c connects five instances of the Part b module in a similar manner.

For this toy network, the clustering coefficient of a node in this network drops inversely proportionally to the node's degree: $C(k) \approx \frac{2}{k}$. This is an interesting "signature" for this hierarchically modular network: the more connected a node is, the less interconnected its neighbors are.



Network Science Book by Barabasi, Figure 9.13

7.1.16 Clustering Coefficient Vs. Degree In Practice

Hierarchical modularity is an important property of many (but not all) real-world networks, and when present, it exhibits itself with an inversely proportional between the clustering coefficient and the node degree. However, this relation is not sufficient evidence for the presence of hierarchical modularity.

7.1.17 Hierarchical Modularity Through Recursive Community Detection

Another approach to investigate the presence of hierarchical modularity is by applying a community detection algorithm in a recursive manner. The first set of discovered communities correspond to the top-level, larger communities. Then we can apply the same algorithm separately to each of those communities, asking whether there is a finer resolution community structure within each of those top-level communities. The process can continue until we reach a level at which a given community does not have a clear sub-community structure.

7.2 Reading Notes

7.2.1 Chapter 9, A-L Barabasi: Communities

Basics of Communities The **fundamental hypothesis** of this chapter is: A network's community structure is uniquely encoded in its wiring diagram. Ie, there is a ground truth about a network's community organization that can be uncovered by inspecting A_{ij}.

Our sense of communities also depends on a **second hypothesis**: A community is a locally dense connected subgraph in a network. Ie, all members of a community must be reachable through other members of the same community (connectedness), and we expect nodes that belong to a community to have a higher probability to link to other nodes of the community than to nodes outside the community (density).

There are multiple ways to define communities that would be consistent with the above hypothesis:

- **Maximum Cliques**: We can define a community as a group of individuals whose members all know each other. In graph theoretic terms, a community is a complete subgraph, or a clique. Drawbacks: larger cliques are rare in networks, and requiring that a community be a complete subgraph may be too restrictive, missing legitimate communities.
- **Strong Community** Consider a subgraph C, with N_C nodes. The *internal degree*, k_i^{int}, of node i is the number of links that connect i to other nodes in C. The *external degree*, k_i^{ext}, is the number of links that connect i to the rest of the network. C is

a strong community if each node within C has more links within the community than with the rest of the graph, ie, for each node $i \in C$, $k_i^{int}(C) > k_i^{ext}(C)$.

- **Weak Community** C is a weak community if the total internal degree of a subgraph exceeds its total external degree, ie $\sum_{i \in C} k_i^{int}(C) > \sum_{i \in C} k_i^{ext}(C)$. This relaxes the strong community requirement by allowing some nodes to violate the requirement that the internal degree exceeds the external degree, as long as the community as a whole has more internal than external connections.

8 Lesson 8: Advanced Topics in Community Detection

Objectives:

- Identify overlapping communities
- Generate synthetic networks with known community structure
- Evaluate and compare community structures
- Explain the role of nodes depending on their position between/within communities
- Identify dynamic/evolving communities

Reading:

- Chapter-9 from A-L. Barabási, Network Science.
- (Recommended) Dynamic Community Detection, by Rémy Cazabet, Frédéric Amblard, 05 October 2014.
- δ-MAPS: from spatio-temporal data to a weighted and lagged network between functional domains by Fountalis, I., Dovrolis, C., Bracco, A. et al., 2018

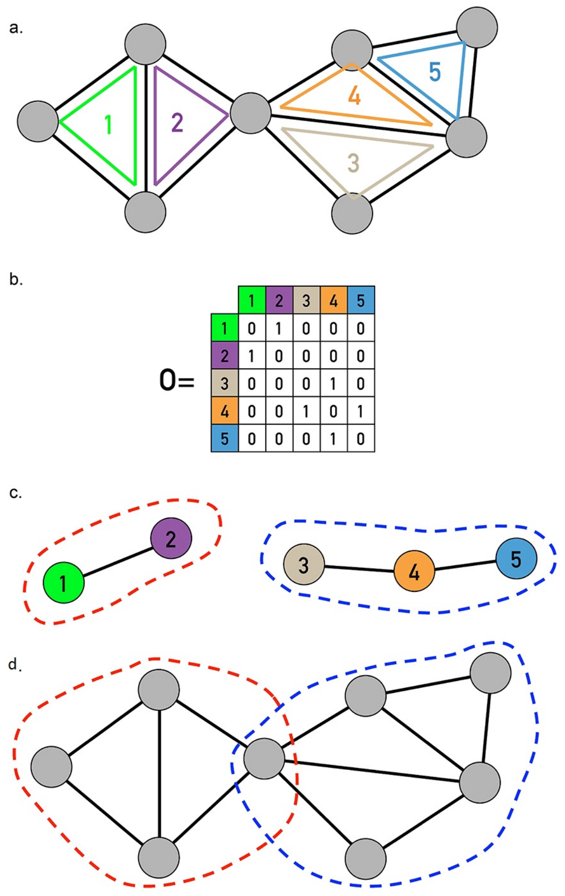
8.1 Lecture Notes

8.1.1 Overlapping Communities and CFinder Algorithm

In practice, a node can often belong to more than one community.

The CFinder Algorithm was one of the first community detection methods that can discover overlapping communities. Its worst-case runtime is exponential with the network size, but it can be effective in networks with a few thousand nodes.

Instead of being based on modularity maximization, it is based on the idea that the community is a dense subgraph of nodes that contain many overlapping small cliques. Its main parameter is size k of these small cliques.



$$O=\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 1 & 0 \\ 4 & 0 & 0 & 1 & 0 & 1 \\ 5 & 0 & 0 & 0 & 1 & 0 \end{array}$$

Network Science Book by Barabasi, Figure 9.39

For example, if $k = 3$, the CFinder algorithm finds all triangles in the network (see Figure part a). Two k-cliques are overlapping if they share $k-1$ nodes (in the case of $k = 3$, triangles must share an edge for them to be overlapping).

Create an overlap matrix, O, shows which k-cliques overlap, where element $O_{i,j} = 1$ if k-clique i overlaps with k-clique j, and $O_{i,j} = 0$ if not (Figure part b).

The communities that the CFinder algorithm discovers are the connected components of the k-clique network (Figure part c). When projected back to the original network, nodes can belong to multiple communities if it belongs to k-cliques that are separated into different connected components (Figure part d).

8.1.2 Critical Density Threshold in CFinder

Would the CFinder detect communities even in completely random networks? A completely random network should not have any community structure. With k=3, if the network is sufficiently dense, it will have many triangles strictly by chance, so the algorithm would detect a number of communities even though the network structure is completely random.

So, for a given network size n and value of k, what is the maximum density of a random network of size n so that the appearance of k-cliques is unlikely? This density is called the **"critical density threshold"**.

The higher k is, the higher the critical density threshold should be.

The critical density threshold is:

$$p_c(k) = [(k-1)n]^{\frac{-1}{k-1}}$$

When the density is lower than $p_c(k)$ we expect only few isolated k-cliques. If the network density exceeds $p_c(k)$, we observe numerous cliques that form k-clique communities.

For k=2 the k-cliques are simply individual links and the critical density threshold reduces to $p_c(k) = \frac{1}{n}$, which is the condition for the emergence of a giant connected component in Erdos-Renyi networks.

For k =3 the k-cliques become triangles and the critical density threshold is $p_c(k) = \frac{1}{\sqrt{2n}}$.

The moral of the story is that before we apply the Cfinder algorithm, we should also check whether the density of the network is below or above the critical threshold for the selected value of k. If it is larger than the threshold, we should consider larger values of k (even though this would make the algorithm more computationally intensive).

8.1.3 Link Clustering Algorithm

Another approach to compute overlapping communities is the **link clustering algorithm**. It is based on the idea that even though a *node* may belong to multiple communities, each *link* is usually part of only one community.

So, instead of trying to compute communities by clustering nodes, we can try to compute communities by clustering links. However, how can we cluster links? We first need a similarity metric to quantify the likelihood that two links belong to the same community.

Recall that if two nodes i and j belong to the same community, we expect that they will have several common neighbors (because a community is a dense subgraph). So the similarity $S((i,k),(j,k))$ of two links (i,k) and (j,k) that attach to a common node k can be evaluated based on the number of common neighbors that nodes i and j have – with the convention that the set of neighbors of a node i includes the node itself. We normalize this number by the total number of distinct neighbors of i and j, so that it is at most equal to 1 (when the two nodes i and j have exactly the same set of neighbors).

In other words, if N(i) is the set of neighbors of node i (including i) and N(j) is the set of neighbors of node j, then the link similarity metric is defined as:

$$S((i,k),(j,k)) = \frac{N(i) \cap N(j)}{N(i) \cup N(j)}$$

Now that we have a similarity metric for pairs of links, we can follow exactly the same hierarchical clustering process we studied in the previous lesson to create a dendrogram of links, based on their pairwise similarity.

The computational complexity of the link clustering algorithm depends on two operations:

1. the similarity computation for every pair of links, and
2. the hierarchical clustering computation

. The first depends on the maximum degree in the network, which we had derived earlier in lesson 4 as $O(n^{\frac{2}{\alpha-1}})$ for power-law networks with n nodes and degree exponent α. The latter is $O(m^2)$, where m is the number of edges. For sparse networks, this last term is $O(n^2)$ and it dominates the former term.

8.1.4 Application of Link Clustering Algorithm

Les Mis pairwise interactions

8.1.5 GN Benchmark

To test whether a community detection algorithm actually identifies the right set of communities, we can create synthetically generated networks with known community structure, and check whether the algorithm can discover that structure. These synthetic networks are referred to as **"benchmarks"**.

The simplest benchmark is the **"Girvan-Newman (GN)"**. In this model:

- all communities have the same size
- any pair of nodes within the same community has the same connection probability, p_{int}
- any pair of nodes in two different communities also has the same connection probability, p_{ext}

Consequently, the GN benchmark cannot generate a heavy-tailed degree distribution.

To generate a GN network, we specify:

- the number of communities n_c
- the number of nodes in each community N_c
- the parameter μ:

  $$\mu = \frac{k^{ext}}{k^{ext} + k^{int}}$$

  where k^{int} is the average number of internal connections in a communinty and k^{ext} is the average number of external connections between communities.

If μ is close to 0, almost all connections are internal and the community is very clearly separated. As μ increases, the community structure becomes weaker. When $\mu > 0.5$ it is more likely to see external connections than internal, ie no community structure.

8.1.6 Community Size Distribution

The GN benchmark creates communities of identical size.

Even though we typically do not have the ground-truth for real-world networks, there is increasing evidence that most networks have a power-law community size distribution. This implies that most communities are quite small, maybe a tiny percentage of all nodes, but there are also few large communities with comparable size to the whole network.

8.1.7 LFR Benchmark

The **LFR Benchmark (Lancichinetti-Fortunato-Radicchi)** is more realistic, because it generates networks with scale-free degree distribution, and additionally, the community size follows a power-law size distribution.

We need to specify:

- the total number of nodes n
- the parameter μ (same as in the GN benchmark)
- an exponent parameter γ for the node degree distribution
- an exponent parameter ζ for the community size distribution

LFR starts with n disconnected nodes, giving each of them a degree (number of stubs) based on the degree distribution.

Each node is assigned to a community based on the community size distribution.

The stubs of each node are then classified as either internal, to be connected to other internal stubs of nodes in the same community - or external, to be connected to available external stubs of nodes in other communities. Thisi stub partitioning process is controlled by the parameter μ.

The connections between nodes are then assigned randomly as long as a node has available stubs of the appropriate type (internal vs external).

8.1.8 Normalized Mutual Information (NMI) Metric

Suppose we have two different community partitionings of the same network, P_1 and P_2. In order to compare these two partitions statistically, we need the joint distribution $p(C_i^1, C_j^2)$ of the two partitions, i.e., the probability that a node belongs to the ith community of the first partition, C_i^1, and the jth community of the second partition, C_j^2.

The marginal distribution $p(C_i^1)$ of the first partition represents the probability that a node belongs to community C_i^1 of the first partition. Similarly for the second partition.

The **"confusion matrix"** can be used to calclate the joint distribution as well as the two marginal distributions.

A common way to compare two statistical distributions in Information Theory is through the Mutual Information – this metric quantifies how much information we get for one of two random variables if we know the value of the other. If the two random variables are statistically independent, we do not get any information. If the two random variables are identical, we get the maximum information.

When the two random variables represent the two community partitions of the same network, the definition of Mutual Information is written as:

$$\sum_{i,j} p(C_i^1, C_j^2) \log_2 \frac{p(C_i^1, C_j^2)}{p(C_i^1)p(C_j^2)}$$

To get a metric that varies between 0 (statistically independent partitions) and 1 (identical partitions), we need to normalize the mutual information with the average entropy of the two partitions, $\frac{1}{2}[H(p(C^1)) + H(p(C^2))]$. Recall that H(p(X)) is the entropy of the distribution p(X) of random variable X, defined as:

$$H(X) = -\sum_x p(x) \log_2 p(x)$$

So, the Normalized Mutual Information metric (NMI) is defined as:

$$\text{NMI} = I_n = \frac{\sum_{i,j} p(C_i^1, C_j^2) \log_2 \frac{p(C_i^1, C_j^2)}{p(C_i^1)p(C_j^2)}}{\frac{1}{2}[H(p(C^1)) + H(p(C^2))]}$$

8.1.9 Accuracy Comparison of Community Detection Algorithms

8.1.10 Run Time Comparisons

8.1.11 Dynamic Communities

In practice, many networks change over time through the creation or removal of nodes and edges. There are various ways in which the community structure may change from time t to time t+1:

- **Growth**: a community can grow through new nodes.
- **Contraction**: a community can contract through node removals.
- **Merging**: two or more communities can merge into a single community.
- **Splitting**: one community can split into two or more communities.
- **Birth**: a new community can appear at a given time.
- **Death**: a community can disappear at any time.
- **Resurgence**: a community can only disappear for a period and come back later.

Detecting communities in dynamic networks is a more recent problem, and there is no single method that is considered generally accepted. There are three general approaches.

8.1.12 Dynamic Communities: Approach 1

The first approach is that a community detection algorithm is applied independently on each network snapshot, without any constraints imposed by earlier or later network snapshots.

Then, the communities identified at time T+1 are matched to the communities identified at time T based on maximum node and edge similarity.

The main advantage of this approach is its simplicity, given that we can apply any algorithm for community detection on static networks.

Its main disadvantage however is that the communities of successive snapshots can be significantly different, not because there is a genuine change in the community structure, but because the static community detection algorithm may be unstable, producing very different results even with minor changes in the topology of the input graph.

8.1.13 Dynamic Communities: Approach 2

The second approach attempts to compute a good community structure at time T+1 also considering the community structure computed earlier at time T.

To do so, we need to modify the objective function of the community detection optimization problem: instead of trying to maximize modularity at time T+1, we aim to meet two goals simultaneously:

- have both high modularity at time T+1, and

- maintain the community structure of time T as much as possible.

This is a trade-off between the quality of the discovered communities (quantified with the modularity metric) and the "smoothness" of the community evolution over time.

Such dual objectives require additional optimization parameters and trade-offs, raising concerns about the robustness of the results.

8.1.14 Dynamic Communities: Approach 3

A third approach is to consider all network snapshots simultaneously. Or, a more pragmatic approach would be to consider an entire "window" of network snapshots $(T, T+1, T+2, \ldots T+W-1)$ at the same time.

Algorithms of this type create a single "global network" based on all these snapshots as follows:

1. A node of even a single snapshot is also a node of the global network, and an edge of even a single snapshot is also an edge between the corresponding two nodes at the global network.
2. Additionally, the global network includes edges between the instances of any node X at different snapshots (so if X appears at time T and at time T+1, there will be an edge between the corresponding two node instances in the global network).
3. After creating the global network, a community detection algorithm is applied on that network to compute a "reference" community structure.
4. Then, that reference community structure is mapped back to each snapshot, based on the nodes and edges that are present at that snapshot.

This approach usually produces the smoothest (or most stable) results – but it can miss major steps in the evolution of the community structure (such as communities that merge or split). This last point however also depends on the length of the window W.

8.1.15 Participation Coefficient

An interesting question is to examine the role of individual nodes within the community they belong to. Is a node mostly connected to other nodes of the same community? Is it connected to several other communities, acting as a bridge?

A commonly used metric to answer such questions is the **Participation Coefficient** of a node, measuring the participation of that node in *other* communities.

Suppose that we have already discovered that the network has n_c communities (or modules). Also, let $k_{i,s}$ be the number of links of node i that are connected to nodes of community s, and k_i the degree of node i. The participation coefficient is then defined as:

$$P_i = 1 - \sum_{s=1}^{n_c} \left(\frac{k_{i,s}}{k_i} \right)^2$$

Note that if all the connections of node i are with other nodes in its own community, we have that $P_i = 0$.

On the other hand, if the connections of node i are uniformly distributed with all n_c communities, the participation coefficient will be close to 1.

The (loose) upper bound $P_i = 1$ is approached for nodes with very large degree k_i, if their connections are uniformly spread to all n_c communities.

8.1.16 Within-Module Degree

Another important question is whether a node is a hub or not within its own community.

A metric to answer this question is the normalized **"within-module degree"**:

$$z_i = \frac{\kappa_i - \bar{\kappa}_i}{\sigma_{\kappa_i}}$$

where κ_i is the "internal" degree of node i, considering only connections between that node and other nodes in the same community, $\bar{\kappa}_i$ is the average internal degree of all nodes that belong in the community of node i, and σ_{κ_i} is the standard deviation of the internal degree of those nodes.

So, if a node i has a large internal degree, relative to the degree of other nodes in its own community, we consider it a hub within its module.

Combining the information provided by both the participation coefficient and the within-community degree, we can identify four types of nodes:

1. **Connector Hubs:** nodes with high values of both P_i and z_i. These are hubs within their own community that are also highly connected to other communities, forming bridges to those communities.
2. **Provincial Hubs:** nodes with high value of z_i but low value of P_i. These are hubs within their own community but not well connected to nodes of other communities.

3. **Connectors:** nodes with high value of P_i but low value of z_i. These are not hubs within their module – but they form bridges to other communities.

4. **Peripheral nodes:** nodes with low values of both P_i and z_i. Everything else – this is how the bulk of the nodes will be classified as.

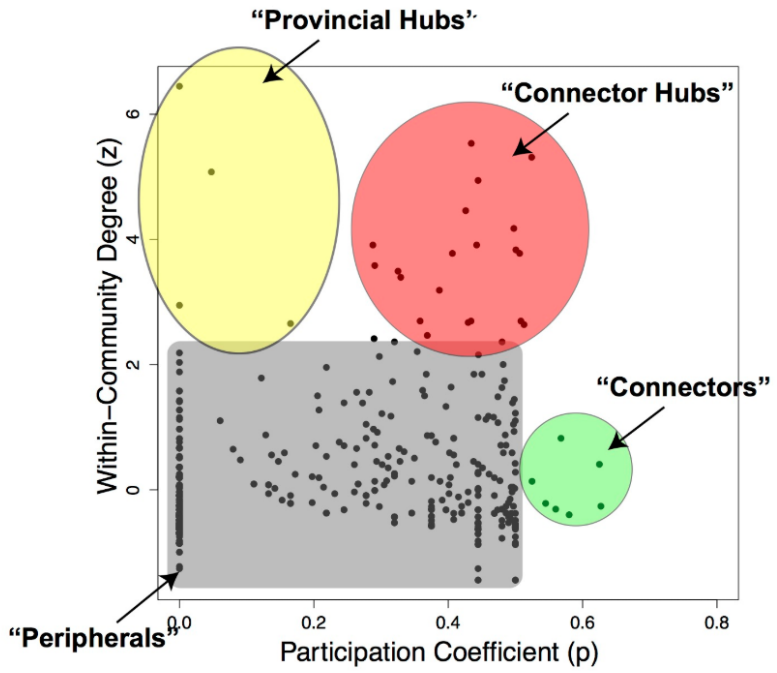


Figure from Worked Example: Centrality and Community Structure of US Air Transportation Network by Dai Shizuka

8.1.17 Case Study: Metabolic Networks

8.1.18 δ-MAPS

δ**-MAPS** ("Delta Maps") is a method to detect communities in the analysis of spatiotemporal data.

δ-MAPS are appropriate for any application in which we are given spatiotemporal data that show an activity time-series of each point of a grid space. The goal of δ-MAPS is to detect contiguous blocks of space, aka functional domains, that behave in a homogeneous manner over time.

Common in climate science, where the activity time series might be temperature, humidity, etc. at different points on the planet or the atmosphere.

8.1.19 δ-MAPS: Domain Homogeneity Constraint

8.1.20 δ-MAPS: Domain Homogeneity Constraint

8.1.21 δ-MAPS: Domain Homogeneity Constraint

8.1.22 δ-MAPS: Domain Homogeneity Constraint

8.1.23 δ-MAPS: Domain Homogeneity Constraint

8.1.24

8.2 Reading Notes

8.2.1 subsection

9 Lesson x: lesson theme

Objectives:

-
-
-

Reading:

- Chapter-2 from A-L. Barabási, Network Science.
- (Recommended) Fibonacci Heap. Wikipedia. `https://en.wikipedia.org/wiki/Fibonacci_heap`

9.1 Lecture Notes

9.1.1 subsection

9.2 Reading Notes

9.2.1 subsection