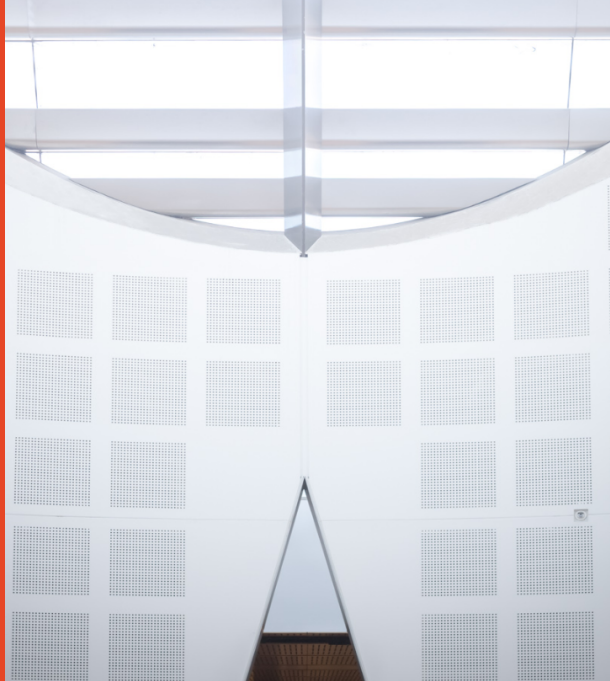
COMP5313/COMP4313 - Large Scale Networks

Week 4: Community Detection

Lijun Chang

March 20, 2025

THE UNIVERSITY OF SYDNEY



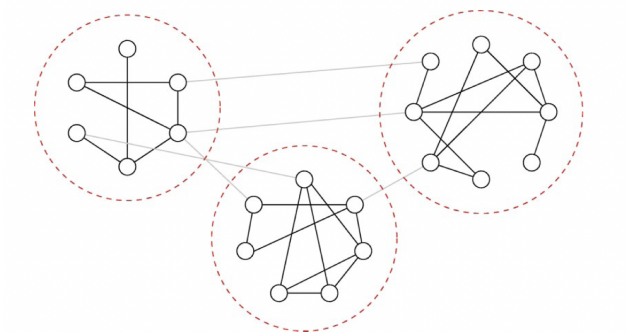
Tightly-knit Groups

- Week 2's lecture on strong/weak ties suggests that social networks usually contain tightly-knit groups that are connected by weak-ties
- Also, Week 3's lecture on homophily suggests that a social network usually can be divided into densely connected homogeneous parts that are weakly connected to each other
- Today, we will study how to identify the tightly-knit groups by looking at the structure of the network only
  - Tightly-knit groups are usually regarded as communities

Outline

Community Structure

- Community structure: a cohesive group of nodes that are connected "more densely" to each other, than to nodes outside this community



An example network with three communities

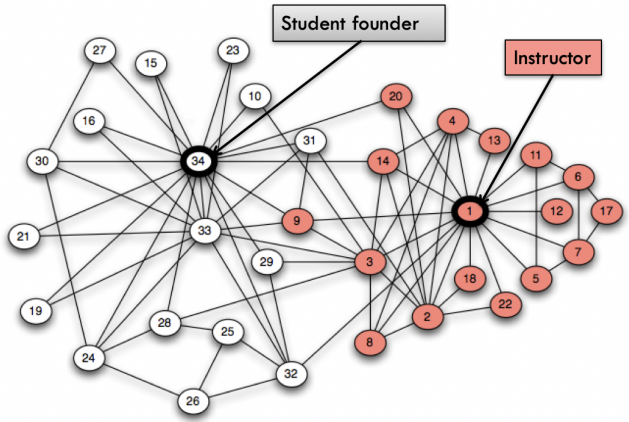
Community Structure

- Social networks are full of easy-to-spot communities
  - The employees of a company are more likely to interact with their coworkers than with employees of other companies
  - Communities could also represent
    - circles of friends
    - a group of individuals who pursue the same hobby together
    - individuals living in the same neighborhood
- Communities play an important role in understanding diseases
  - Proteins that are involved in the same disease tend to interact with each other [1]

[1] J. Menche, A. Sharma, M. Kitsak, S. Ghiassian, M. Vidal, J. Loscalzo, A. Barabasi. Uncovering disease-disease relationships through the incomplete human interactome. 2015

Community Structure

- Two rival karate clubs represented by different colors
  - A conflict between the club's president and the instructor split the club in two.



Community Detection/Finding

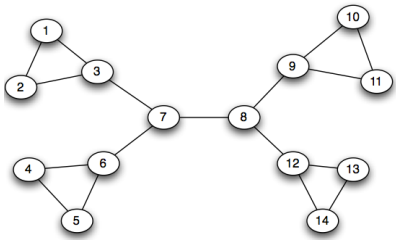
- Community detection/finding: identify communities in a network based on the structure of the network only
  - Partition a network into groups such that the intra-group edges are dense while the inter-group edges are sparse
  - It is related to graph partitioning and clustering
- Challenges
  - Finding communities within a network can be a computationally difficult task
  - The evaluation of algorithms, to determine which are better at detecting community structure, is still an open question
- Despite these difficulties, however, several methods for community finding have been developed and employed with varying levels of success

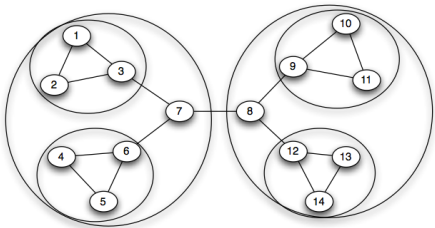
Hierarchical Clustering

- Two general ways to pull the communities out of the network
  - Divisive methods: Some consist of identifying and removing the "spanning links" between densely connected regions, and reiterate on the resulting graph
  - Agglomerative methods: Others consist of finding nodes that are likely to belong to the same region and merge them together

Hierarchical Clustering: An Example

- Consider the following network on the left

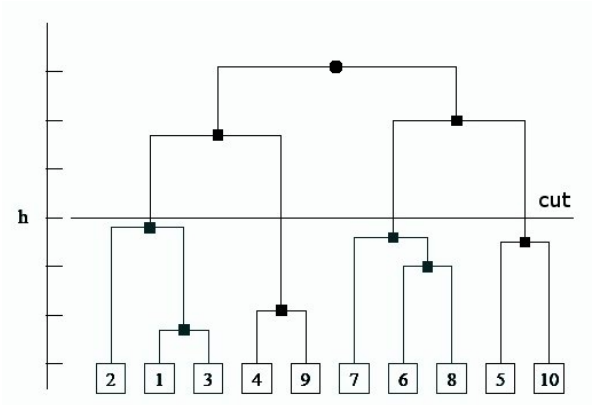




- The divisive method identifies nested regions within larger regions by removing edge $(7, 8)$ first
- The agglomerative method achieves the same result the other way: by first merging the four triangles into clumps and then pairing off triangle

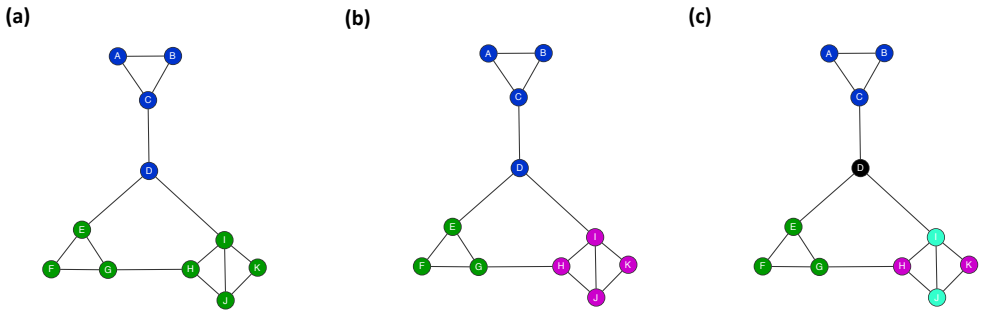
Dendrogram

- Hierarchical clustering builds a hierarchical tree, or dendrogram
  - Visualize the history of the merging or splitting process the algorithm follows
  - Horizontal cuts of this tree offer various community partitions



How to select the number of clusters? i.e., where to cut the dengrogram?

- The method gives us only a succession of splits of the network into smaller and smaller communities, but it gives no indication of which splits are best.
- One possible way is to optimize modularity.



Modularity

- Let $\mathbb{S}$ be the set of communities
- Modularity Q is a measure of how well a network is partitioned into communities
  - the fraction of the edges that fall within the given groups minus the expected such fraction if edges were distributed at random

$$Q \propto \sum_{S \in \mathbb{S}} \big((\#\text{edges within group } S) - (\text{expected } \#\text{edges within group } S) \big)$$

Modularity

- Given an undirected graph $G = (V, E)$ on n nodes and m edges, construct rewired network G'
  - Same degree distribution but random connections
  - Consider G' as a multigraph (parallel edges and self-loops are allowed)
  - The expected number of edges between nodes i and j of degrees d_i and d_j is $d_i \frac{d_j}{2m} = \frac{d_i d_j}{2m}$

- Note 1: $\sum_{i \in V} d_i = 2m$
- Note 2: The expected number of edges in (multigraph) G' is:

$$\frac{1}{2} \sum_{i \in V} \sum_{j \in V} \frac{d_i d_j}{2m} = \frac{1}{2} \cdot \frac{1}{2m} \sum_{i \in V} d_i \left(\sum_{j \in V} d_j \right) = \frac{1}{4m} \cdot 2m \cdot 2m = m$$

Modularity

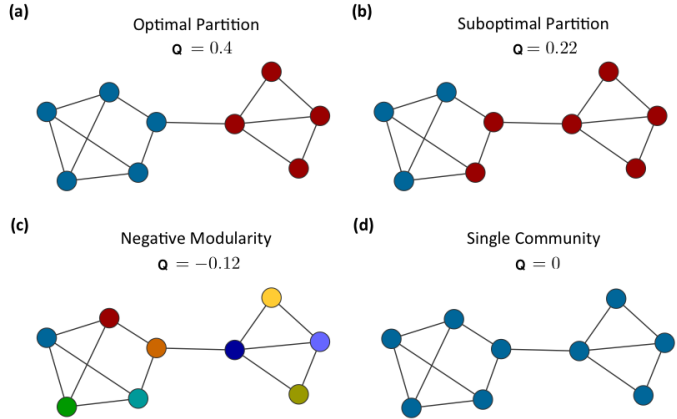
- Modularity of a partitioning $\mathbb{S}$ of a graph G:
  - $Q \propto \sum_{S \in \mathbb{S}} \big((\#\text{edges within group } S) - (\text{expected } \#\text{edges within group } S)\big)$
  - Define $Q(G, \mathbb{S}) = \frac{1}{2m} \sum_{S \in \mathbb{S}} \sum_{i \in S} \sum_{j \in S} (A_{i,j} - \frac{d_i d_j}{2m})$
    - $A_{i,j}$ is the adjacency matrix
    - $A_{i,j} = 1$ if there is a link between node i and node j, it is 0 otherwise
  - Equivalently, $Q(G, \mathbb{S}) = \sum_{S \in \mathbb{S}} \left(\frac{m_S}{m} - \left(\frac{d_S}{2m} \right)^2 \right)$
    - m_S is the number of edges within group S
    - d_S is the sum of degrees of nodes of group S
  - Modularity takes values in the range $[-1, 1]$
    - It is positive if the number of edges within groups exceeds the expected number
    - The larger the better
  - A normalized version of modularity is known as global assortativity. [2]

[2] Leto Peel, Jean-Charles Delvenne, and Renaud Lambiotte. Multiscale mixing patterns in networks. *Proceedings of the National Academy of Sciences*, 115(16):4057–4062, 2018.

Examples

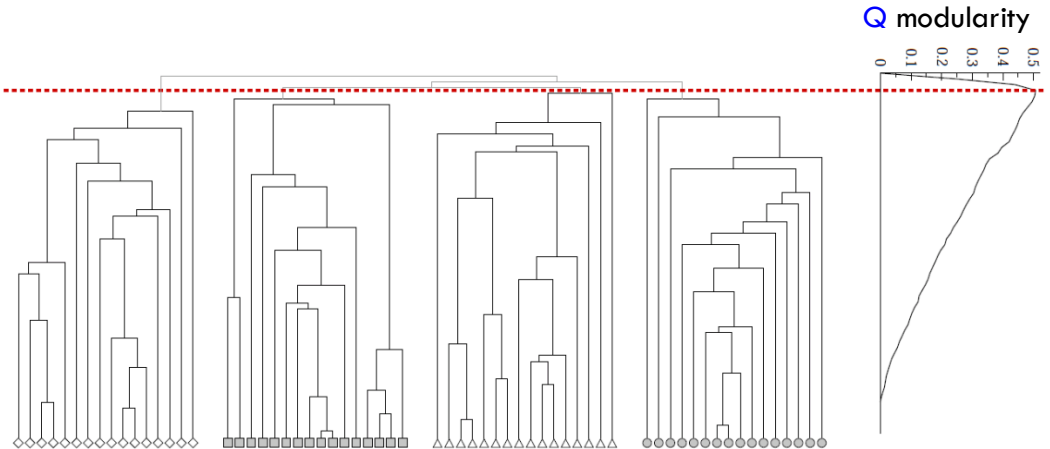
$$Q(G,S) = \sum_{S \in \mathbb{S}} \left(\frac{m_S}{m} - \left(\frac{d_S}{2m} \right)^2 \right)$$

- (a) Optimal partition, that maximizes the modularity.
- (b) Sub-optimal but positive modularity.
- (c) Negative Modularity: If we assign each node to a different community.
- (d) Zero modularity: Assigning all nodes to the same community



Select Number of Clusters

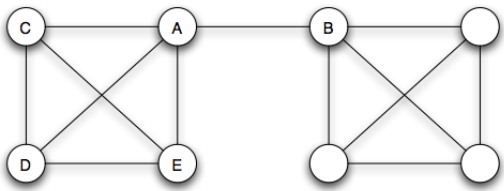
- Modularity is useful for selecting the number of clusters



Outline

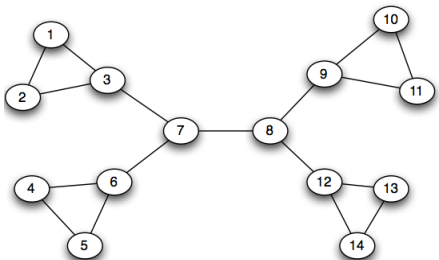
Divisive Hierarchical Clustering

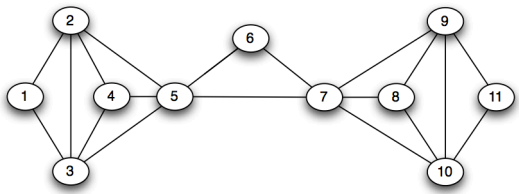
- Divisive algorithms remove the links connecting nodes that belong to different communities, eventually breaking a network into isolated clusters
- A naive idea is to remove bridges or local bridges



Divisive Hierarchical Clustering

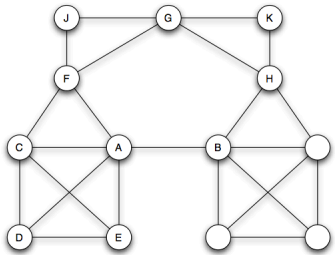
- This is insufficient:
  - It does not tell us which local bridge to remove first among multiple ones
  - Even if all edges belong to a triangle (no local bridges), a subdivision may be needed





Divisive Hierarchical Clustering

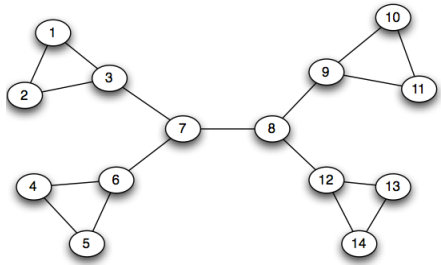
- However, local bridges are important
  - They form parts of the shortest path between pairs of nodes in different parts of the network
  - Without local bridges, paths between some pairs of nodes would be re-routed through a longer way



- We thus define an abstract notion of "traffic" for edges, and look for the edges that carry the most of this traffic.

Edge Betweenness

- For each unordered pair of nodes A and B that are connected by a path, we consider one unit of flow along the edges from A to B.
  - The flow divides itself evenly along all possible shortest paths from A to B
    - If there are k shortest paths from A to B, then $\frac{1}{k}$ flow passes along each path



- The betweenness of an edge is the total amount of flow it carries
  - Take into account the flow between all pairs of nodes using this edge
- Consider, for example, edge $(7, 8)$
  - Full flow unit from pairs of nodes on different sides of the graph passes through it
  - No flow unit from pairs of nodes on the same side passes through it

$\Longrightarrow$ The betweenness of this edge is 7 x 7 = 49

The Girvan-Newman Method

Girvan-Newman method (Input Graph G):

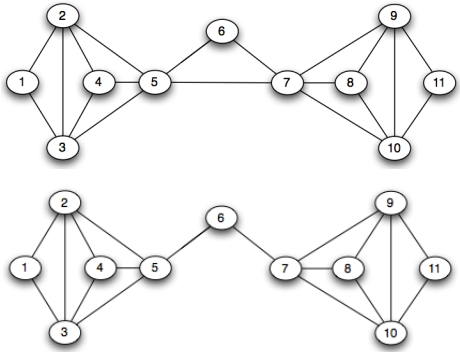
Initially:
$i = 1$

Loop:
As long as there are edges in G, repeat:

- Calculate the highest betweenness of G's edges
- Remove all edges with the highest betweenness from G
- If this splits the graph into new components
  - call the connected components the i^{th} level regions
  - increment i by 1

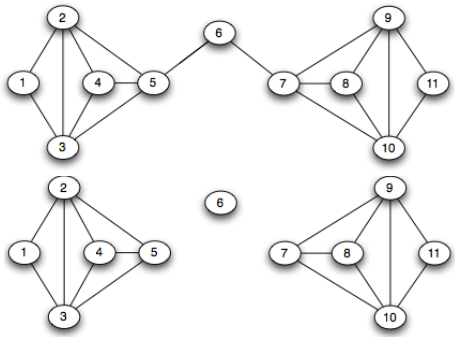
End

The Girvan-Newman Method (running example)



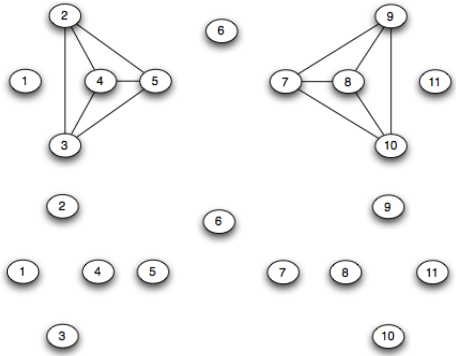
1. Edge $(5, 7)$ carries all traffic from nodes 1–5 to nodes 7–11 for a betweenness of 25. The $(5, 6)$ edge only carries flows from 6 to 1–5 for a betweenness of 5 (same for the $(6, 7)$ edge)

The Girvan-Newman Method (running example)



2. All 25 units of flow that used to be on this deleted edge have shifted onto the path through nodes 5, 6, and 7, and so the betweenness of the $(5, 6)$ edge (and also the $(6, 7)$ edge) has increased to $5 + 25 = 30$

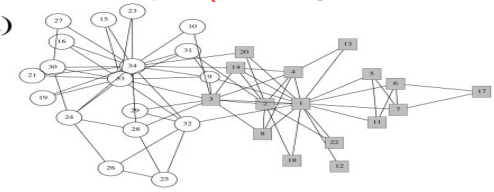
The Girvan-Newman Method (running example)



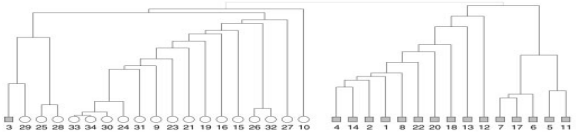
3. Edges $(1,2)$, $(1,3)$, $(9,11)$, $(10,11)$ have the highest betweenness hence they are removed next
4. The remaining edges obtain the same betweenness and are removed last

A Real Example (Zachary's Karate Club)

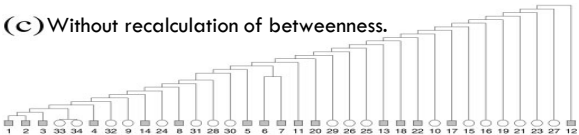
(a)



(b) With recalculation of betweenness.



(c) Without recalculation of betweenness.

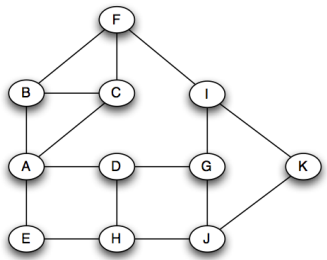


How to efficiently compute betweenness?

- This method can be costly to run on a large data set
- The betweenness quantity requires to reason about all shortest paths between pairs of nodes and should be computed for each remaining edge at each step, which can take very long time

Betweenness Computation

- The betweenness of edge (a, b) is $B(a, b) = \sum_{i<j} \frac{\delta(i,j \mid a,b)}{\delta(i,j)}$
  - $\delta(i, j)$ is the number of shortest paths from i to j
  - $\delta(i, j \mid a, b)$ is the number of shortest paths from i to j that go through edge (a, b)
- Consider the graph from one node at a time
  - For a fixed i, compute $\sum_j \frac{\delta(i,j|a,b)}{\delta(i,j)}$
  - Let's see how the flow from this node to all others spread along the edges
- Then, by adding the flows we obtain the betweenness on every edge
  - $B(a, b) = \frac{1}{2} \sum_i \sum_j \frac{\delta(i,j|a,b)}{\delta(i,j)}$



Example: consider this graph and how the flow from A reaches all nodes

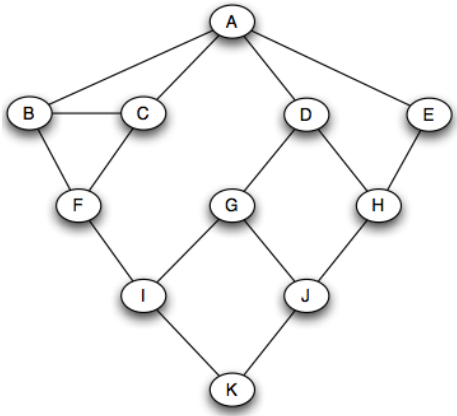
Betweenness Computation

To compute $\sum_j \frac{\delta(i,j|a,b)}{\delta(i,j)}$ for a fixed node i

1. Perform a breadth-first search on the graph starting at i
2. Determine the number of shortest paths from i to every other node
3. Based on these numbers, determine the amount of flow from i to all other nodes that use each edge

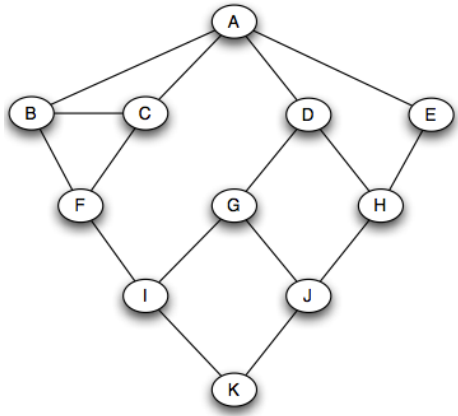
Betweenness Computation

1. Executing breadth-first search (BFS)
   - Breadth-first search divides a graph into layers with all nodes in layer d at distance d from the source (A, in our example).
   - The shortest paths from A to any node is precisely the downward path through increasing layers from A to this node
   - There are two shortest paths from A to F, each of length 2:
     - One is A, B, F, the other is A, C, F.



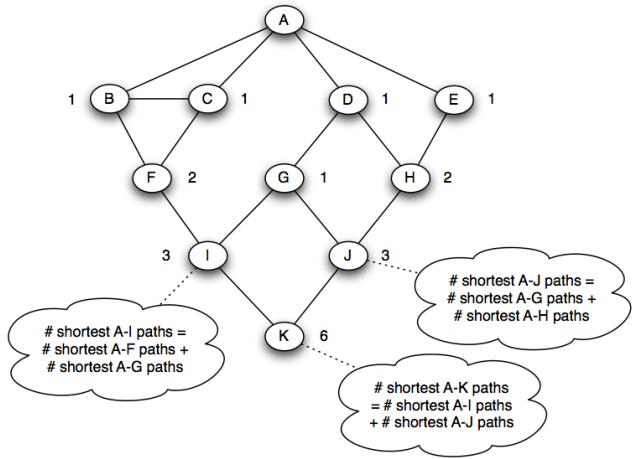
Betweenness Computation

2. Determining the number of shortest paths
   - Note that all shortest paths from A to I must take their last step through F or G since these are the two nodes above I in the breadth-first search
   - To be shortest path to I, it must be a shortest path to F or G followed by the corresponding one of these two last edges



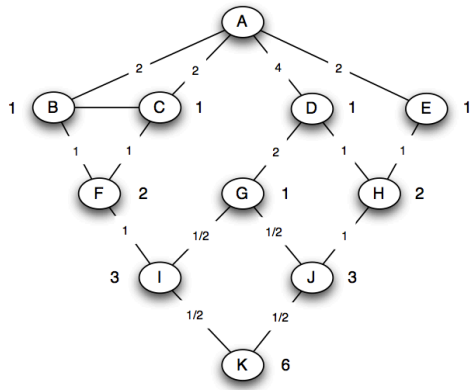
Betweenness Computation

2. Determining the number of shortest paths (con't)
   We can generalize this:
   - Each node in the first layer is a neighbor of A, so it has only one shortest path to A
   - We move down through the BFS layers and compute the number of shortest paths as the sum of the number of shortest paths to all nodes directly above it in the BFS.



Betweenness Computation

3. Determining flow values
   - **Let's start at the bottom layer (with node K):** A single unit of flow arrives to K and an equal number of shortest paths from A to K go through I and J so this unit of flow is equally divided between (I,K) and (J,K)
   - **Let's go upward:** the total amount of flow arriving at I is equal to the one unit actually destined for I + the half unit passing through to K = 3/2. How does this get divided over the edges leading upward from I to F and G, respectively? We see that there are twice as many shortest paths from A through F as through G, so twice as much flow should come from F, thus we put 1 unit of flow on F and a half-unit of the flow on G.
   - We continue like this for each other node working upward through the layers



Betweenness Computation

- To complete the process, we build this BFS structure for each node in the network, determine flow values using this procedure and then sum up the flow values to get the betweenness value for each edge.

- Note that we count the flow values between each pair of nodes X, Y twice: once we do the BFS from X and once we do it from Y. So at the end, we divide everything by 2 to cancel out this double counting.

Complexity Analysis

- For any one target node, BFS gives betweenness of every edge w.r.t. that target node, in $O(m)$ time
  - m is the number of edges in a network
- Doing so for every node treated as target node requires $O(nm)$ time for final betweenness score for every edge
  - n is the number of nodes in a network
- The Girvan-Newman method is quite elegant, but recalculation of betweenness bumps up complexity to $O(nm^2)$

- Instead of implementing the algorithm from scratch, you can invoke

  NetworkX.edge_betweenness_centrality(G, k=None, normalized=False, weight=None, seed=None)
- Do not use **NetworkX.girvan_newman(G)** in assessments.

Outline

Agglomerative Algorithms

- Agglomerative algorithms iteratively merge nodes and communities
  - To choose which edge to merge, we compute a similarity matrix
    - Measures how similar two nodes are to each other, for each pair of nodes
  - Agglomerative methods merge nodes and communities that have high similarity

- The Ravasz algorithm is proposed to identify functional modules in metabolic networks [3]
  - Phase 1: define the similarity matrix
  - Phase 2: decide group similarity
  - Phase 3: apply hierarchical clustering
  - Phase 4: build the dendrogram

[3] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, A.-L. Barabasi. Hierarchical Organization of Modularity in Metabolic Networks. Science, 2002.

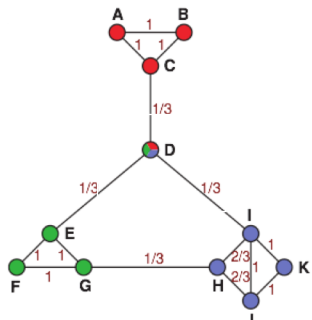
Ravasz Algorithm

Phase 1: Define the similarity matrix

- High for node pairs that likely belong to the same community
- Low for those that likely belong to different communities
- Nodes that connect directly to each other and share multiple neighbors are more likely to belong to the same community, hence their similarities should be large

Topological overlap similarity: $x_{ij}^0 = \dfrac{J(i,j)}{\min\{d_i, d_j\}}$

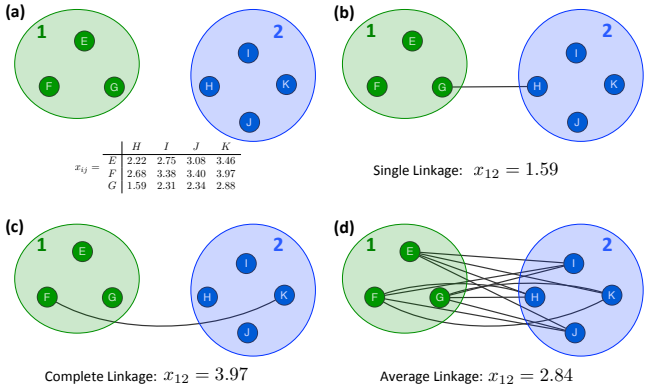
d_i: the degree of node i
$J(i,j)$: the number of common neighbors of node i and j; $+1$ if there is a direct link between i and j



Ravasz Algorithm

Phase 2: Decide group similarity

- As nodes are merged into small communities, we must measure how similar two communities are
  - Single (i.e., min), complete (i.e., max), and average cluster similarity



$x_{ij} =$

	H	I	J	K
E	2.22	2.75	3.08	3.46
F	2.68	3.38	3.40	3.97
G	1.59	2.31	2.34	2.88

Single Linkage: $x_{12} = 1.59$

Complete Linkage: $x_{12} = 3.97$

Average Linkage: $x_{12} = 2.84$

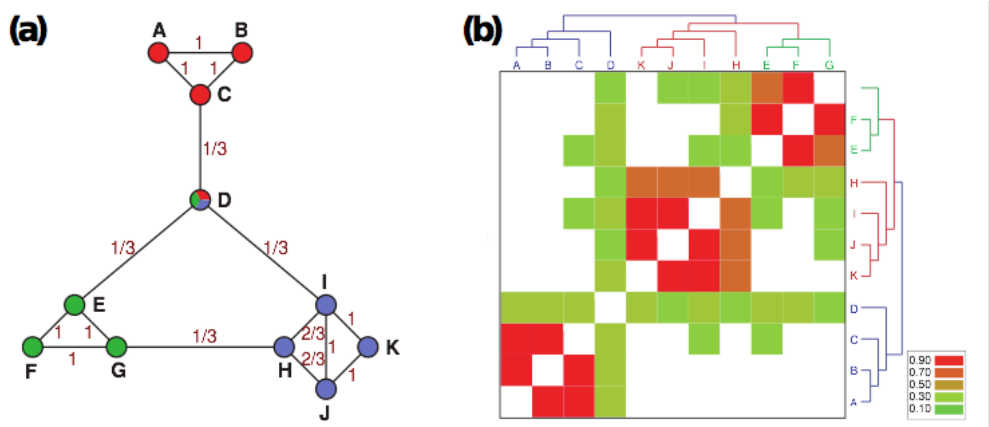
Ravasz Algorithm

Phase 3: Apply hierarchical clustering

1. Assign each node to a community of its own and evaluate similarity for all node pairs. The initial similarities between these "communities" are simply the node similarities
2. Find the community pair with the highest similarity and merge them to form a single community
3. Calculate the similarity between the new community and all other communities
4. Repeat from Step 2 until all nodes are merged into a single community

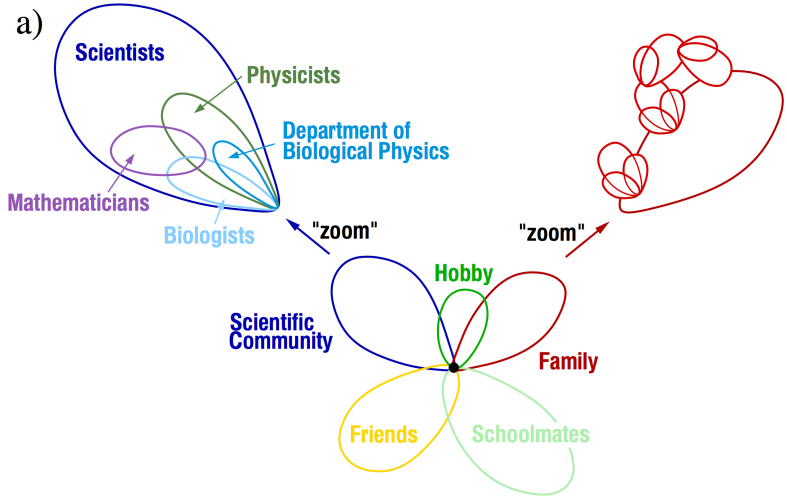
Phase 4: Build dendrogram

- Describes the precise order in which the nodes are assigned to communities



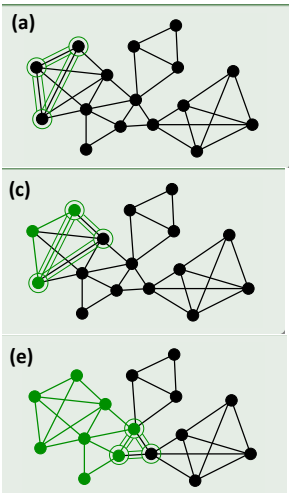
Outline

Overlapping Communities



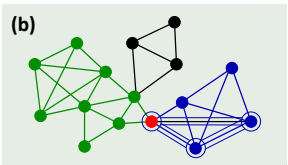
Clique Percolation

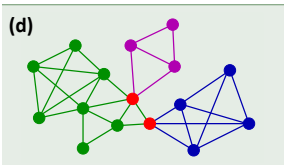
- Two k-cliques (complete subgraphs of k nodes) are considered adjacent if they share $k-1$ nodes
- A k-clique community is the largest connected subgraph obtained by the union of all adjacent k–cliques
- Other k-cliques that can not be reached from a particular k-clique correspond to other k-clique-communities



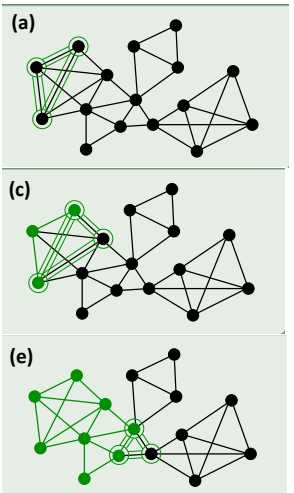
Clique Percolation

- Other k-cliques that can not be reached from a particular k-clique correspond to other k-clique-communities





Software: https://www.cfinder.org



Conclusion

- Community structure is a cohesive group of nodes that are connected "more densely" to each other, than to nodes outside the community
  - Community structures are quite common in real networks.
- Communities can either be computed by divisive methods or agglomerative methods
- Modularity measures how well a network is partitioned into communities

Other Community Detection Methods

- Louvain: Based on Modularity optimization
  - http://perso.uclouvain.be/vincent.blondel/research/louvain.html
- METIS: Heuristic but works really well in practice
  - https://github.com/KarypisLab/METIS
- Graclus: Based on kernel k-means
  - https://www.cs.utexas.edu/~dml/Software/graclus.html

Reading

- Reading for this week
  - Advanced material of Chapter 3 of the textbook
  - Chapters 9.1–9.4 of Network Science (http://networksciencebook.com/chapter/9)
- Reading for next week
  - Chapter 13 of the textbook
  - Chapter 14 of the textbook, excluding the PageRank part