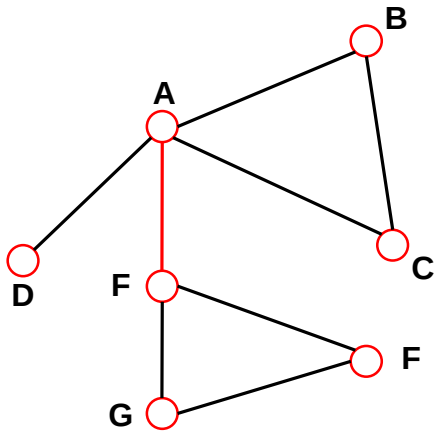
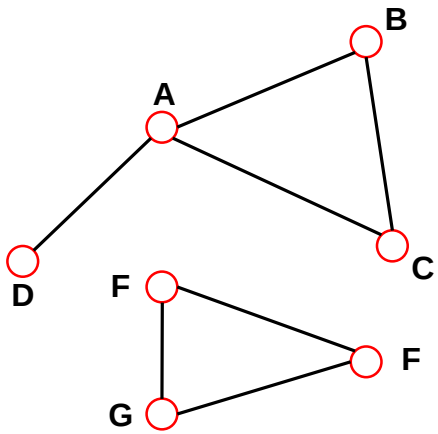
# Community Structure in Networks

# Connectivity

Connected (undirected) graph: any two vertices can be joined by a path.
A disconnected graph is made up by two or more connected components.




Largest Component:
**Giant Component**

The rest: **Isolates**

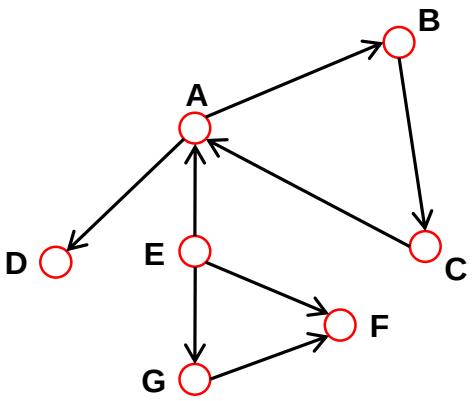
Bridge: if we erase it, the graph becomes disconnected.

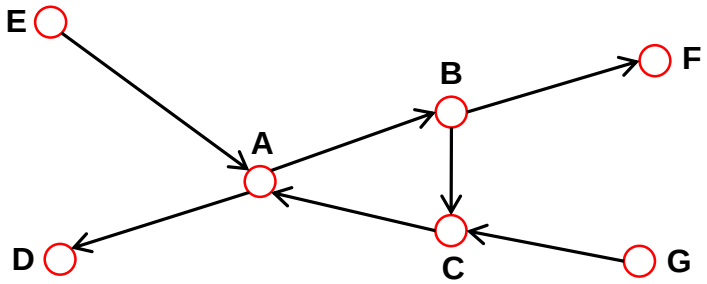
# Connectivity

Strongly connected directed graph: has a path from each node to every other node and vice versa (e.g. AB path and BA path).

Weakly connected directed graph: it is connected if we disregard the edge directions.

Strongly connected components can be identified, but not every node is part of a nontrivial strongly connected component.




In-component: nodes that can reach the scc,

Out-component: nodes that can be reached from the scc.

# Community Finding

Given a network that is generally connected, how do we find communities?

Goals:

- Edge density **within** communities is **high.**
- Edge density **between** communities is **low.**

# Modularity

Modularity was proposed by Newman and Girvan (2004) as a way to measure the quality of a given partition of nodes. It is a scalar in the interval *[-1,1]*. For a given partition of set *V* (denoted *C*), modularity is defined as follows:

$$M(C) = \frac{1}{2m} \sum_{c \in C} \sum_{i,j \in c} w_{ij} - \frac{k_i k_j}{2m}$$

Where $w_{ij}$ is the weight of an edge between nodes *i* and *j* (set to zero if there is no edge, in an unweighted graph, this is set to 1 if there is an edge) and *m* is the number of edges (or sum of edge weights).

# Modularity Maximization

Hence, a natural way to find an “optimal” community structure is to search for a partition of nodes that maximizes *M.*

However, modularity maximization is an *NP-hard* problem, which leads us to use heuristics.

The first heuristic was proposed by Newman and Girvan – a greedy approach.

# Greedy Approach

- Initialize each node in a single community – this is the initial *C*
- Flag = true
- While  flag:
  - Iterate through all edges, pick the edge that joins two communities and increases modularity
    - If there is no such edge, Flag = false
  - Merge the two communities connected by that edge; update data structures
- Return resulting C

Takes O((m+n)n) time, some improvement possible with data structures, etc.

# Faster Approach: Local Search (Blondel et al., 2008)

"Louvain Algorithm"

- Initialize each node in a single community – this is the initial *C*
- BigFlag = true
- While BigFlag
  - Flag = true
  - While flag:
    - For each node, consider the gain in modularity if it is removed from its community and placed in one of its neighbors communities; if there is a gain, then do so.
      - If no node provides a gain in modularity, the Flag = False
      - If we only iterated through the nodes once, then BigFlag = False
  - Collapse communities into single nodes (an aggregate graph), create edges between communities weighted by number of connections
- Return communities represented by the nodes in the aggregate graph

Observed to take approx. linear time in practice.

Ordering of nodes does not seem to impact results

Generally obtains higher modularity than other approaches.

Shown to find real-world communities in many applications.

# Label-Propagation Community Finding

Simple, “near linear-time” algorithm (Ragavan et al.)

- No Unique solution
- Each iteration takes O(m) time
- Sequential and simultaneous update versions (sequential converges quicker, simultaneous provides a more stable solution)
- Does not provide a unique solution – and there may be a large number of solutions possible

1. Initialize the labels at all nodes in the network. For a given node $x$, $C_x(0) = x$.
2. Set $t = 1$.
3. Arrange the nodes in the network in a random order and set it to $X$.
4. For each $x \in X$ chosen in that specific order, let $C_x(t) = f(C_{x_{i1}}(t), ..., C_{x_{im}}(t), C_{x_{i(m+1)}}(t-1), ..., C_{x_{ik}}(t-1))$. $f$ here returns the label occurring with the highest frequency among neighbors and ties are broken uniformly randomly.
5. If every node has a label that the maximum number of their neighbors have, then stop the algorithm. Else, set $t = t + 1$ and go to (3).

# Game-Theoretic Community Finding

Modularity and graph cuts converge based on a global property of the network.

However, real communities are often formed by individuals without concern for the overall global knowledge

Game Theoretic Approach: Each node in the network is a playing a game where he chooses community membership such that a payoff function is maximized (i.e. Lung et al use the following):

$$f_C = \frac{k_{in}^C}{(k_{in}^C + k_{out}^C)^\alpha}$$

Resulting community partition is results from an equilibrium condition (i.e. Nash equilibrium – no player improves payoff by unilaterally shifting strategies/communities)