Graph Theory

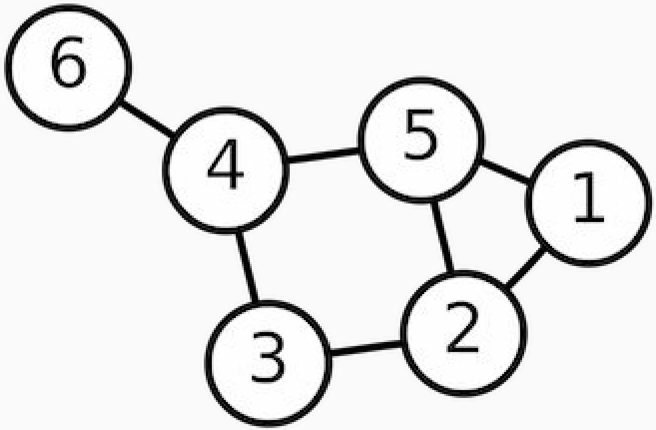
Motivation

Model connections between things (people, places, objects, events, etc...) in a common mathematical framework.

Examples

Facebook Friends

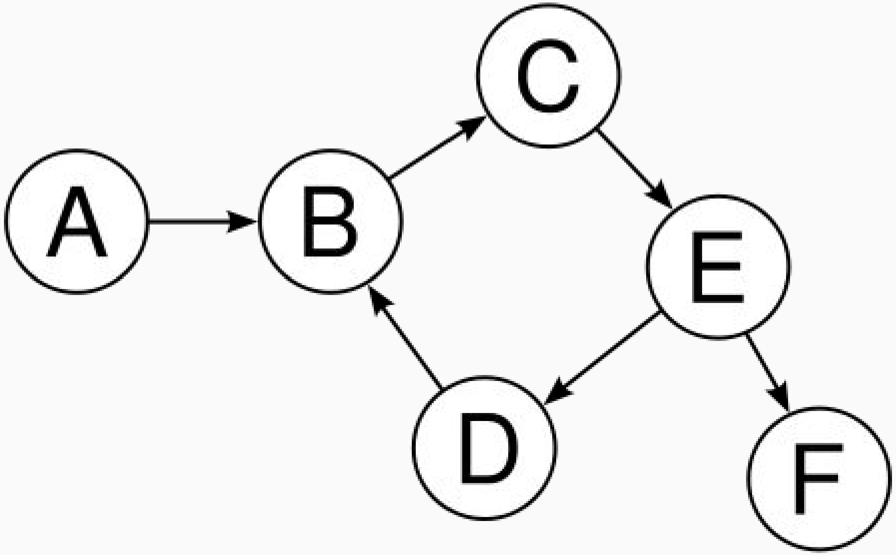
LinkedIn Connections



Credit: Wikipedia

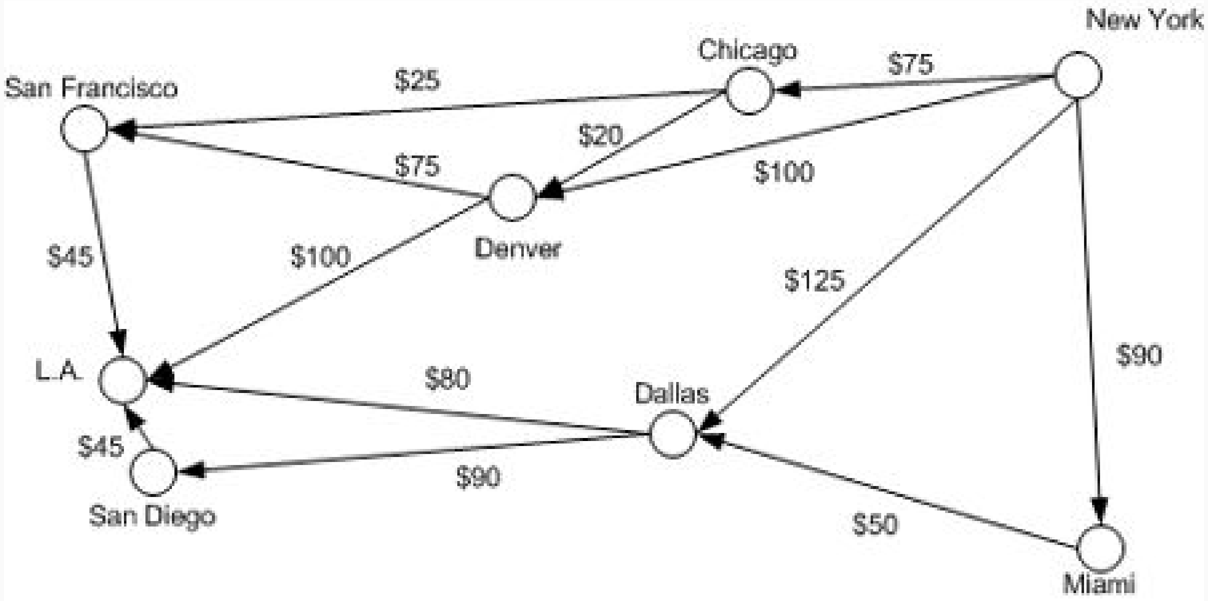
Examples

Twitter Followers



Examples

Airline Network



Examples

Lots more ...

Graph Definition

Graphs (aka Networks) consist of vertices (aka nodes) linked to each other by edges (aka arcs).

Mathematically:

- The graph ***G*** is an ordered pair ***G*** = (V, E) consisting of two sets:
  - V is the set of all vertices in graph ***G***
  - E is the set of all edges connecting the vertices of graph ***G***, such that each element of E is a two element subset of V: {Vi, Vj}

Terminology

- Directed: Graph where edges between vertices go only in one direction.
- Undirected: Graph where edges go both ways.

- Weighted: Edges have weights based on some value, such as price, distance, time, counts between vertices.
- Unweighted: All edges are the same. (Often represented as a weighted graph where edge weights are all set to 1)

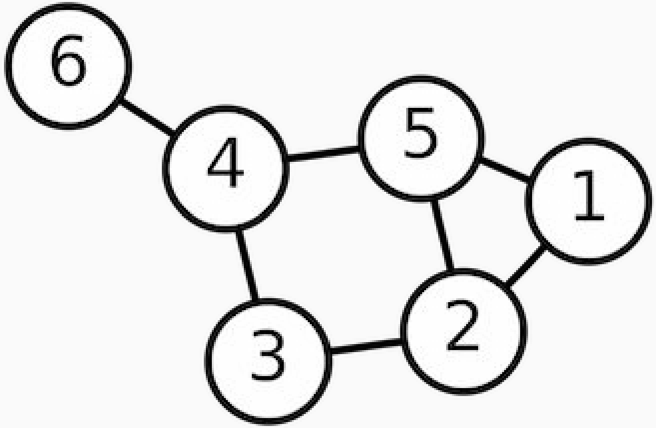
More Terminology

- **Neighbors**: The *neighbors* of a vertex are the vertices that it is connected to. e.g., B is a neighbor of A if there is an edge from A to B.
- **Degree**: The *degree* is the number of neighbors a vertex has. In directed graphs distinguish between *outdegree* and *indegree.*
- **Path**: A *path* is a series of nodes and the edges that connect them.
- **Connected**: A graph is *connected* if there is a path from every node to every other node.
- **Subgraph**: A *subgraph* is a subset of the nodes of a graph and all the edges between them.
- **Connected Component**: A *connected component* is a subgraph that is *connected*.

Representing Graphs

Adjacency Matrix

```
     1 2 3 4 5 6
1 [[0 1 0 0 1 0]
2  [1 0 1 0 1 0]
3  [0 1 0 1 0 0]
4  [0 0 1 0 1 1]
5  [1 1 0 1 0 0]
6  [0 0 0 1 0 0]]
```



Credit: Wikipedia

Representing Graphs

Adjacency Lists

1: {2, 5}

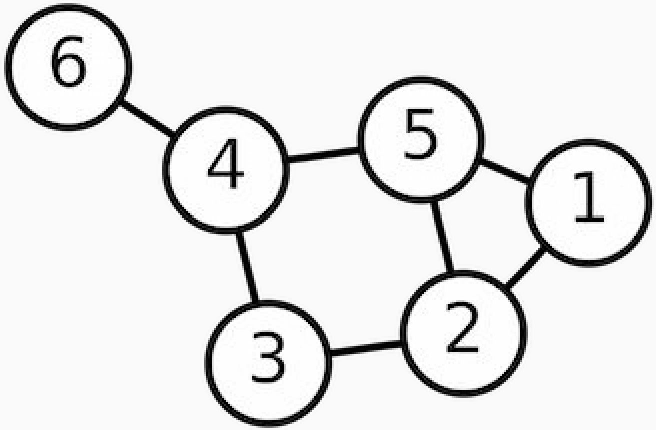
2: {1, 3, 5}

3: {2, 4}

4: {3, 5, 6}

5: {1, 2, 4}

6: {4}



Credit: Wikipedia

Representing Graphs

- Storage:
  - Adjacency List: **O**(|V| + |E|)
  - Adjacency Matrix: **O**(|V|^2)

- Time to lookup an edge between two vertices:
  - Adjacency List: **O**(|V|) … (depending on implementation)
  - Adjacency Matrix: **O**(1)

Graph Search

- Are two vertices connected?
  - Do two people have friends in common?
  - Can I drive from point A to point B?
  - Is there a dependency between two steps in an industrial process?
- What are the connected components?
  - How many non-intersecting groups of friends are in a social network?
- What is the shortest path between two vertices?

Generic Graph Search

Build set of all vertices that can be reached from a starting point.

Goals: Find everything “findable” from a start vertex.
Don’t explore anything twice.

Pseudocode:

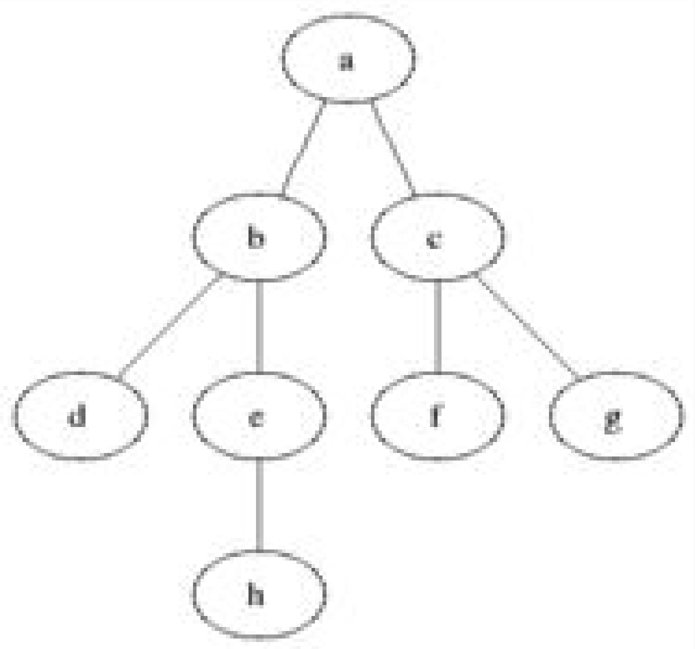
- Given graph ***G***, starting vertex ***s***
  - Add ***s*** to explored
  - While possible:
    - Choose an edge (***u, v)*** where ***u*** is in explored and ***v*** is NOT in explored
    - Add ***v*** to explored

Generic Graph Search

- Requires some design choices for implementation:
  - Recall: “Choose an edge (***u, v)*** where ***u*** is in explored and ***v*** is NOT in explored”
    - How this edge is chosen affects results and runtime.
      - Breadth First vs. Depth First

Breadth-First Search

Starting with a given vertex, find all of that vertex's Neighbors. Then find all of those neighbors' neighbors, and so on.



Credit: Wikipedia

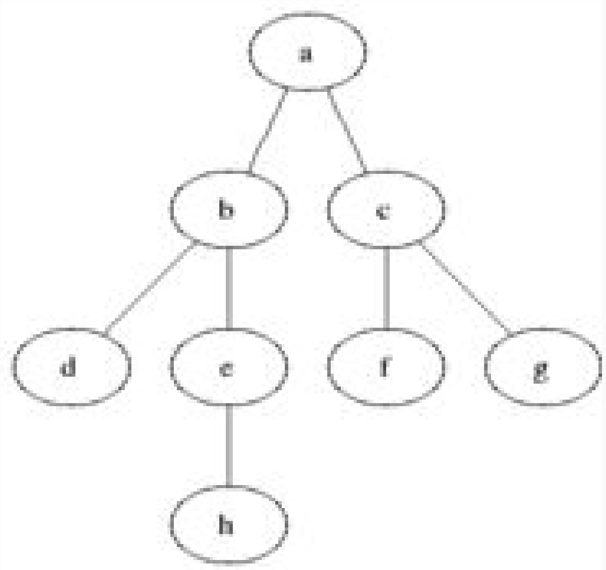
Write BFS in Python

Depth-First Search

Choosing next vertex's edges to traverse:

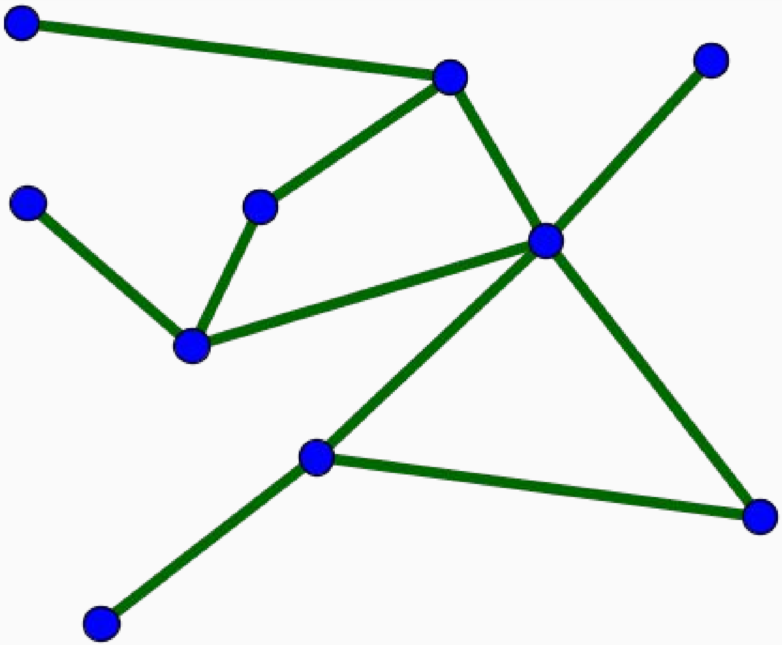
Use a Last In First Out data structure (aka Stack)

(well-suited for searching directed graphs)



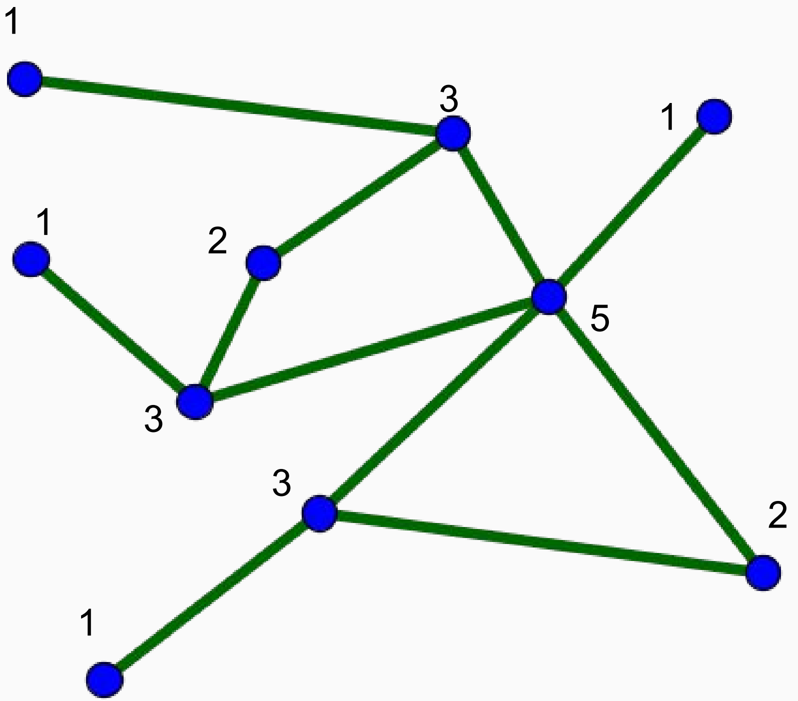
Social Network Analysis

- Identify “Important” Vertices
- Detect Communities



Degree Centrality

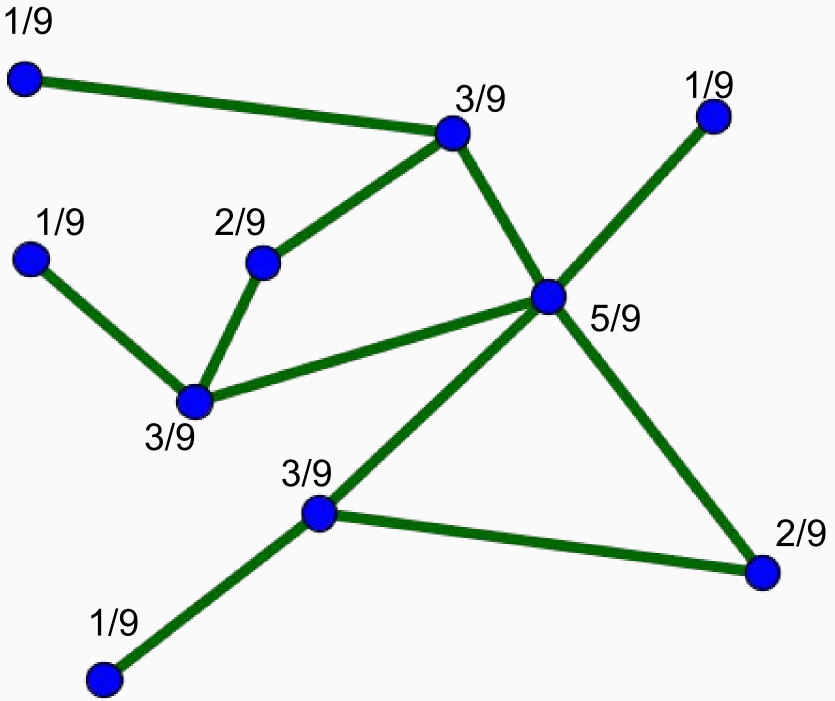
Vertices with high degree are the "celebrities" of the network



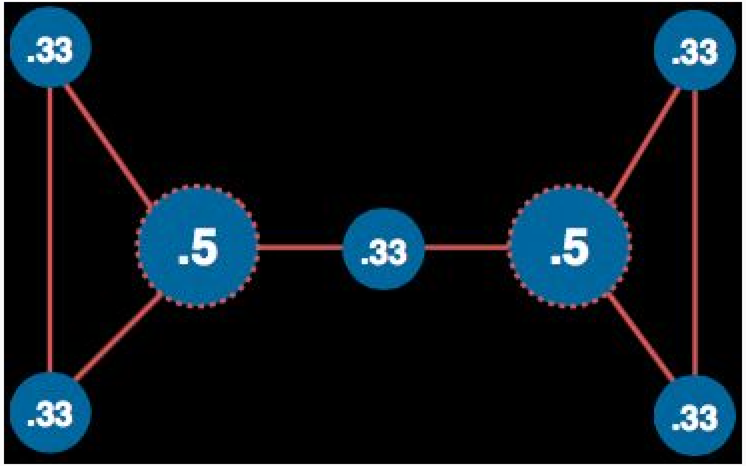
Normalized Degree Centrality

Useful to compare across graphs, so want a normalized measure of centrality.

$$\frac{\text{Degree of Vertex}}{\text{\# Vertices in graph - 1}}$$

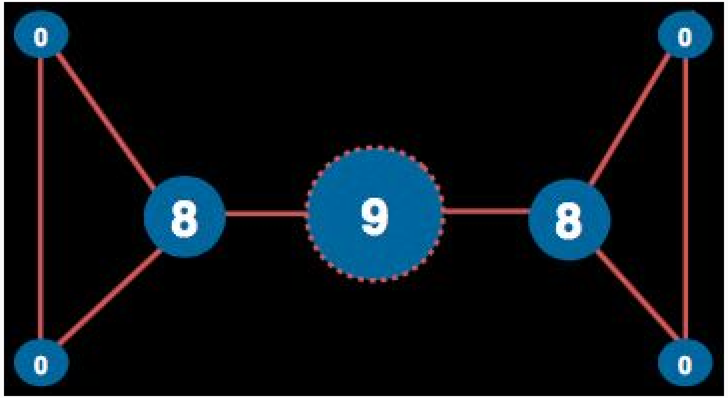


Is Degree Centrality the best measure of importance here?



Betweenness Centrality

Vertices that are part of a large number of shortest paths are “community bridgers”



Normalized Betweenness Centrality

$$\text{betweenness}(v) = \sum_{s \neq v \neq t} \text{percent of shortest paths from } s \text{ to } t \text{ which pass through } v$$

$$= \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where

$$\sigma_{st}(v) = \# \text{ of shortest paths from } s \text{ to } t \text{ which pass through } v$$

$$\sigma_{st} = \# \text{ of shortest paths from } s \text{ to } t$$

$$\text{normalized betweenness}(v) = \frac{\text{betweenness(v)}}{(n-1)(n-2)}$$

(For directed graphs)

Number of pairs of vertices with vertex v excluded.
For undirected graphs that is (n-1)(n-2)/2

Betweenness Centrality Continued

$$\text{betweeness}(4) = \sum_{s \neq 4 \neq t} \frac{\sigma_{st}(4)}{\sigma_{st}}$$

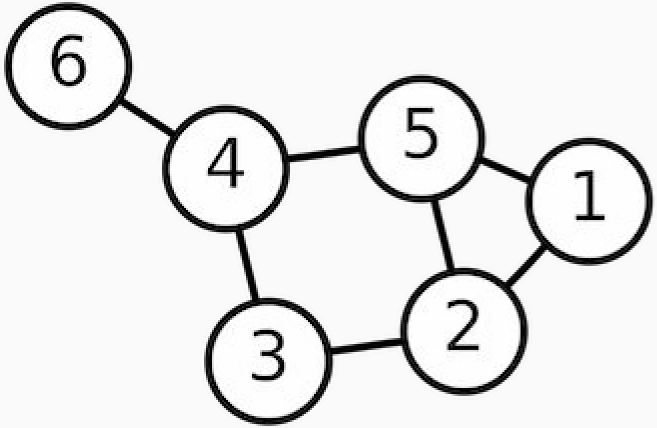
$$= \frac{\sigma_{16}(4)}{\sigma_{16}} + \frac{\sigma_{26}(4)}{\sigma_{26}} + \frac{\sigma_{36}(4)}{\sigma_{36}} + \frac{\sigma_{56}(4)}{\sigma_{56}} + \frac{\sigma_{35}(4)}{\sigma_{35}}$$

$$= \frac{1}{1} + \frac{2}{2} + \frac{1}{1} + \frac{1}{1} + \frac{1}{2}$$

$$= 4.5$$

$$\text{normalized betweeness}(4) = \frac{4.5}{(6-1)(6-2)/2} = 0.225 \; {*2}$$

$$= 0.45$$



Credit: Wikipedia

Eigenvector Centrality

A vertex's centrality depends on the centrality of its neighbors. Vertices with high scores are the "power brokers" or "Don Corleones".

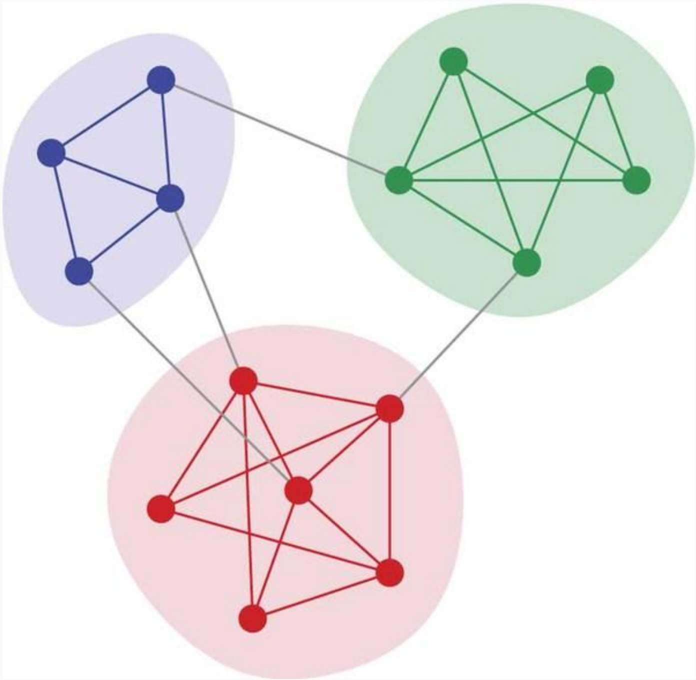
Note: Google's PageRank is a special case of eigenvector centrality

1. Start with unit centrality scores for all vertices.
2. Calculate the centrality of a vertex as the weighted sum of the centralities of all its neighbors.
3. Normalize by dividing the largest value seen in step 2.
4. Repeat 2 and 3 until convergence.

This process can be reformulated as the eigen-decomposition of the adjacency matrix.

https://en.wikipedia.org/wiki/Centrality#Eigenvector_centrality

Community Detection



Community Detection

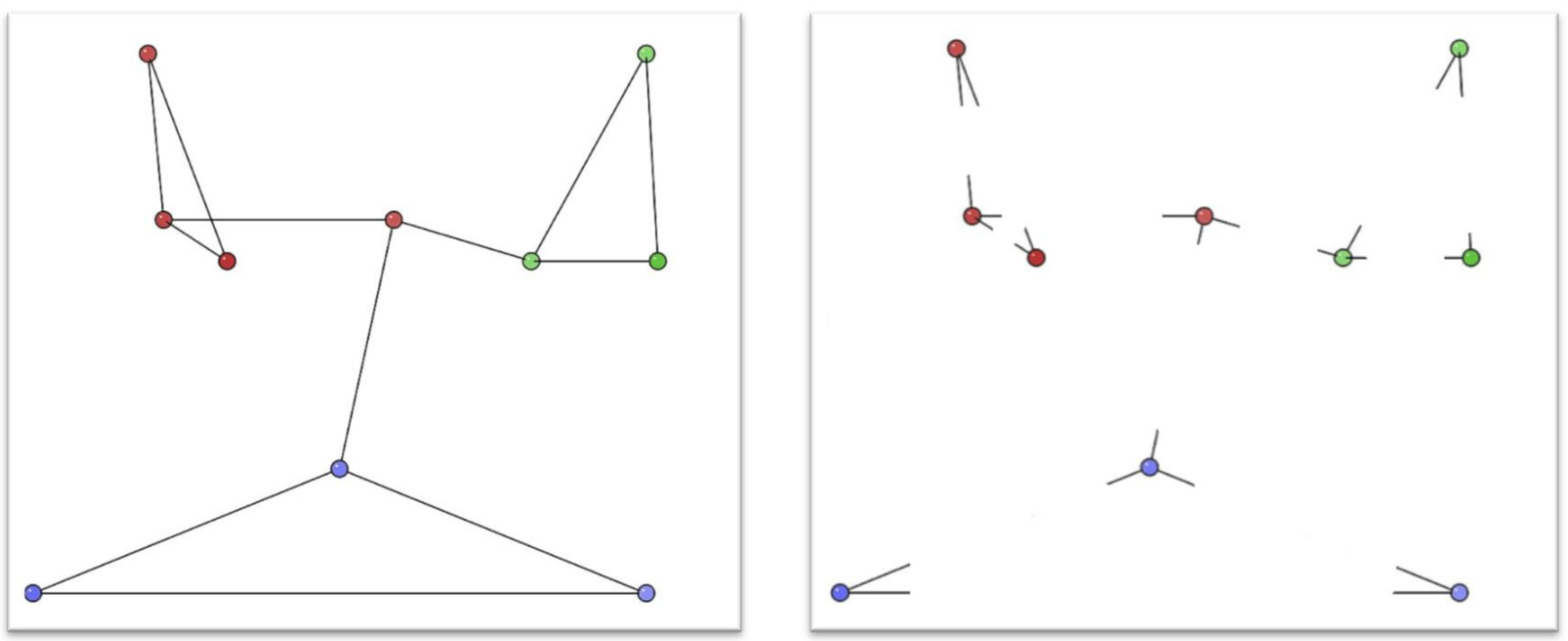
Want our communities to be tightly connected internally and loosely connected externally (i.e. have few connections between communities).

Modularity: “Tightness” of our groups of vertices.

Edge Betweenness Centrality: slight extension of vertex betweenness used to identify the rare cross-community connections.

Modularity

Compare the number of connections within a candidate community with a "random" graph with the same degree properties.



Randomized Graph

$$\text{P(single edge stub gets connected to } j) = \frac{d(j)}{2m}$$

$$\text{E(edge from } i \text{ to } j) = d(i) \cdot \frac{d(j)}{2m} = \frac{d(i)d(j)}{2m}$$

$$\text{where}$$

$$d(i) = \text{degree of node } i$$

$$m = \text{number of edges in the graph}$$

$$\text{E(edges within communities)} = \sum_{i,j \text{ in same community}} \frac{d(i)d(j)}{2m}$$

Modularity

$$\text{modularity}(G, \mathcal{C}) = \frac{1}{2m} \sum_{C \in \mathcal{C}} \sum_{i,j \in C} A_{ij} - \frac{d(i)d(j)}{2m}$$

$$\begin{aligned} \text{where} \\ \mathcal{C} &= \text{the collection of communities} \\ m &= \text{number of edges in the graph} \\ A_{ij} &= \begin{cases} 1 & \text{if } (i,j) \text{ is an edge} \\ 0 & \text{if } (i,j) \text{ is not an edge} \end{cases} \\ d(i) &= \text{degree of node } i \end{aligned}$$

Modularity

$$\text{modularity}(G,\mathcal{C}) = \frac{1}{2m}\overbrace{\sum_{C\in\mathcal{C}}\overbrace{\sum_{i,j\in C} \overset{\text{edges in commun.}}{A_{ij}} - \overset{\text{expected edges if random}}{\frac{d(i)d(j)}{2m}}}^{\text{In community } C}}^{\text{Across all communities in graph}}$$

where

$$\mathcal{C} = \text{the collection of communities}$$

$$m = \text{number of edges in the graph}$$

$$A_{ij} = \begin{cases} 1 & \text{if } (i,j) \text{ is an edge} \\ 0 & \text{if } (i,j) \text{ is not an edge} \end{cases}$$

$$d(i) = \text{degree of node } i$$

Edge Betweenness Centrality

Edges with high betweenness are likely to occur between communities, recall that when we calculated betweenness for vertices we called them "community bridgers".

Use this knowledge to isolate communities.

$$\text{betweenness}(e) = \sum_{s \neq v \neq t} \text{percent of shortest paths from } s \text{ to } t \text{ which pass through } e$$

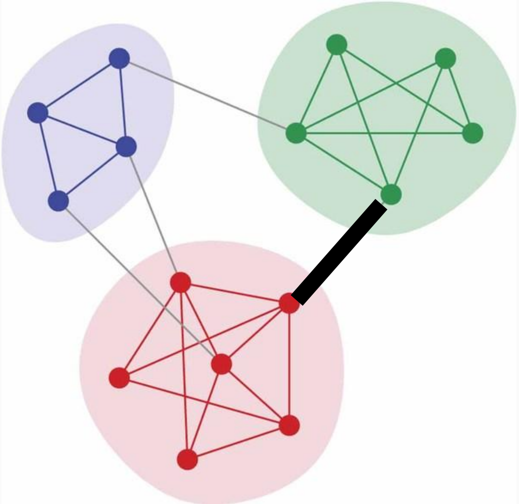
$$= \sum_{s \neq v \neq t} \frac{\sigma_{st}(e)}{\sigma_{st}}$$

where

$$\sigma_{st}(e) = \#\text{ of shortest paths from } s \text{ to } t \text{ which pass through } e$$

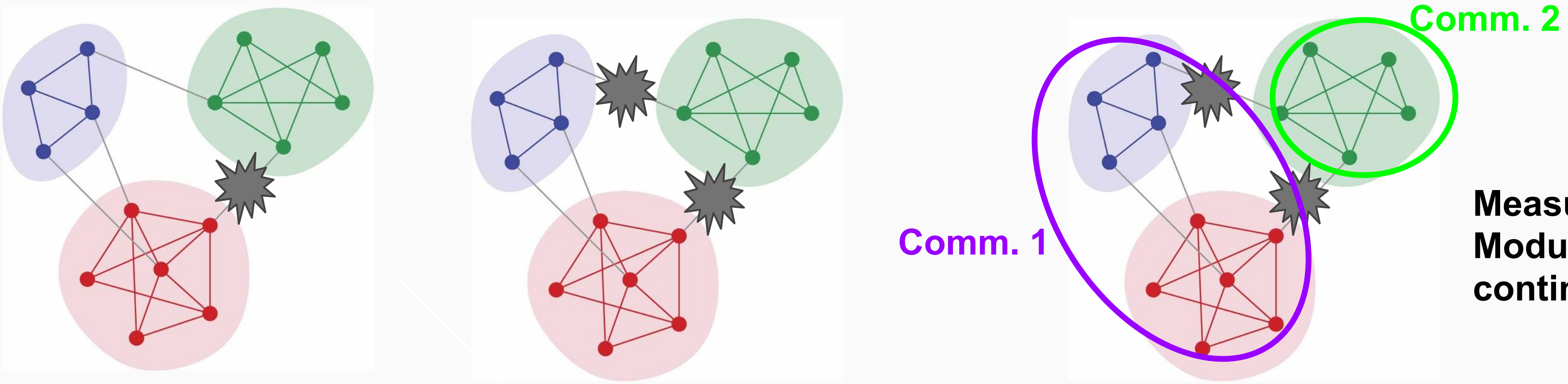
$$\sigma_{st} = \#\text{ of shortest paths from } s \text{ to } t$$

$$\text{normalized betweenness}(e) = \frac{\text{betweenness(e)}}{(n-1)(n-2)}$$



Girvan - Newman

Greedy algorithm identifies communities by repeatedly cutting edges with high betweenness until a new connected component is formed. Measure modularity after each of these events, pick communities with highest modularity.



Measure Modularity and continue...

Pseudocode

```
function GirvanNewman:
    repeat:
        repeat until a new connected component is created:
            calculate the edge betweenness centralities for all the edges
            remove the edge with the highest betweenness
```

Python Graph Packages

- Networkx: suitable for small graphs (up to ~10,000 vertices)
- igraph: works for larger graphs (C code)
- Graph-tool: even bigger graphs (Heavily optimized C code) (Pain to install)