NUS | Computing
National University of Singapore

CS5228: Knowledge Discovery and Data Mining

Lecture 9 — Graph Mining

Course Logistics

- Deadline Reminders
  - Submission of A4: Nov 14, 11.59 pm
  - Submission of project report: Nov 14, 11.59 pm
- Project
  - Check your TEAMMATES result

Quick Recap

- Recommender Systems
  - Problem: information overload (too many items: products, songs, movies, news articles, restaurants, etc.)
  - Goal: identify relevant items matching a user's preferences

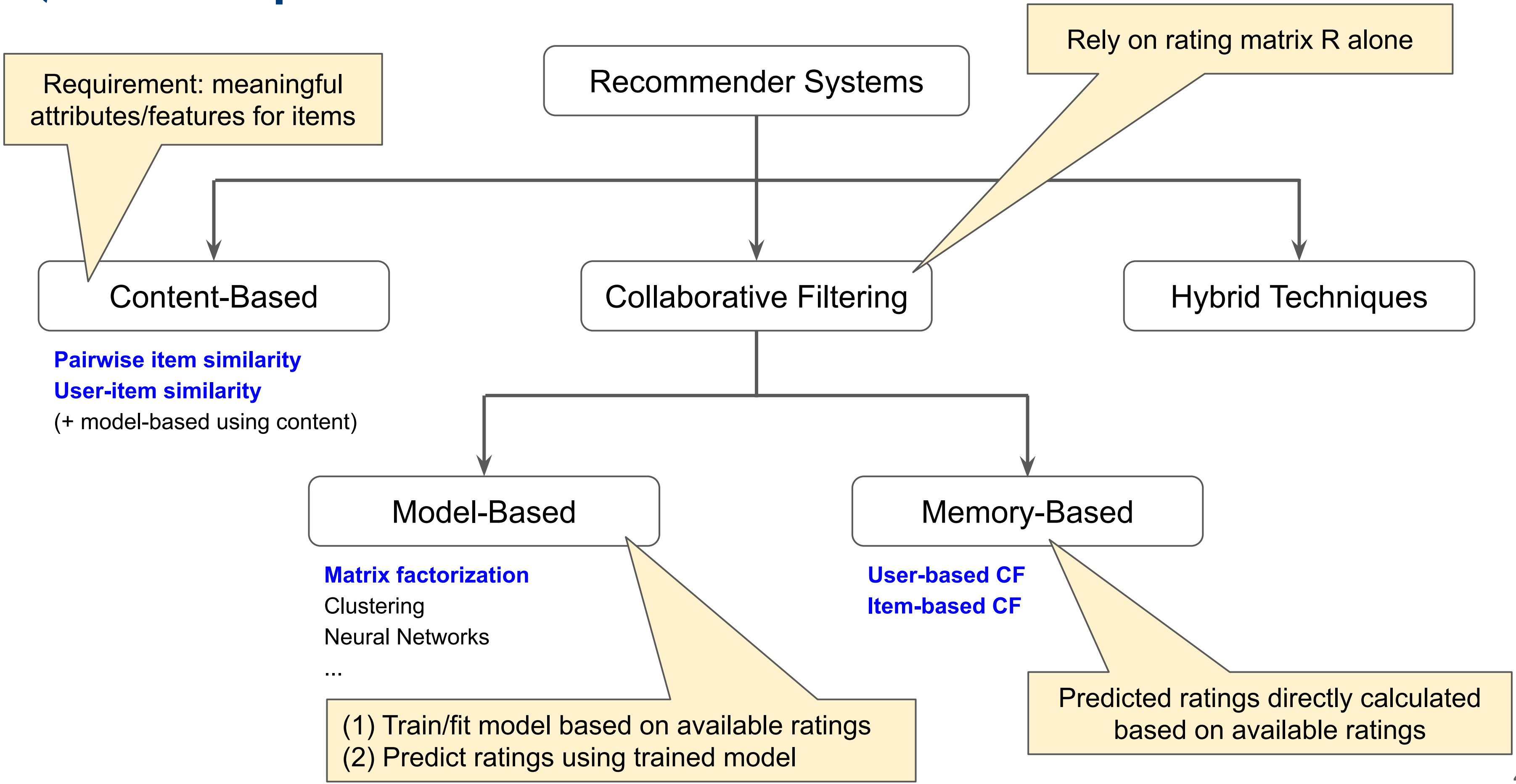
- Basic setup
  - Set of users *U*, set of items *V*
  - Rating matrix *R* with
    |*U*| rows and |*V*| columns
  - Matrix element R_{uv}: *u*'s rating of *v*
    (e.g., 1-5 starts, binary 0/1)
  - If available: information (features) about each item

items

users

	2	5		1			
		4		5	5		
		4	2				
2		3			2		3
					4		
5		5				2	
						4	3
3				1			
		2		2			
		5		1			4

Quick Recap


Requirement: meaningful attributes/features for items
Recommender Systems
Rely on rating matrix R alone
Content-Based
Collaborative Filtering
Hybrid Techniques
Pairwise item similarity
User-item similarity
(+ model-based using content)
Model-Based
Memory-Based
Matrix factorization
Clustering
Neural Networks
...
User-based CF
Item-based CF
(1) Train/fit model based on available ratings
(2) Predict ratings using trained model
Predicted ratings directly calculated based on available ratings

Outline

- **Graph Mining**
  - **Application Examples**
  - Basic definitions
- Community Detection
  - Basic definition & goals
  - Overview to different algorithms
- Centrality
  - Basic definition & goals
  - Overview to different algorithms
- Summary

Graphs are Everywhere

Transportation Networks

Nodes: MRT stations

Edges: Direct train connections between stations

Applications:

- Find shortest travels
- Find busy stations
- Plan bus routes

Source: https://www.lta.gov.sg/

Graphs are Everywhere

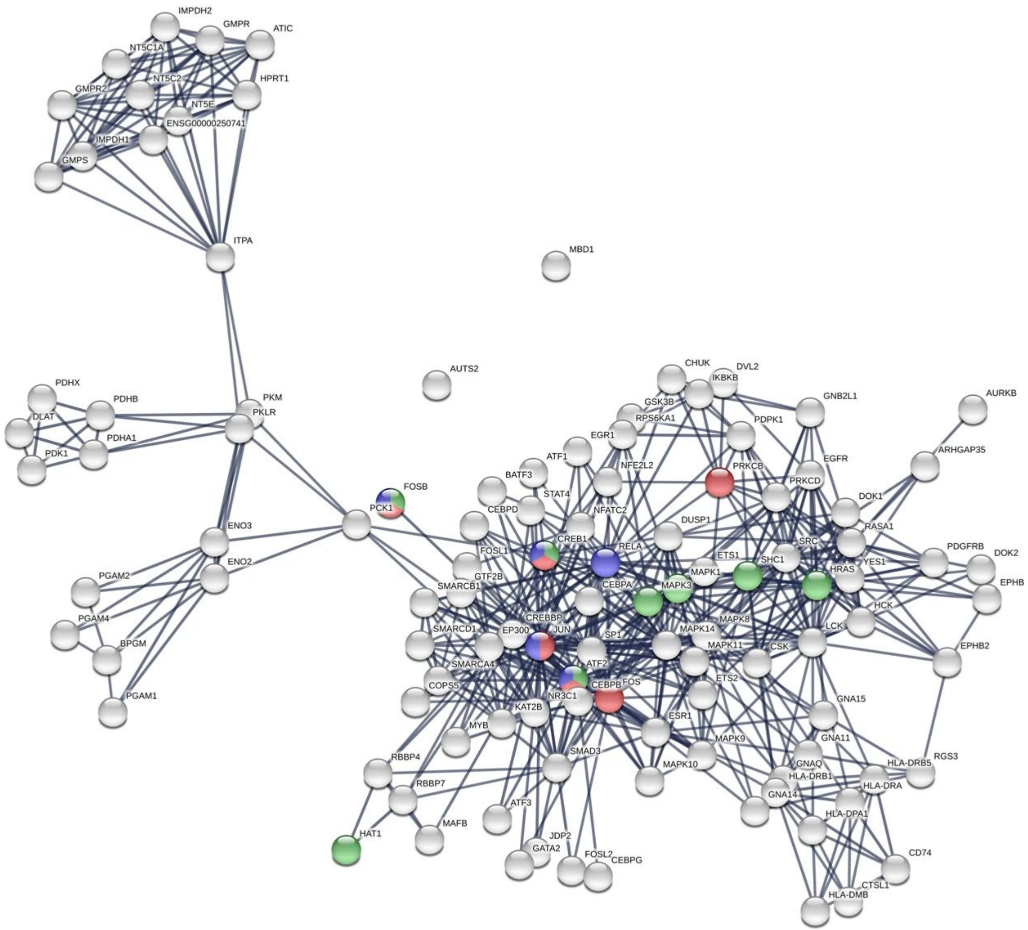
Protein-Protein Interactions

Nodes: Proteins

Edges: Physical interactions between two proteins

Applications:

- Understanding of diseases
  (disease prognosis, disease susceptibility)
- Drug discovery



Source: [Construction and Analysis of Protein-Protein Interaction Network of Heroin Use Disorder]

Graphs are Everywhere

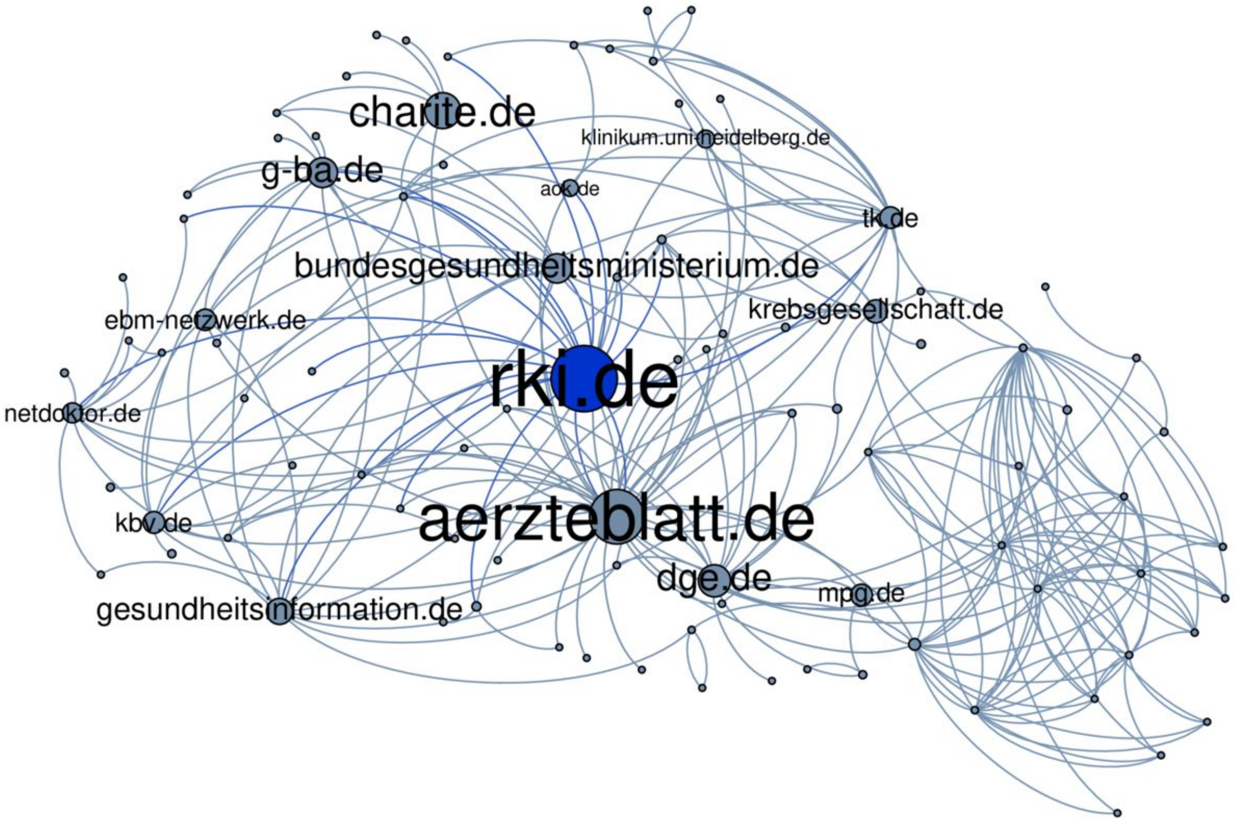
Web Graph

Nodes: Web pages

Edges: Hyperlinks

Applications:

- Identify authoritative sites
- Effective Web search



Paper: [Crawling the German Health Web: Exploratory Study and Graph Analysis]

Graphs are Everywhere

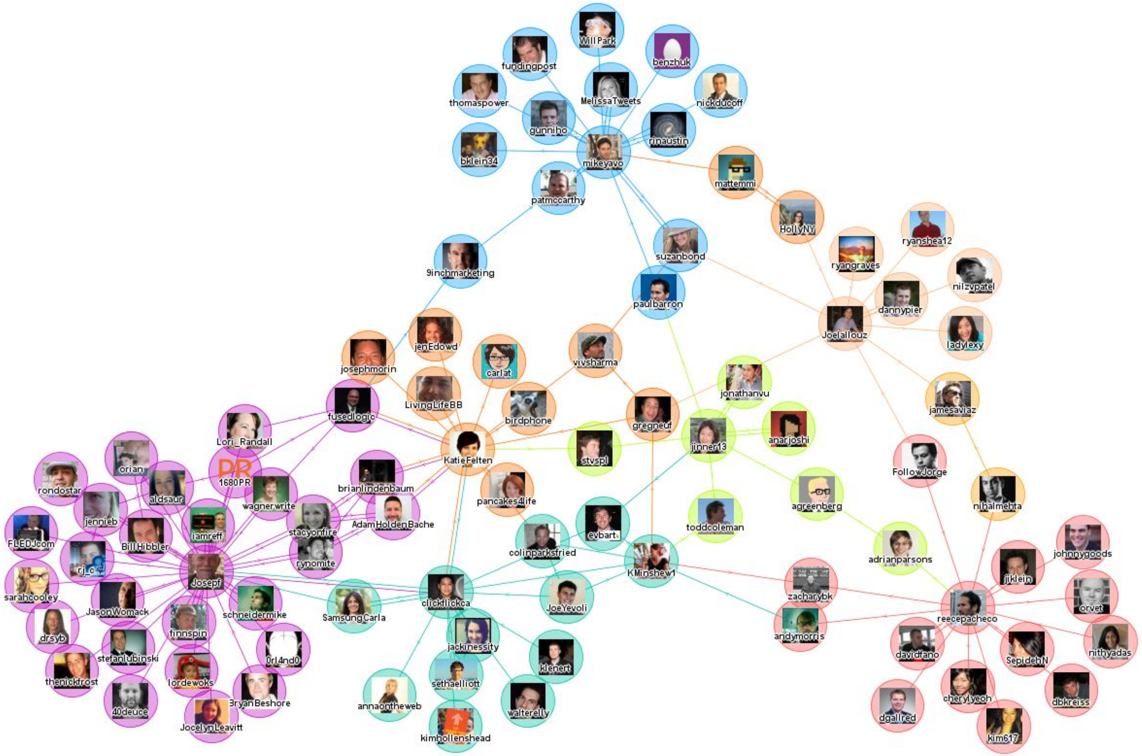
Social Networks

Nodes: People, users

Edges: friendship relationships

Applications:

- Identify communities
- Find influential people
- Friendship recommendations
- Effective Information diffusion
  (e.g., advertising, political campaigns)



Source: http://touchgraph.com/

Outline

- **Graph Mining**
  - Application Examples
  - **Basic definitions**
- Community Detection
  - Basic definition & goals
  - Overview to different algorithms
- Centrality
  - Basic definition & goals
  - Overview to different algorithms
- Summary

Graphs — Mathematical Definition

- Graph: Formalism for representing **relationships** between **items**

- A graph is a tuple *G = (V, E)*
  - Set of vertices (or nodes) *V*

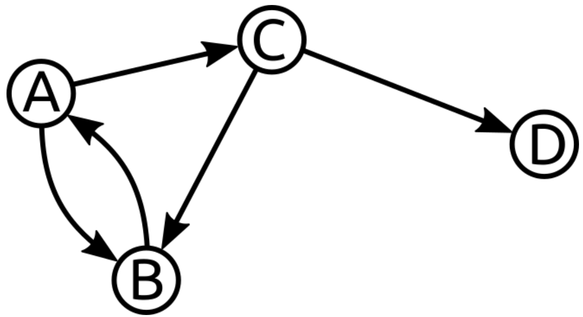
    $$V = \{v_1, v_2, ..., v_n\}$$

  - Set of edges E

    $$E = \{e_1, e_2, ..., e_m\}$$

    where an edge is a pair of vertices

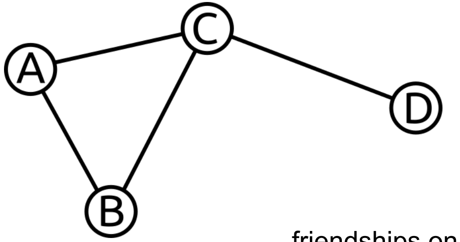
    $$e_i = (v_j, v_k)$$

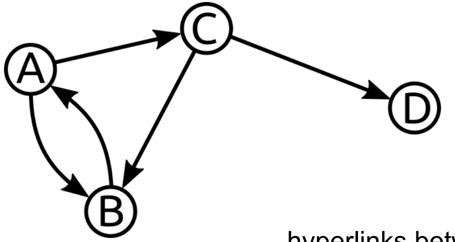


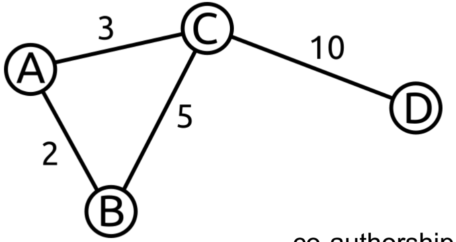
$$V = \{A, B, C, D\}$$

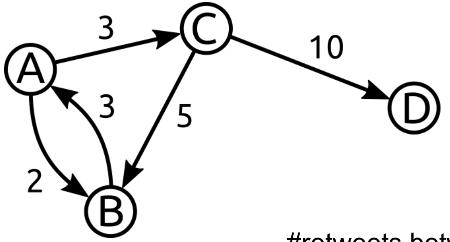
$$E = \{(A, B), (A, C), (C, D), (B, A), (C, B)\}$$

Types of Graphs

<table>
<tr><th></th><th>undirected</th><th>directed</th></tr>
<tr><td>unweighted</td><td>

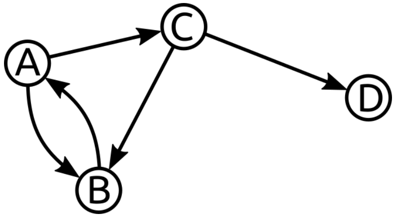
friendships on Facebook
directly connected MRT stations</td><td>

hyperlinks between web pages
followers on Twitter</td></tr>
<tr><td>weighted</td><td>

co-authorship with number of papers
transport network with travel times</td><td>

#retweets between Twitter users
borrower network with money owed</td></tr>
</table>

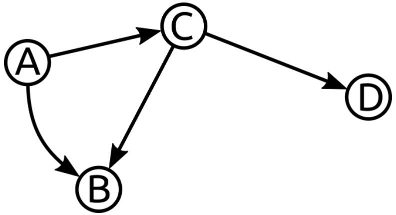
Types of Graphs

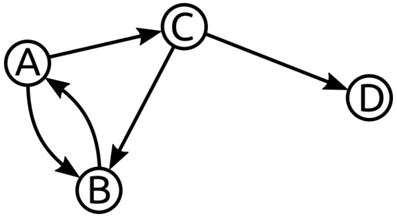


Cyclic Graph

vs.

Acyclic Graph

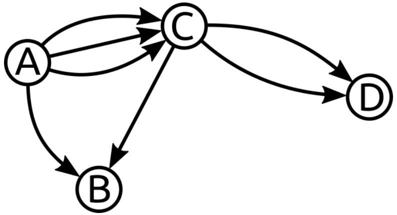


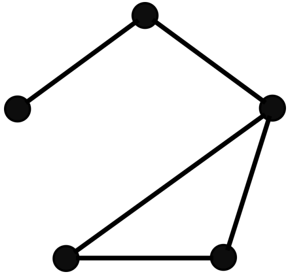


(Simple) Graph

vs.

Multigraph

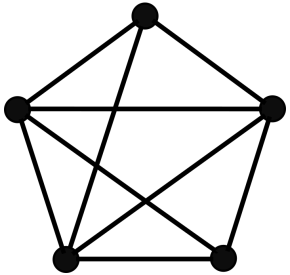




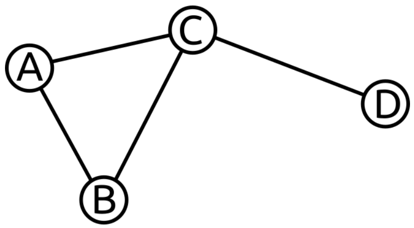
Sparse Graph

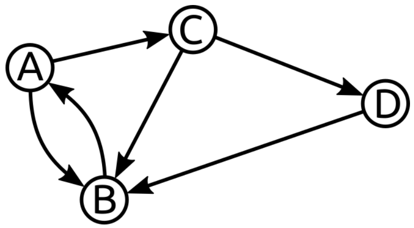
vs.

Dense Graph



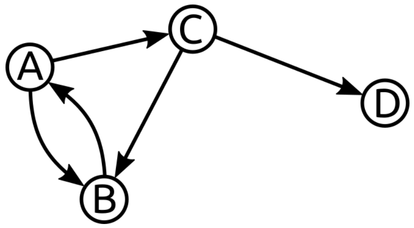
Types of Graphs





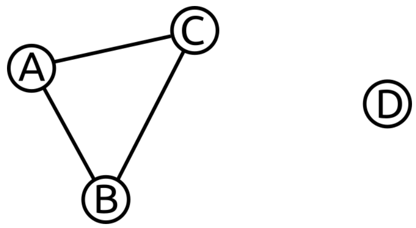
(Strongly) Connected Graph

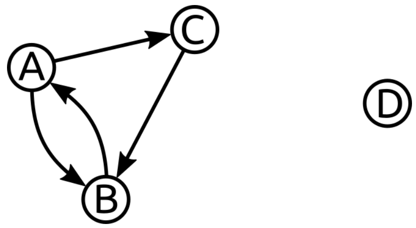
There exists a path from each node to every other node



Weakly Connected Graph

A directed graph where the underlying undirected graph is (strongly) connected

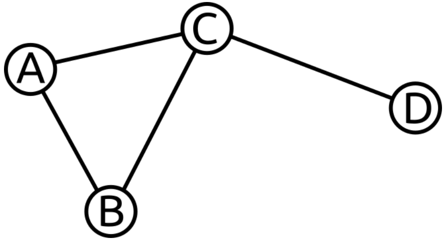




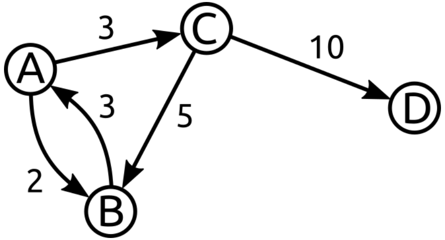
Disconnected Graph

A graph that is not connected

Representing Graphs — Adjacency Matrix



$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



$$A = \begin{bmatrix} 0 & 2 & 3 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 5 & 0 & 10 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Quick Quiz

When representing the **MRT** network as a graph G which **type of graph** will G definitely be?

A Connected

B Undirected

C Weighted

D Unweighted

Quick Quiz

Which type of graph is **not suitable** to represent the SBS bus network? Why?

A Sparse

B Undirected

C Weighted

D Unweighted

Overview

- Graph Mining
  - Application Examples
  - Basic definitions
- **Community Detection**
  - Basic definition & goals
  - Overview to different algorithms
- Centrality
  - Basic definition & goals
  - Overview to different algorithms
- Summary

Community Detection

- No formal definition of a community
  - From "Networks: An Introduction" (Mark Newman):

"Loosely stated, [community detection] is the problem of finding the natural divisions of a network into groups of vertices such that there are many edges within groups and few edges between groups. What exactly we mean by "many" or "few," however, is debatable, [...]"

- Wide range of applications
  - Identifying groups in social networks
  - Recommendation systems
  - Market segmentation
  - Outlier/anomaly detection

Doesn't that sound a bit like clustering?

YES!

Community Detection — Modularity

- Modularity $Q \in [-1/2, 1]$ of an undirected graph G with adjacency matrix A
  - Measures the relative density of edges inside communities with respect to edges outside communities
  - Optimizing modularity is NP-hard ➜ Practical algorithms based on heuristics

A[v,w] — weight of edge between nodes v and w

k_i — sum of the weights of edges attached to node i

c_i — community of a node i

$$Q = \frac{1}{2m} \sum_{vw} \left[A[v,w] - \frac{k_v k_w}{2m} \right] \underbrace{\delta(c_v, c_w)}$$

m — sum of weights of all edges

$$\delta(c_v, c_w) = \begin{cases} 1 & , c_v = c_w \\ 0 & , \text{otherwise} \end{cases}$$

Community Detection — Louvain Algorithm

- Method for the optimization of modularity

Initialization: Each node is a community

Repeat until no further change

Phase 1: **Modularity Optimization**

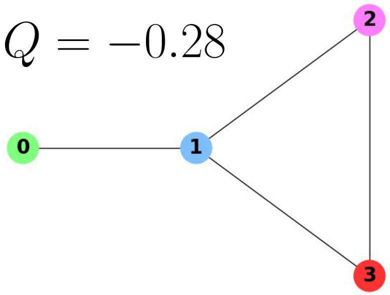
- For each node *v*, check if moving it to an adjacent community improves modularity
- Move *v* to community that maximizes modularity

Phase 2: **Graph Aggregation**

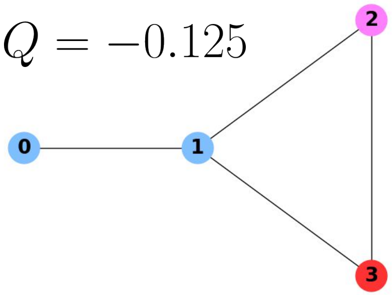
- Represent each community as a new node
- Update weights between new nodes

Louvain Algorithm — Modularity Optimization

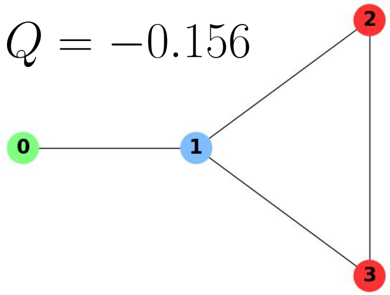
Input: Each node a community



Moving community 0 to community 1

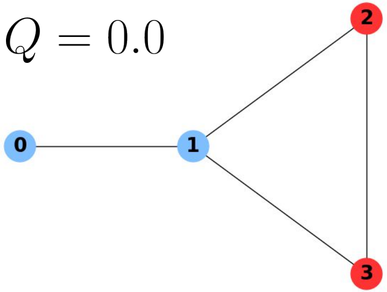


Moving community 2 to community 3



➜ Move community 0 to community 1 as it maximizes modularity

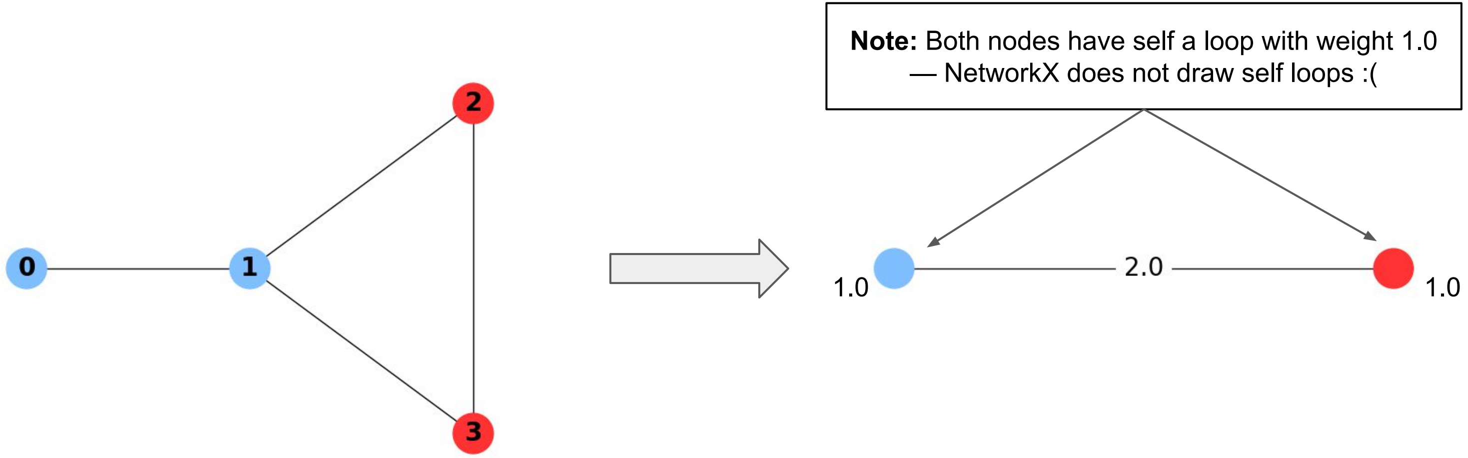
Output after all iterations



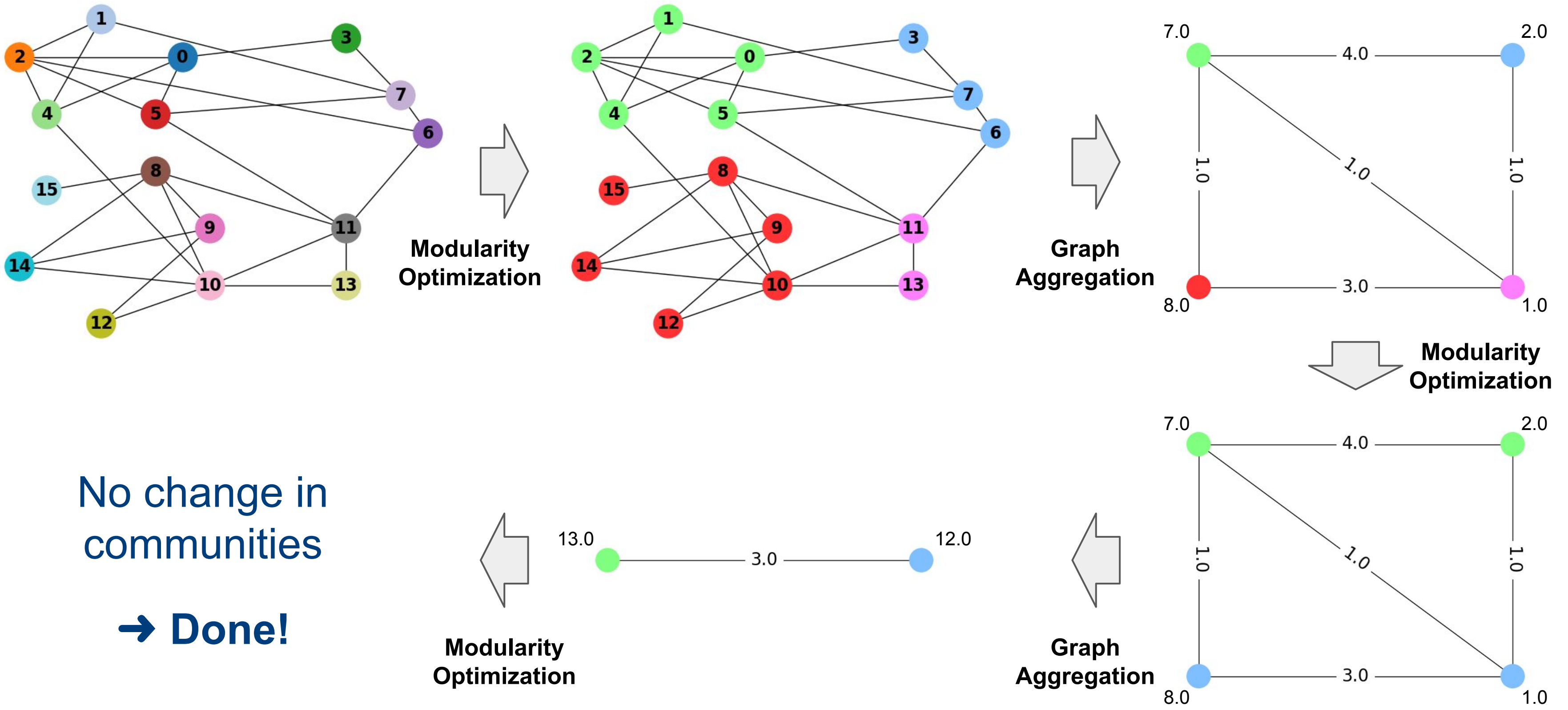
Note: Moving a node may not be permanent but can change again in later iteration before convergence.

Louvain Algorithm — Graph Aggregation

- 2 main steps
  - Merge all nodes of a community into a new single node
  - Aggregate edge weights accordingly



Louvain Algorithm — Full Example



No change in communities

➜ Done!

Louvain Algorithm — Remarks

- Heuristic
  - Optimizes modularity locally on all nodes
  - No guarantees for optimal modularity globally
    (in practice often superior to other methods)

- Performance optimization
  - Phase 1 (Modularity Optimization) requires to calculate change in modularity ΔQ
    (difference between modularity of G before and after moving communities)
  - Calculating ΔQ can be done based on local changes in community assignments
    (does not require the recalculate the modularity G after each change)

Full details in paper: [Fast Unfolding of Communities in Large Networks]() (Blondel et al., 2008)

Community Detection — Girvan-Newman Algorithm

- Divisive hierarchical approach
  - Start with whole graph representing a community
  - Iteratively remove edges until community is split into 2 sub-communities
    (continue splitting sub-communities recursively if needed)

- Criteria for removing edges: **Edge Betweenness Centrality**
  - Sum of fraction of all-pairs shortest paths that pass through an edge e

number of shortest paths from v to w going through edge e

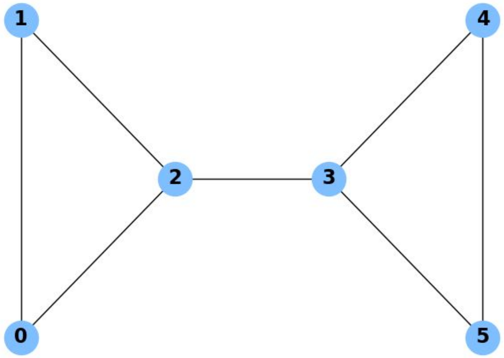
$$c_B(e) = \sum_{v,w \in V} \frac{\sigma(v, w|e)}{\sigma(v, w)}$$

number of shortest paths from v to w

Full details in paper: [On Variants of Shortest-Path Betweenness Centrality and their Generic Computation]() (Brandes et al., 2008)

Girvan-Newman Algorithm — Edge Betweenness Centrality

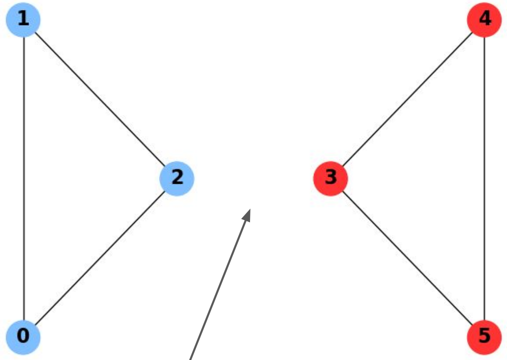
Input community



Betweenness Centrality
for all edges

e	$c_B(e)$
(0, 1)	0.067
(0, 2)	0.267
(1, 2)	0.267
(2, 3)	**0.600**
(3, 4)	0.267
(3, 5)	0.267
(4, 5)	0.067

Output communities



Remove edge e with max. $c_B(e)$

Continue recursively until community is plit (not needed in this simple example)

Girvan-Newman Algorithm

- Algorithm splits a graph G(V, E) into 2 disconnected components

Repeat

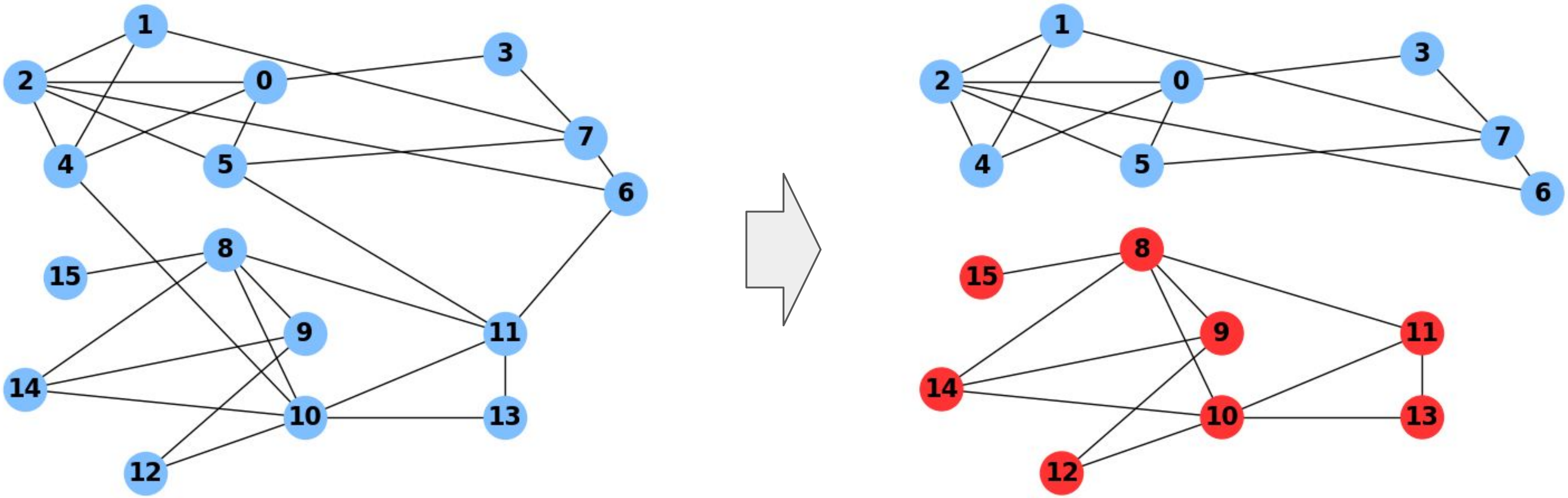
Calculate $c_B(e)$ for all $e \in E$

Remove edge from with max. $c_B(e)$

Until G is split into 2 components

- Recursive step
  - Apply algorithm to each new component
  - Stops if a component contains only single node
    (or early stop based on user specifications)

Girvan-Newman Algorithm — Full Example



Girvan-Newman Algorithm — Remarks

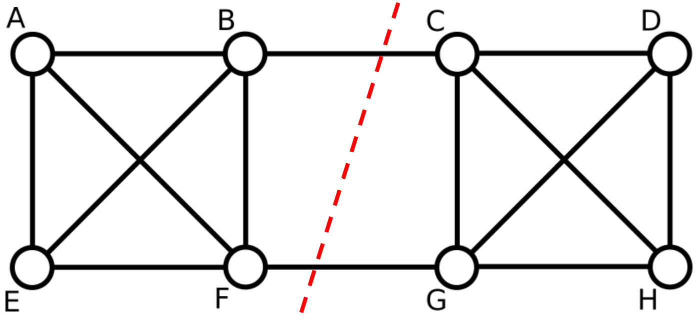
- Complexity Analysis
  - Core concept of algorithm: Edge Betweenness Centrality
  - Requires to solve the **All-Pairs Shortest Path (APSP)** problem
  - Various algorithms and complexities depending on type of graph
    (directed vs. undirected, cyclic vs. acyclic, with or without negative weights, etc.)

Time Complexity
n^3
$n^3(loglogn)/logn^{1/3}$
$n^3(loglogn/logn)^{1/2}$
$n^3/(logn)^{1/2}$
$n^3(loglogn/logn)^{5/7}$
$n^3loglogn/logn$
$n^3(loglogn)^{1/2}/logn$
$n^3/logn)$
$n^3(loglogn/logn)^{5/4}$
$n^3(loglogn)^3/(logn)^2$
$n^3(loglogn)/(logn)^2$

Table taken from: [A Survey of Shortest-Path Algorithms](#)
(Madkour et al., 2017) — $n = |V|$, number of nodes

Karger's Algorithm for Min-Cut

- Min-Cut Problem
  - Given a graph G, cut G into 2 components such that the number of edges between both components is minimal



➜ |Min-Cut| = 2

  - Fundamental problem in graph theory ➜ many existing algorithms
    (with varying focus and support for different graph types — e.g., directed vs. undirected)

Karger's Algorithm for Min-Cut

- Karger's algorithm
  - **Randomized** method to find Min-Cut
  - Applicable to undirected graphs with positive weights
    (this includes unweighted graphs where the weight can be considered 1)

While |V| > 2

Randomly pick a remaining edge *e = (v, u)*

Merge/contract *v* and *u* into a new node

- Update edges to neighbors of v and u
- Remove self-loops

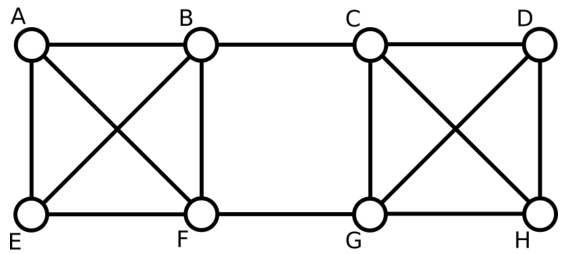
Return edges between the final 2 nodes as Min-Cut

Intuition: Edges that are in the Min-Cut have a lower probability to get picked!

Basic runtime: $\mathbf{O(|V|^2)}$

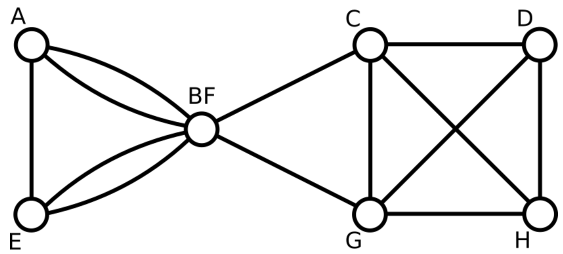
(but further optimizations exists)

Karger's Algorithm for Min-Cut — Example



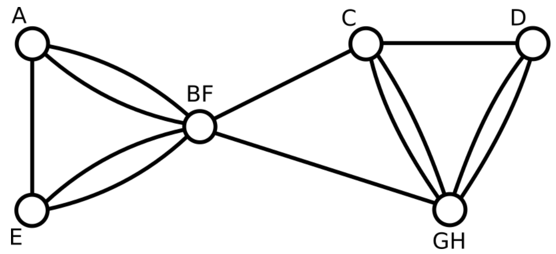
Number of edges: 14

Pick *e* = (*B*, *F*) with *P*(*e*) = 1/14



Number of edges: 13

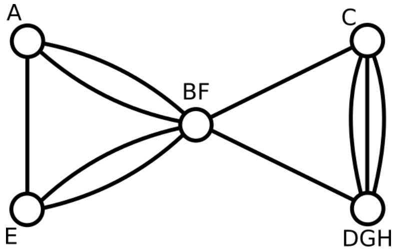
Pick *e* = (*G*, *H*) with *P*(*e*) = 1/13



Number of edges: 12

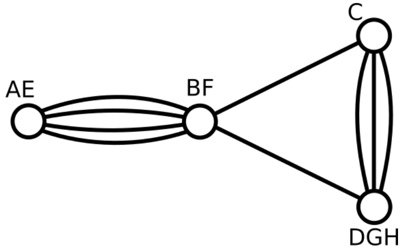
Pick *e* = (*D*, *GH*) with *P*(*e*) = 1/6

Karger's Algorithm for Min-Cut — Example



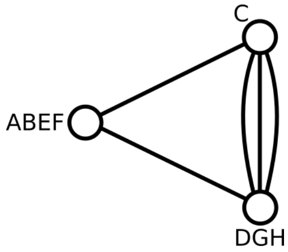
Number of edges: 10

Pick e = (A, E) with $P(e)$ = 1/10



Number of edges: 9

Pick e = (AE, BF) with $P(e)$ = 4/9



Number of edges: 5

Pick e = (C, DGH) with $P(e)$ = 3/5

2 node left ➜ Done, with |**Min-Cut**| = **2**

Karger's Algorithm for Min-Cut — Analysis

- What is the probability that the algorithm finds the "correct" Min-Cut?

- For an undirected graph $G = (V, E)$, with $n = |V|$ and $m = |E|$

The average degree of a node is $\frac{1}{n}\sum_{v\in V} degree(v) = \frac{2m}{n}$

The size of the Min-Cut is limited to $|Min\text{-}Cut| \leq \frac{2m}{n}$ since $\forall v \in V : |Min\text{-}Cut| \leq degree(v)$

➜ $P(\text{randomly selected edge is in Min-Cut}) \leq \frac{\frac{2m}{n}}{m} = \frac{2}{n}$

Karger's Algorithm for Min-Cut — Analysis

- Let P(success) = P(final cut is Min-Cut)

$$\begin{aligned} P(\text{success}) = \; & P(\text{1st selected edge not in Min-Cut}) \times \\ & P(\text{2nd selected edge not in Min-Cut}) \times \ldots \end{aligned}$$

$$\begin{aligned} P(\text{success}) &\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right)\cdots\left(1 - \frac{2}{4}\right)\left(1 - \frac{2}{3}\right) \\ &= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-3} \cdot \ldots \cdot \frac{2}{4} \cdot \frac{1}{3} \\ &= \frac{2}{n(n-1)} = \binom{n}{2}^{-1} \end{aligned}$$

That's a rather low probability :(

➜ Run algorithm multiple times!

But how often?

Karger's Algorithm for Min-Cut — Analysis

- If the algorithm is run k times and take the smallest cut found — What is the probability P(failure) that it is not a Min-Cut?

$$P(\text{failure}) = \left[1 - \binom{n}{2}^{-1}\right]^k$$

Note: For any x ≥ 1:

$$\frac{1}{4} \leq \left(1 - \frac{1}{x}\right)^{cx} \leq \left(\frac{1}{e}\right)^c$$

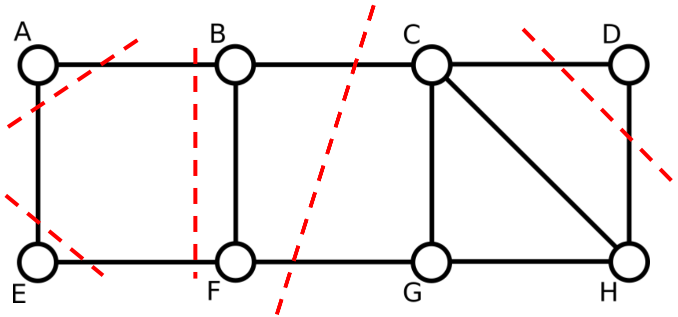
$$\text{with } k = \binom{n}{2} \ln n \quad \rightarrow \quad P(\text{failure}) \leq \left(\frac{1}{e}\right)^{\ln n} = \frac{1}{n}$$

With $k \in O(n^2 \log n)$ ➜ Total runtime of Karger's Algorithm is in $\mathbf{O(n^4 \log n)}$

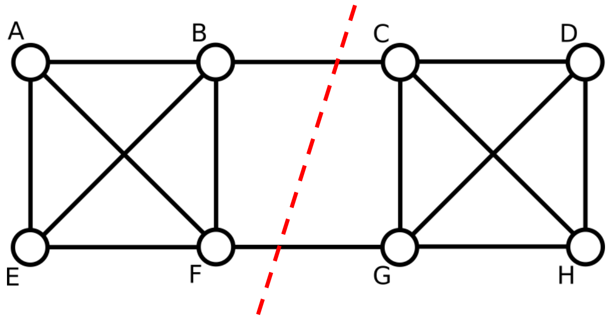
(with a Min-Cut with a high probability)

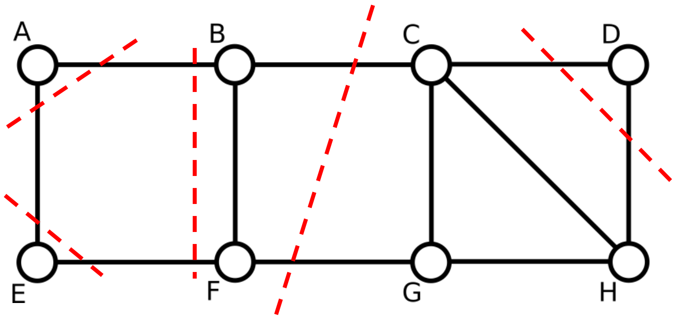
Karger's Algorithm for Min-Cut — Remarks

- In general, a graph has multiple possible Min-Cuts



- Choice of Min-Cut often application-specific, e.g.,
  - Favor Min-Cuts where the 2 components are of similar size (e.g., similar number of nodes)
  - Ignore Min-Cuts where the size of a component is below a threshold






```
|Min-Cut| = 5 -- Components: A,B ### C,D,E,F,G,H
|Min-Cut| = 4 -- Components: A,C,D,E,F,G,H ### B
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,G ### H
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,G ### H
|Min-Cut| = 5 -- Components: A,B,C,D,E,F ### G,H
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,G ### H
|Min-Cut| = 4 -- Components: A,B,D,E,F,G,H ### C
|Min-Cut| = 5 -- Components: A,F ### B,C,D,E,G,H
|Min-Cut| = 4 -- Components: A,B,C,E,F,G ### D,H
|Min-Cut| = 5 -- Components: A,B,C,D,E,F ### G,H
|Min-Cut| = 4 -- Components: A,B,C,E,F ### D,G,H
|Min-Cut| = 3 -- Components: A,B,C,E,F,G,H ### D
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,G ### H
|Min-Cut| = 4 -- Components: A,B,C,E,F,G ### D,H
|Min-Cut| = 4 -- Components: A,B,C,D,E,G,H ### F
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
```

```
|Min-Cut| = 2 -- Components: A,B,C,E,F,G,H ### D
|Min-Cut| = 2 -- Components: A,B,C,E,F,G,H ### D
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
|Min-Cut| = 2 -- Components: A,B,C,E,F,G,H ### D
|Min-Cut| = 2 -- Components: A ### B,C,D,E,F,G,H
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,G ### H
|Min-Cut| = 2 -- Components: A,E ### B,C,D,F,G,H
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,H ### G
|Min-Cut| = 3 -- Components: A,B,C,E,F,G ### D,H
|Min-Cut| = 2 -- Components: A,B,C,E,F,G,H ### D
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,H ### G
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,G ### H
|Min-Cut| = 2 -- Components: A ### B,C,D,E,F,G,H
|Min-Cut| = 2 -- Components: A,B,C,E,F,G,H ### D
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
|Min-Cut| = 2 -- Components: A,B,C,D,F,G,H ### E
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
|Min-Cut| = 2 -- Components: A,B,C,E,F,G,H ### D
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
```

Overview

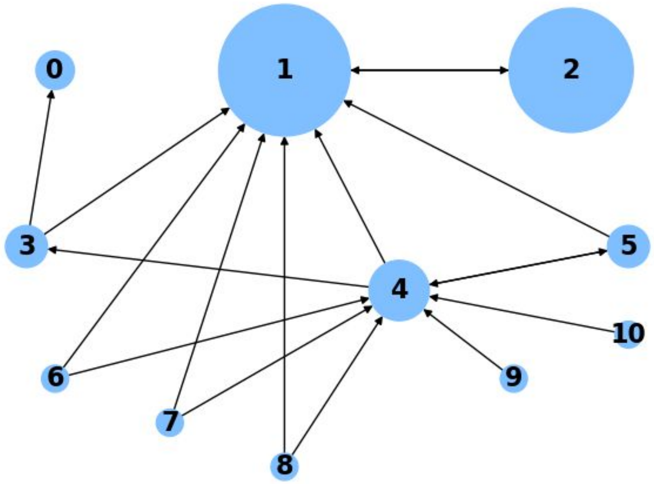
- Graph Mining
  - Application Examples
  - Basic definitions
- Community Detection
  - Basic definition & goals
  - Overview to different algorithms
- **Centrality**
  - Basic definition & goals
  - Overview to different algorithms
- Summary

Centrality

- Centrality — Centrality measures
  - Quantify the **importance** of a node given its topological position in a graph
    (centrality is commonly assigned to nodes but can be extended to edges, cf. Edge Betweenness Centrality)
  - Different measures favor different "flavours" of importance

➜ What makes a node important?
(or more important compared to other nodes)

Example: PageRank of 10 web pages



- Wide range of applications
  - Identify influential social network users
  - Identify superspreader of diseases
  - Identify key infrastructure nodes in infrastructure networks

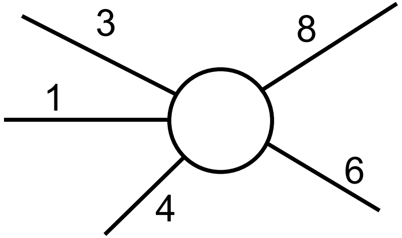
Degree Centrality

- Centrality of a node only dependent on direct neighborhood edges

Undirected graph

Sum of weights of connected edges

$$c_d(v_i) = \sum_{v_j \in V} A[i,j]$$



$$c_d(v) = 22$$

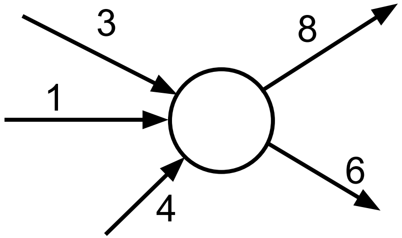
Directed graph

Sum of weights of incoming edges

$$c_{d\_in}(v_i) = \sum_{v_j \in V} A[j,i]$$

Sum of weights of outgoing edges

$$c_{d\_out}(v_i) = \sum_{v_j \in V} A[i,j]$$



$$c_{d\_in}(v) = 8$$

$$c_{d\_out}(v) = 14$$

Note: For unweighted graphs, A[i,j] = 1 for existing edges ➜ sum of weights = edge count

Degree Centrality — Pros & Cons

- Pros
  - Simple measure ➜ easy and fast to calculate
  - For many applications "good enough"

- Cons
  - Local measure — does not take any extended topological information of a node into account
  - Treats all connected edges of a node equally
    (i.e., neighboring source or target node does not matter)
  - Depending on application, easy to manipulate

Examples of manipulation attacks

- Create fake pages with links to a target page to bump up its search result rank
- Create fake followers on social media to increase reputation and attract sponsors
- Create fake feedback to create on e-commerce sites to increase reputation

Eigenvector Centrality

Common Eigenvector equation:

$$A\vec{x} = \lambda\vec{x}$$

All $\vec{x}$ solving this equation are the eigenvectors of A and λ are their corresponding eigenvalues:

$$(A - \lambda I)\,\vec{x} = 0$$

- Idea: Centrality of a node depends on the centrality of its neighbors
  - Basic definition applicable to undirected graph
  - Recursive definition — How to calculate it? (well, it's in the name)

$$c_{ev}(v_i) = \frac{1}{\lambda} \sum_{v_j \in V} \left[A[i,j] \cdot c_{ev}(v_j)\right]$$

λ — some normalization constant

In matrix form: $\lambda c_{ev} = A c_{ev}$ ➜

c_{ev} is the largest eigenvector of adjacency matrix A!

c_{ev} — vector of centrality degrees of all nodes

Eigenvector Centrality — Power Iteration / Power Method

- Power Iteration — numerical method to find largest eigenvector of a matrix
  - Solving eigenvector equation analytically not tractable for large matrices

Input: matrix M, error threshold ε, #max. iterations T

Initialization: $t = 0$, $x_0 = [1/|V|, 1/|V|, \ldots, 1|V|]$

Repeat

$t = t+1$

$x_t = Ax_{t-1}$

$x_t = x_t / ||x_t||$ # Normalize vector

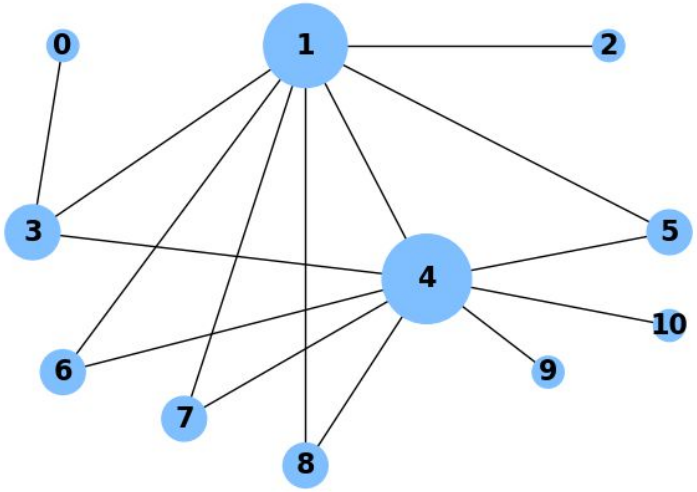
$\delta = ||x_t - x_{t-1}||$ # Calculate difference

Until $\delta < \varepsilon$ **or** $t > T$

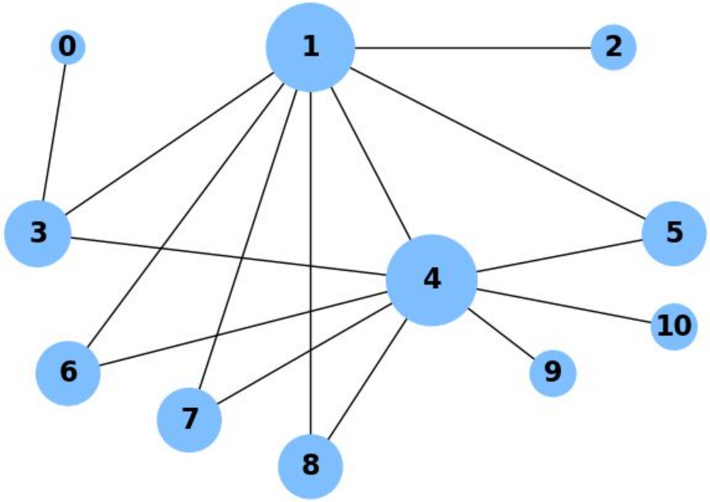
Return x_t

Degree vs. Eigenvector Centrality (size of nodes reflect centrality scores)

Degree



Eigenvector



- Low-degree nodes benefit from connections to high-degree nodes

PageRank

- Goal: Find important pages in the web graph
  - The set of vertices V is set of all (indexed) web pages
  - An edge $e_{ij} \in E$ indicates that there is a hyperlink for page *i* to page *j*

- PageRank — a page v_j has a high PageRank if
  - Many other pages link to v_j
  - Those pages linking to v_j have

    (a) a high PageRank themselves

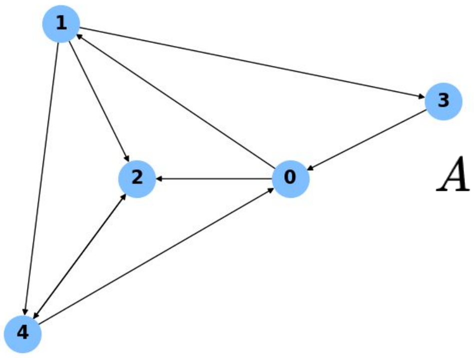
    (b) not many other outgoing links

$$PR(v_j) = \sum_{i \to j} \frac{1}{outdegree(i)} PR(v_i)$$

Matrix notation for all vertices

$$PR(v) = M \cdot PR(v)$$

PageRank — Transition Matrix M



Adjacency matrix A

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

- Transpose A
- Normalize columns
  (column entries sum up to 1)

Transition matrix M

$$M = \begin{bmatrix} 0 & 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{3} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 1 & 0 & 0 \end{bmatrix}$$

Note: M is a so-called column-stochastic matrix.

Computing PageRank Scores

- PageRank is an Eigenvector problem
  - Solvable with Power Iteration Method

Common Eigenvector equation:

$$A\vec{x} = \lambda\vec{x}$$

All solving this equation are the eigenvectors of A and are their corresponding eigenvalues:

$\vec{x}$ λ $(A - \lambda I)\,\vec{x} = 0$

$$\lambda PR(v) = M \cdot PR(v)$$

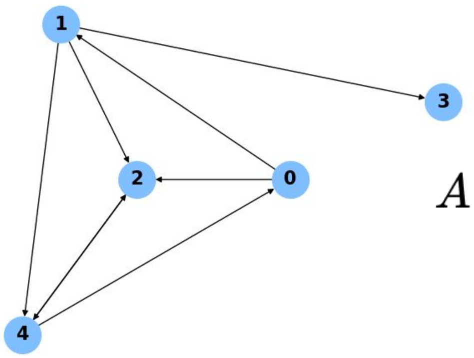
$$\Updownarrow$$

Since M is column-stochastic, the largest eigenvalue λ = 1.

$$PR(v) = M \cdot PR(v)$$

Computing PageRank Scores — Challenge

- Requirement: Graph must be strongly connected
  - Each node can be reached from any other node
  - Requirement does not hold for the web graph ➜ dangling nodes
  - Computing PageRank scores using Power Iteration Method not working



Adjacency matrix A

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ \color{red}{0} & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

- Transpose A
- Normalize columns
  (**column entries sum up to 1?**)

Transition matrix M

$$M = \begin{bmatrix} 0 & 0 & 0 & \color{red}{0} & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & \color{red}{0} & 0 \\ \frac{1}{2} & \frac{1}{3} & 0 & \color{red}{0} & \frac{1}{2} \\ 0 & \frac{1}{3} & 0 & \color{red}{0} & 0 \\ 0 & \frac{1}{3} & 1 & \color{red}{0} & 0 \end{bmatrix}$$

Computing PageRank Scores — Handling Dangling Nodes

- PageRank implements **Random Surfer** model
  - α probability of following a link on a page to the next
  - $(1-\alpha)$ probability of jumping to a random page (i.e., not following a link)

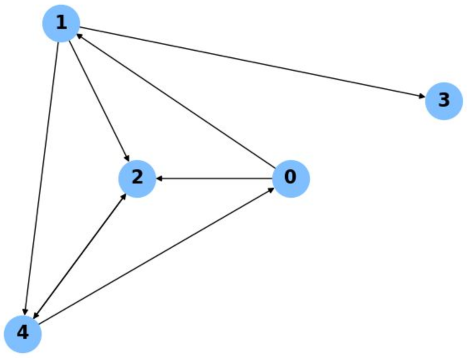
➜ **Any page can be reached, whether linked or not!**

$$PR(v) = M \cdot PR(v) \quad \Longrightarrow \quad PR(v) = \alpha \cdot M \cdot PR(v) + (1-\alpha) \cdot E$$

$$\text{with } E = \begin{bmatrix} 1/|V| \\ 1/|V| \\ \vdots \\ 1/|V| \end{bmatrix}$$

Random Surfer Model — Effects on Transition Matrix

- Random Surfer model introduces "virtual edges"
  - Each node can be reached from any other ➜ virtual links ➜ fully connected graph
  - PageRank scores can be calculated using Power Iteration Method without problems



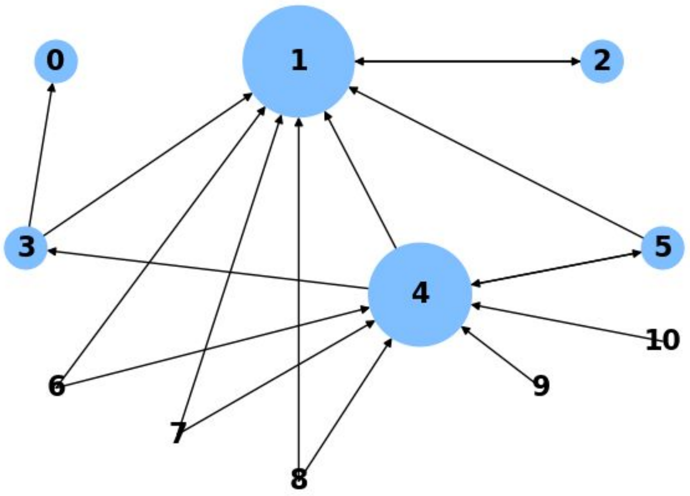
Implied* transition matrix M

$$M = \begin{bmatrix} \frac{1-\alpha}{5} & \frac{1-\alpha}{5} & \frac{1-\alpha}{5} & \frac{1}{5} & \frac{1}{2}\alpha + \frac{1-\alpha}{5} \\ \frac{1}{2}\alpha + \frac{1-\alpha}{5} & \frac{1-\alpha}{5} & \frac{1-\alpha}{5} & \frac{1}{5} & \frac{1-\alpha}{5} \\ \frac{1}{2}\alpha + \frac{1-\alpha}{5} & \frac{1}{3}\alpha + \frac{1-\alpha}{5} & \frac{1-\alpha}{5} & \frac{1}{5} & \frac{1}{2}\alpha + \frac{1-\alpha}{5} \\ \frac{1-\alpha}{5} & \frac{1}{3}\alpha + \frac{1-\alpha}{5} & \frac{1-\alpha}{5} & \frac{1}{5} & \frac{1-\alpha}{5} \\ \frac{1-\alpha}{5} & \frac{1}{3}\alpha + \frac{1-\alpha}{5} & 1\alpha + \frac{1-\alpha}{5} & \frac{1}{5} & \frac{1-\alpha}{5} \end{bmatrix}$$

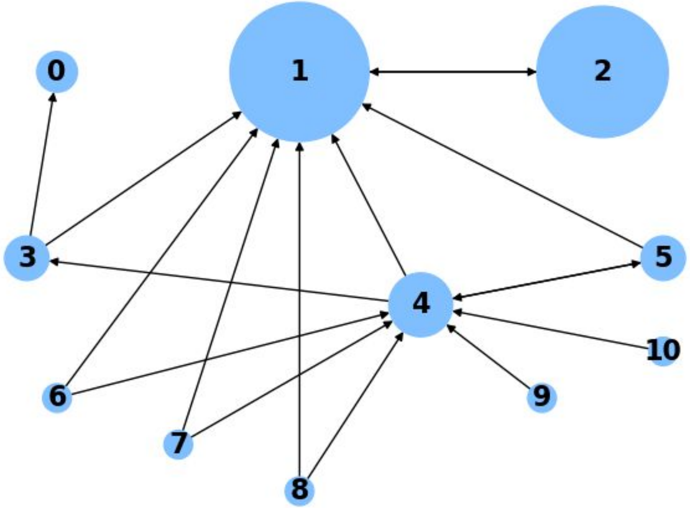
***Note:** Matrix M is not materialized, but this is the matrix that reflects the PageRank formula!

InDegree vs. PageRank (size of nodes reflect centrality scores)

InDegree



PageRank



- Nodes 1 and 4 very similar centrality scores
- Node 2 with only a low centrality score
- Scores of 0 for nodes with no incoming edges

- Clear difference between Nodes 1 and 4
- Node 2 with a high centrality score since linked to from Node 1 with a very high score
- Non-zero scores for nodes with no incoming edges

Quick Quiz

Given a Graph with 10 nodes and a setting of α=0.85, what is the **lowest possible** PageRank score?

$$c_{pr}(v_i) = \alpha M c_{pr}(v_i) + (1 - \alpha)E$$

A 0.0

B 0.015

C 0.085

D 0.15

Eigenvector-Based Measures — Remarks

- Measuring centrality by solving an Eigenvector problem
  - Recursive definition of centrality very intuitive
  - Many other similar measures (e.g., HITS, SALSA, Katz)
  - Many application-specific extensions to basic measures (e.g., personalization of PageRank where random jumps are no longer uniform)
  - More complex calculation than for local measures but calculation of largest Eigenvectors very scalable through parallelization

Closeness Centrality

- Intuition: A node *t* is central if the distance to all other nodes is small
  - Small distance to node *t* = short paths from all other nodes to *t*
  - For directed paths, the closeness of node *t* can differ greatly when considering incoming or outcoming edges for calculating distances
  - Basic definition applicable to unweighted graph
    (generalized definitions for weighted graphs have been proposed)

N — number of nodes from which v is reachable

$$c_{cl}(v) = \frac{N - 1}{\sum_{w \in V} \underbrace{d(w, v)}}$$

d(*w,v*) — length of shortest path from *w* to *v*

Note: For directed graphs, this definition calculates closeness using the nodes' incoming edges — more common case.
To consider outgoing edges, *d*(*w,v*) becomes *d*(*v,w*), and N becomes the number of nodes reachable from *v*.

Betweenness Centrality

- Intuition: A node *t* is central if many shortest paths between all other nodes pass through node t
  - Removing such nodes would cause the most "disruption" in a graph
  - Directly applicable to directed/undirected and weighted/unweighted graphs
    (since the the notion of shortest path is well-defined for all graph types)

$$c_b(v) = \sum_{s,t \in V; s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

number of shortest paths from *s* to *t* passing through node *v*

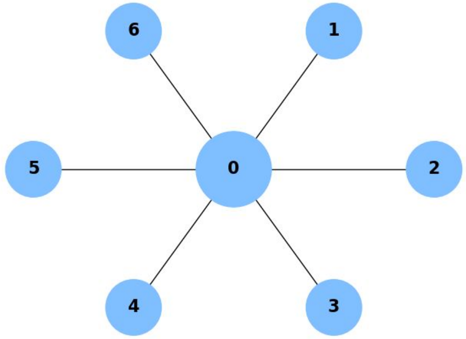
total number of shortest paths from *s* to *t*

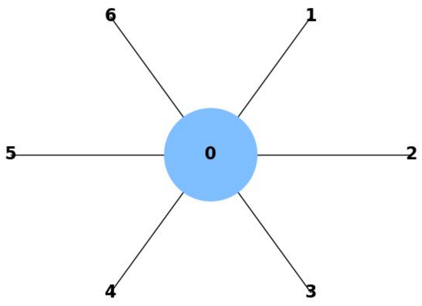
Closeness vs. Betweenness (size of nodes reflect centrality scores)

Closeness

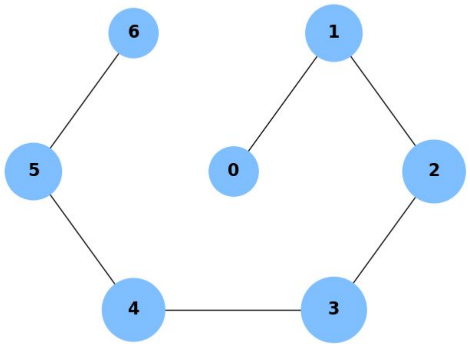
Betweenness

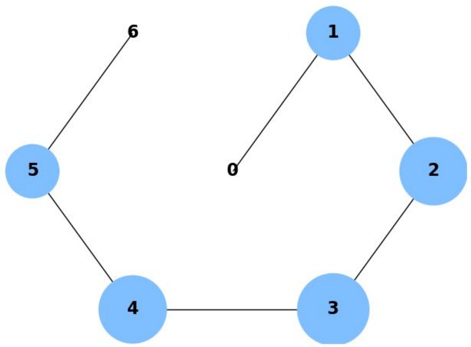
Star network graph





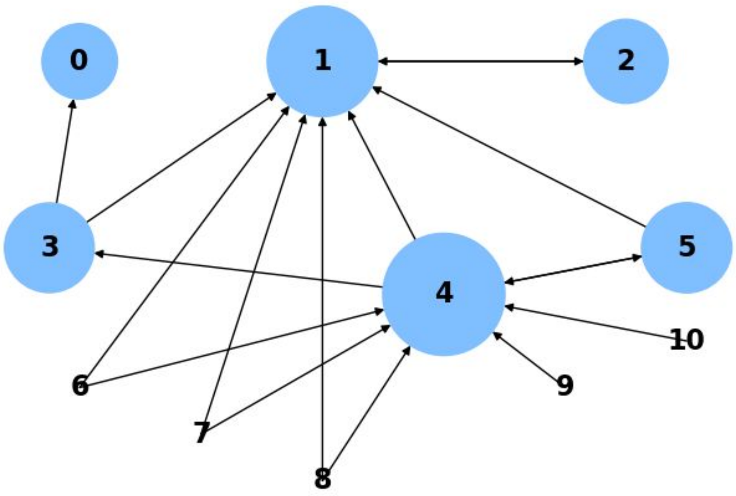
Sequence graph



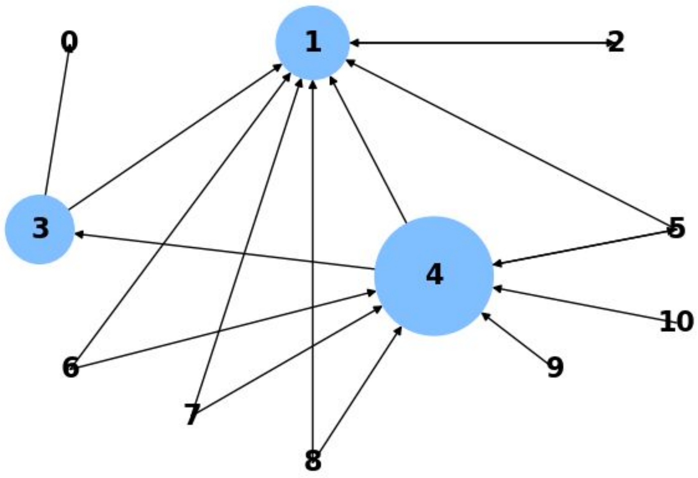


Closeness vs. Betweenness (size of nodes reflect centrality scores)

Closeness



Betweenness



Closeness & Betweenness — Remarks

- Distance-based measures
  - Both measures rely on the notion of shortest paths between nodes
  - Requires to solve the **All-Pairs Shortest Path (APSP)** problem
  - Various algorithms and complexities depending on type of graph
    (directed vs. undirected, cyclic vs. acyclic, with or without negative weights, etc.)

Time Complexity
n^3
$n^3(loglogn)/logn^{1/3}$
$n^3(loglogn/logn)^{1/2}$
$n^3/(logn)^{1/2}$
$n^3(loglogn/logn)^{5/7}$
$n^3loglogn/logn$
$n^3(loglogn)^{1/2}/logn$
$n^3/logn)$
$n^3(loglogn/logn)^{5/4}$
$n^3(loglogn)^3/(logn)^2$
$n^3(loglogn)/(logn)^2$

Table taken from: A Survey of Shortest-Path Algorithms
(Madkour et al., 2017) — n = $|V|$, number of nodes

Centrality — Discussion

- Centrality = importance of nodes on a graph
  - Important concept of graph mining with many applications
  - Wide range of proposed measures that differ in their definition of a node's importance
  - Not all measures are applicable (or suitable) to all types of graphs

- Overview to a selected set of popular measures
  - Local measures — Degree, InDegree, OutDegree
  - Eigenvector-based measures — Eigenvector Centrality, PageRank
  - Distance-based measures — Closeness, Betweenness

Quick Quiz

Given an **undirected** and **connected** graph G, which centrality measure can yield **scores of 0**?

A Eigenvector

B Closeness

C Degree

D Betweenness

Quick Quiz

Which MRT station has the highest **Closeness centrality** score?

(graph does not include LRT stations)

A Dhoby Ghaut

B Buona Vista

C City Hall

D Stevens

Quick Quiz

Which MRT station has the highest **Betweenness centrality** score?

(graph does not include LRT stations)

A Bugis

B Botanic Gardens

C Clementi

D Outram Park

Solutions to Quick Quizzes

- Slide 16: A
- Slide 17: B
- Slide 54: B
- Slide 62: D
- Slide 63: A
- Slide 64: B