

CS5228: Knowledge Discovery and Data Mining

Lecture 9 — Graph Mining

Course Logistics



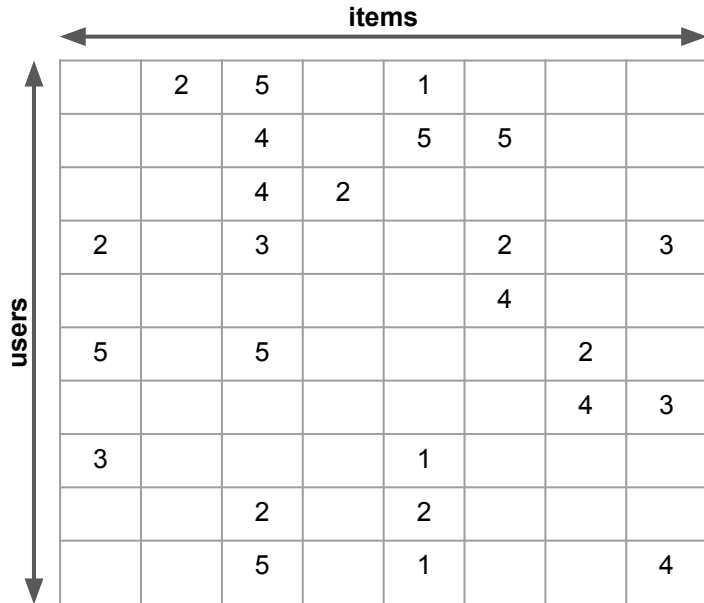
Quick Recap

- Recommender Systems

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

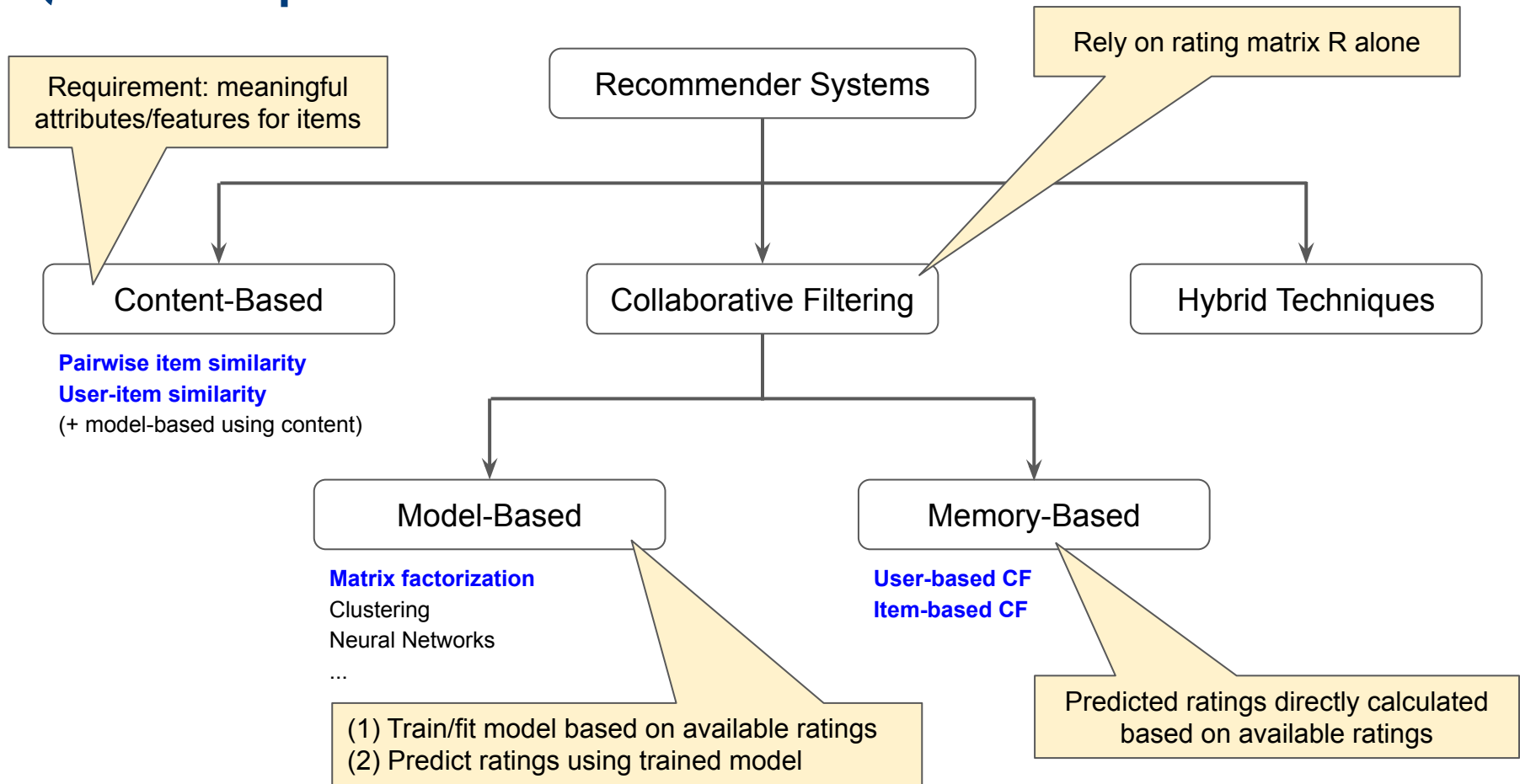
- Basic setup

- 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 stars, binary 0/1)
- If available: information (features) about each item



	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



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

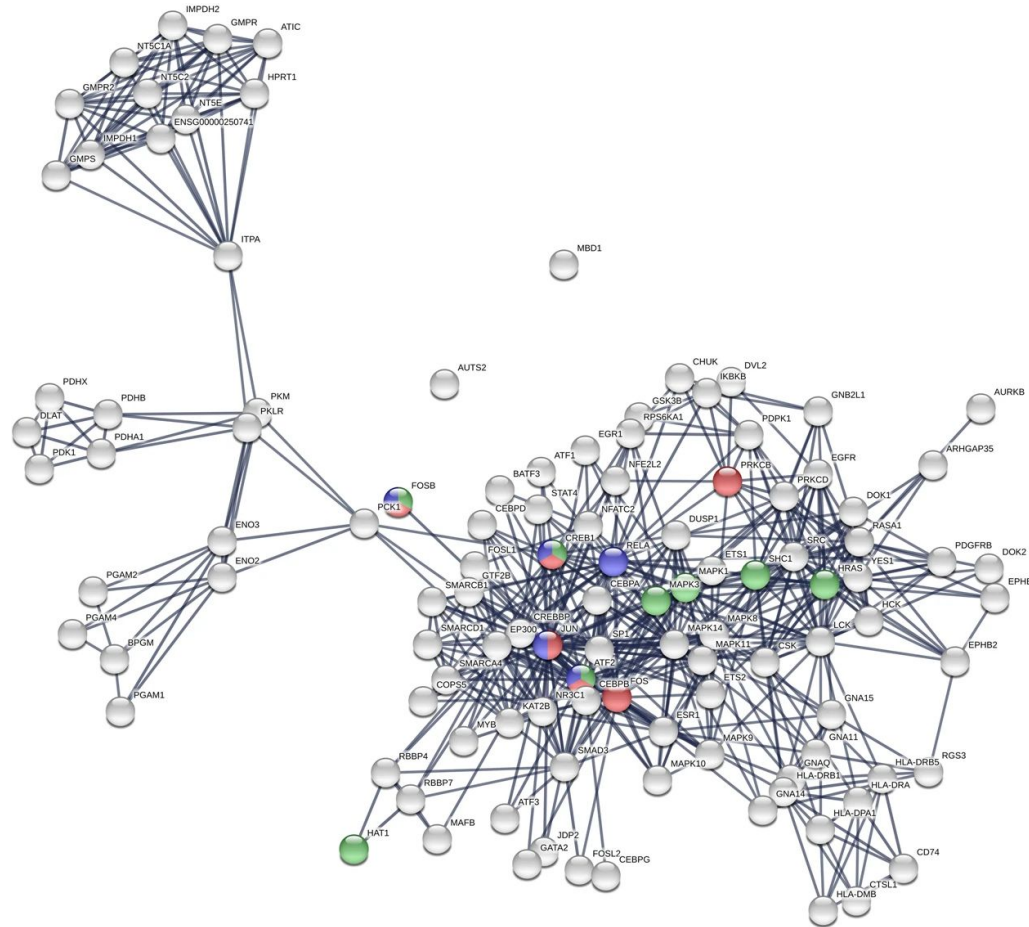
Protein-Protein Interactions

Nodes: Proteins

Edges: Physical interactions between two proteins

Applications:

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



Graphs are Everywhere

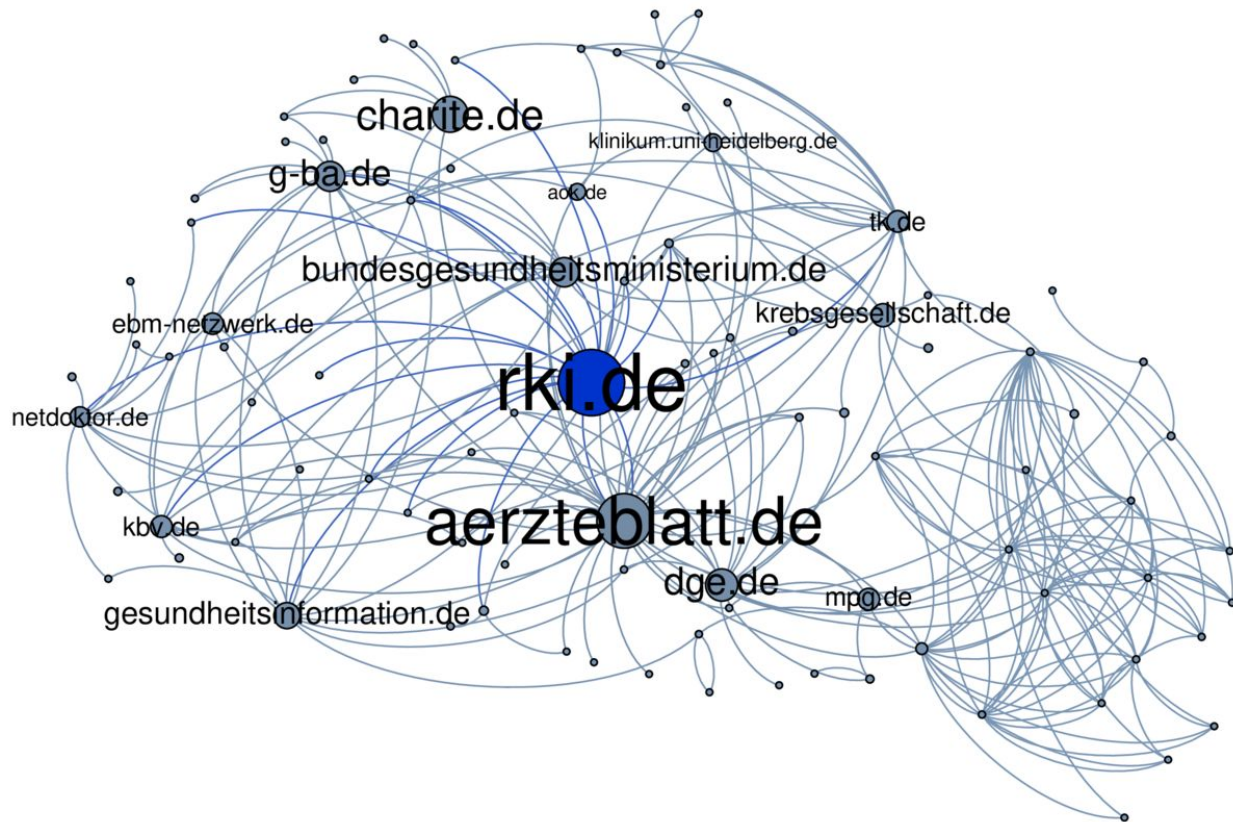
Web Graph

Nodes: Web pages

Edges: Hyperlinks

Applications:

- Identify authoritative sites
- Effective Web search



Graphs are Everywhere

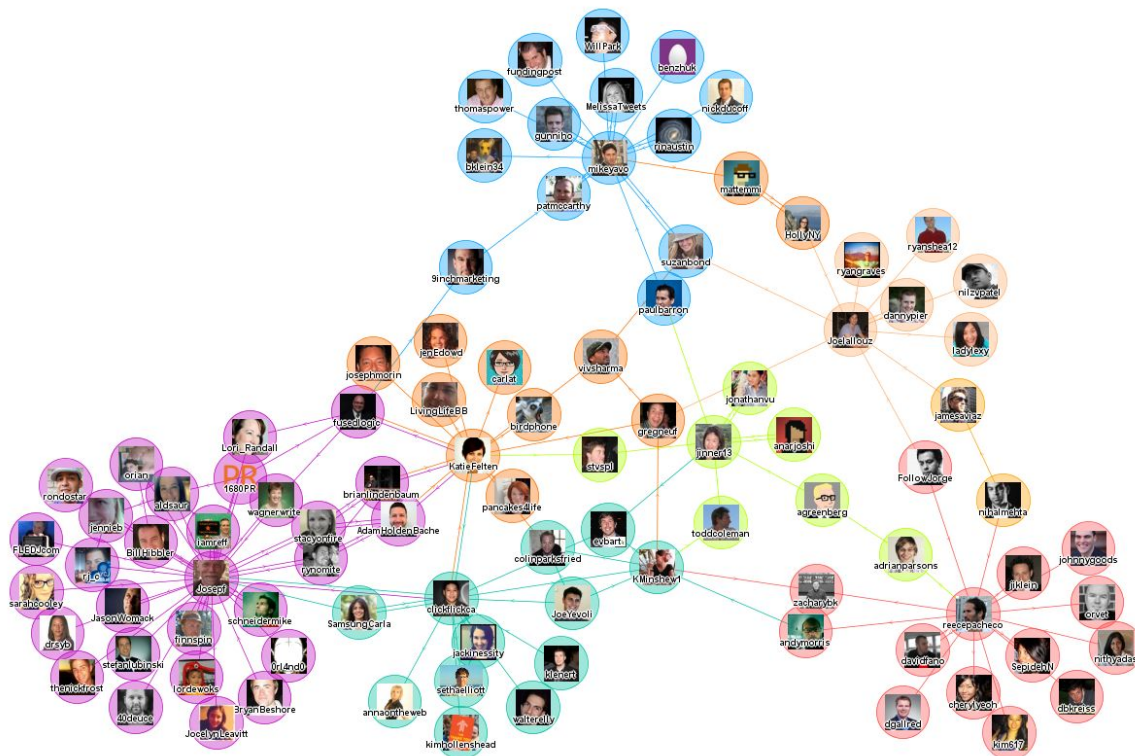
Social Networks

Nodes: People, users

Edges: friendship relationships

Applications:

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



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

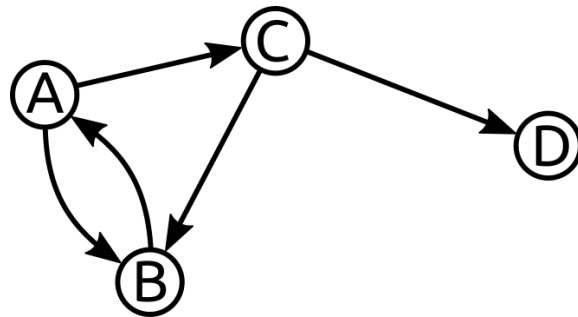
$$V = \{v_1, v_2, \dots, v_n\}$$

- Set of edges E

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

where an edge is a pair of vertices

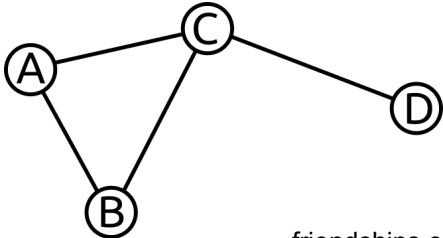
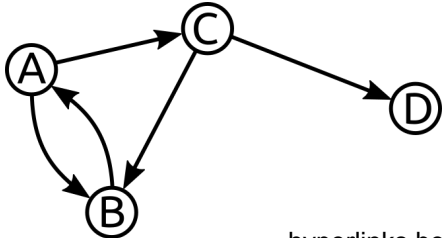
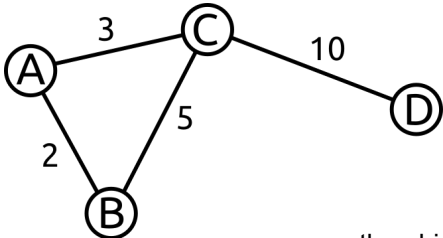
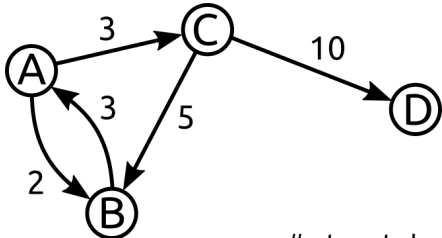
$$e_i = (v_j, v_k)$$



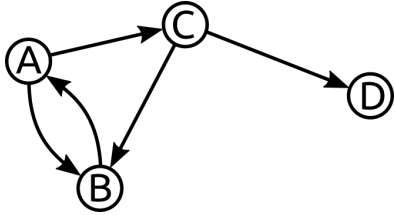
$$V = \{A, B, C, D\}$$

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

Types of Graphs

	undirected	directed
unweighted	 <p>friendships on Facebook directly connected MRT stations</p>	 <p>hyperlinks between web pages followers on Twitter</p>
weighted	 <p>co-authorship with number of papers transport network with travel times</p>	 <p>#retweets between Twitter users borrower network with money owed</p>

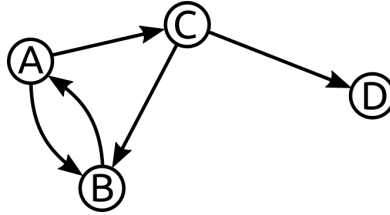
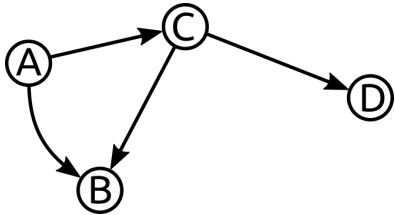
Types of Graphs



Cyclic Graph

vs.

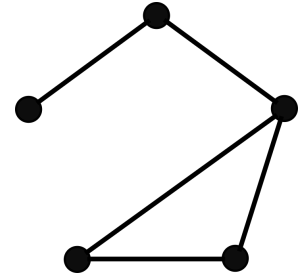
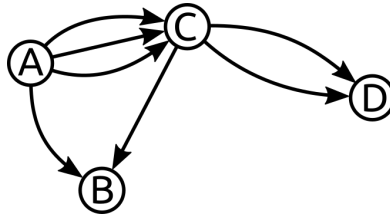
Acyclic Graph



(Simple) Graph

vs.

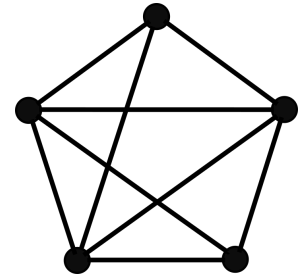
Multigraph



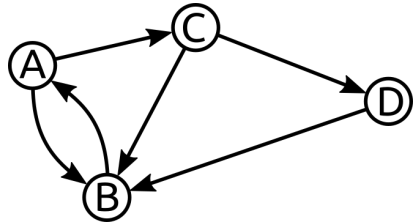
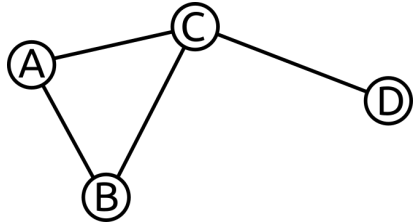
Sparse Graph

vs.

Dense Graph

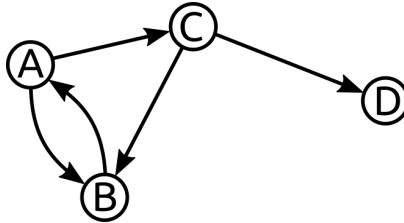


Types of Graphs



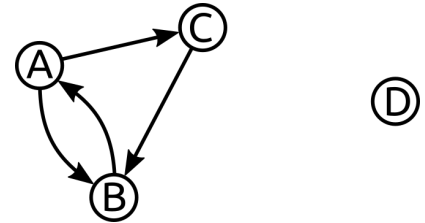
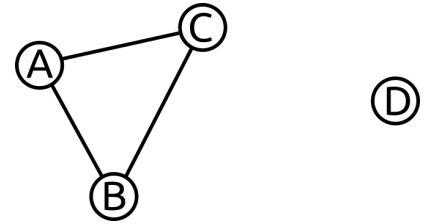
(Strongly) Connected Graph

There exists a path from each node to every other node



Weakly Connected Graph

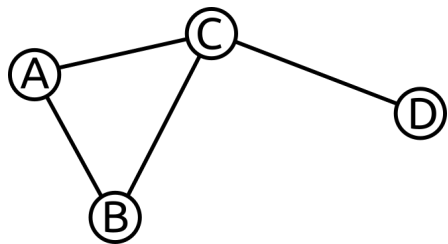
A directed graph where the underlying undirected graph is (strongly) connected



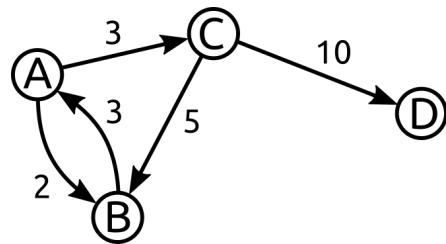
Disconnected Graph

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



Quick Quiz



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

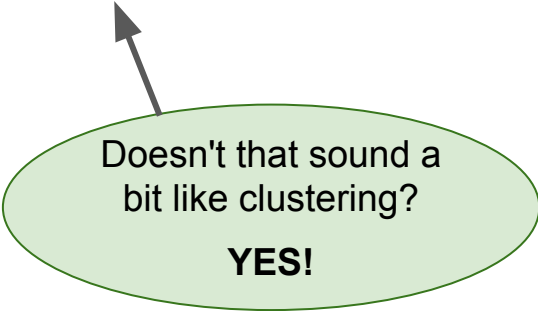
- No formal definition of a community

- 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 application

- 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

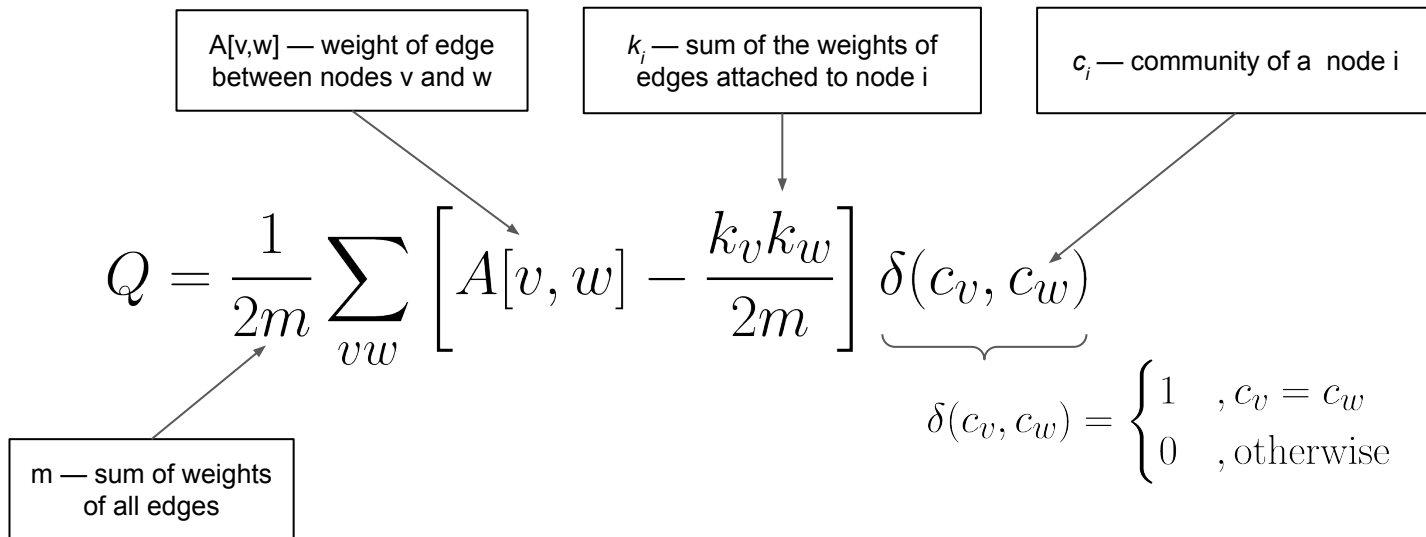


Diagram illustrating the components of the Modularity formula Q :

- $A[v,w]$ — 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
- m — sum of weights of all edges

$$Q = \frac{1}{2m} \sum_{vw} \left[A[v, w] - \frac{k_v k_w}{2m} \right] \underbrace{\delta(c_v, c_w)}_{\delta(c_v, c_w) = \begin{cases} 1 & , c_v = c_w \\ 0 & , \text{otherwise} \end{cases}}$$

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

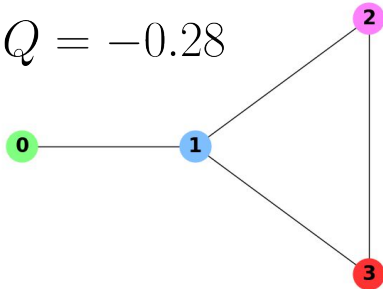
Phase 2: **Graph Aggregation**

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

Louvain Algorithm — Modularity Optimization

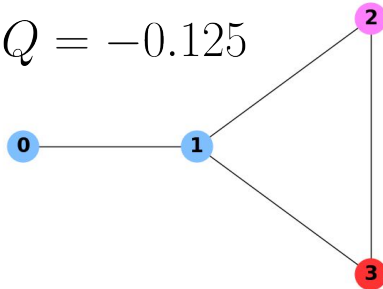
Input: Each node a community

$$Q = -0.28$$



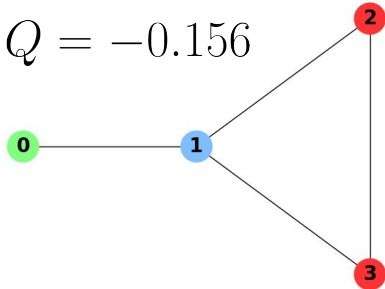
Moving community 0 to community 1

$$Q = -0.125$$



Moving community 2 to community 3

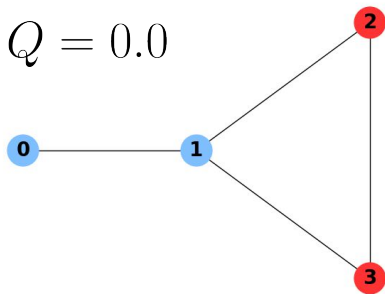
$$Q = -0.156$$



→ Move community 0 to community 1 as it maximizes modularity

Output after
all iterations

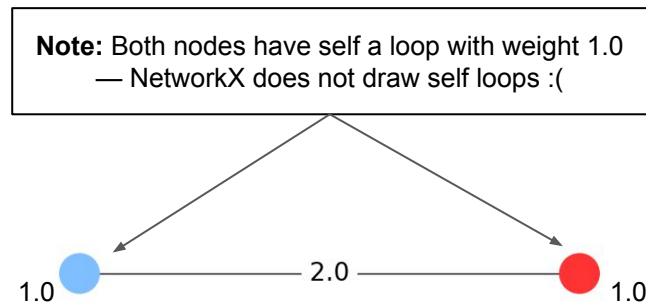
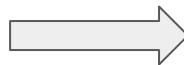
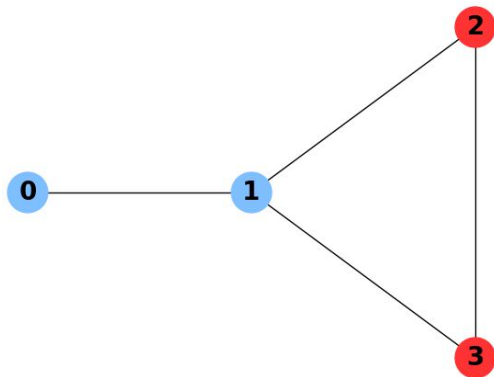
$$Q = 0.0$$



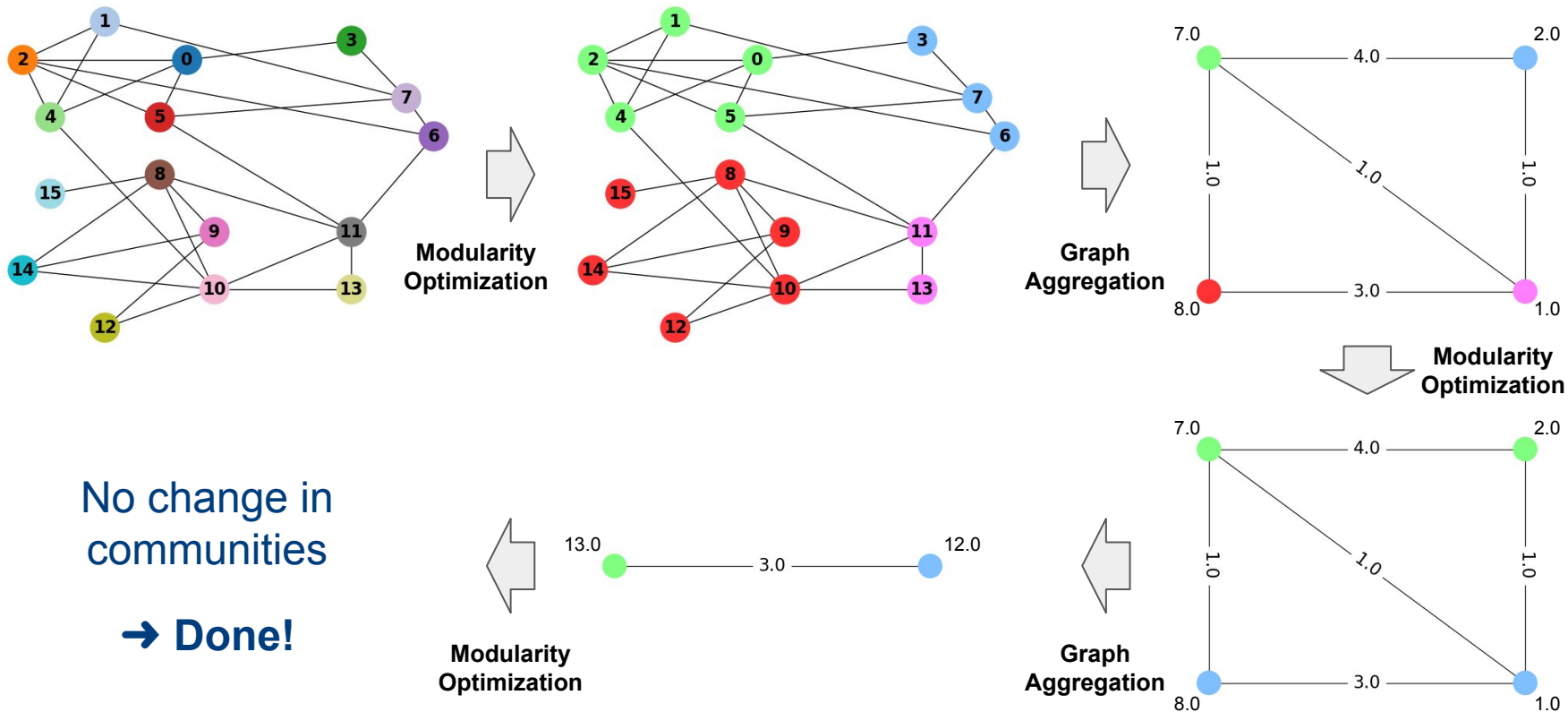
Note: Moving a node may not be permanent but can change again in later iteration before convergence.

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



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)

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

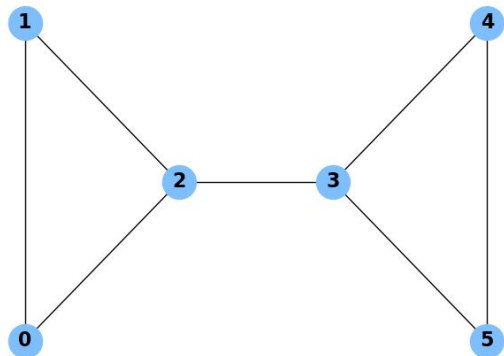
$$c_B(e) = \sum_{v,w \in V} \frac{\sigma(v, w|e)}{\sigma(v, w)}$$

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

number of shortest paths from v to w

Girvan-Newman Algorithm — Edge Betweenness Centrality

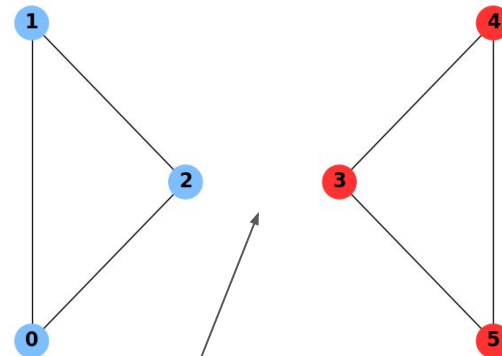
Input community



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 split
(not needed in this simple example)

Girvan-Newman Algorithm

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

Repeat

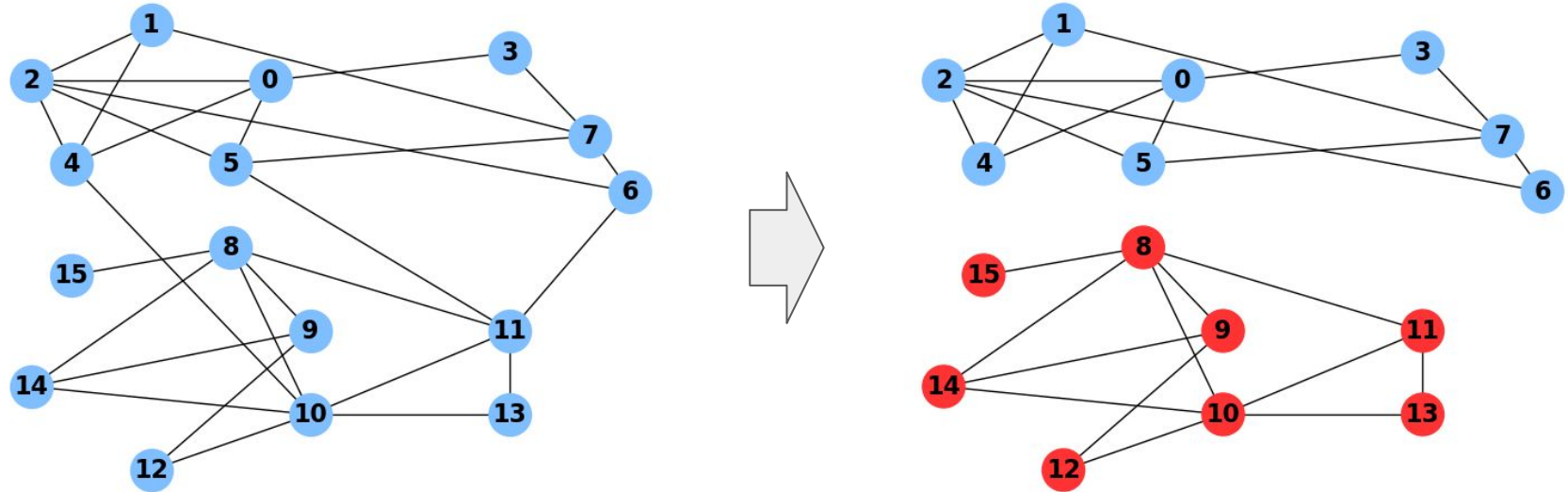
Calculate $c_B(e)$ for all $e \in E$

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.)

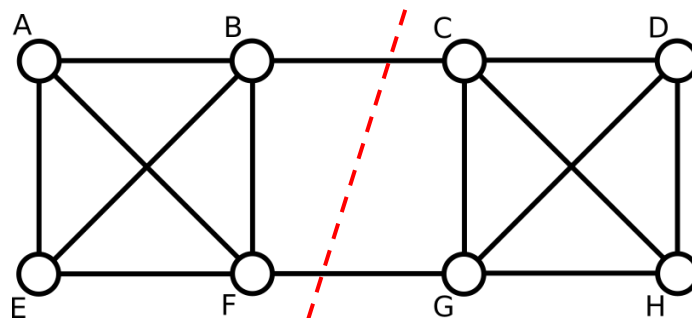
Time Complexity
n^3
$n^3(\log\log n)/\log n^{1/3}$
$n^3(\log\log n/\log n)^{1/2}$
$n^3/(\log n)^{1/2}$
$n^3(\log\log n/\log n)^{5/7}$
$n^3\log\log n/\log n$
$n^3(\log\log n)^{1/2}/\log n$
$n^3/\log n$
$n^3(\log\log n/\log n)^{5/4}$
$n^3(\log\log n)^3/(\log n)^2$
$n^3(\log\log n)/(\log n)^2$

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



→ $|\text{Min-Cut}| = 2$

- 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

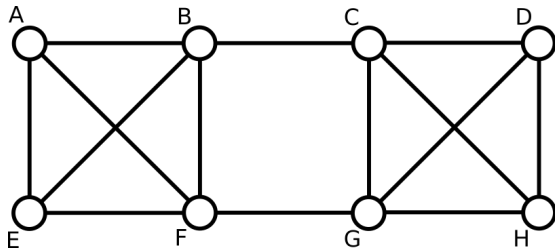
Return edges between the final 2 nodes as Min-Cut

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

Basic runtime: $O(|V|^2)$

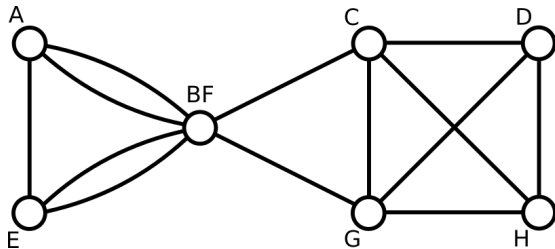
(but further optimizations exists)

Karger's Algorithm for Min-Cut — Example



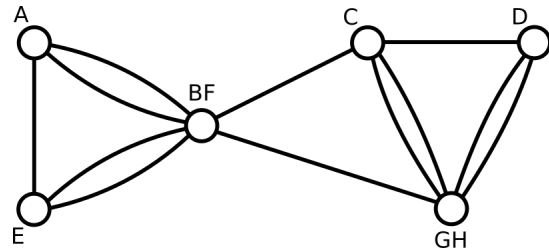
Number of edges: 14

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



Number of edges: 13

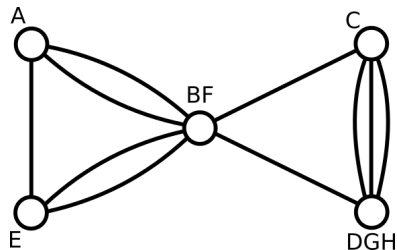
Pick $e = (G, H)$ with $P(e) = 1/13$



Number of edges: 12

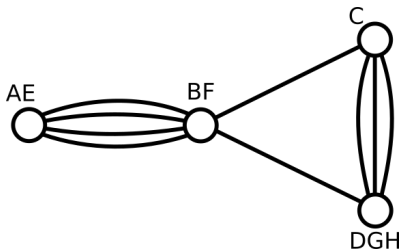
Pick $e = (D, GH)$ with $P(e) = 1/6$

Karger's Algorithm for Min-Cut — Example



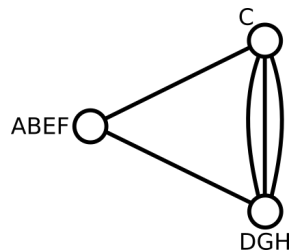
Number of edges: 10

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 $\frac{1}{n} \sum_{v \in V} \text{degree}(v) = \frac{2m}{n}$

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

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

Karger's Algorithm for Min-Cut — Analysis

- Let $P(\text{success}) = P(\text{final cut is Min-Cut})$

$$P(\text{success}) = P(\text{1st selected edge not in Min-Cut}) \times \\ P(\text{2nd selected edge not in Min-Cut}) \times \dots$$

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

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 $P(\text{failure})$ that it is not a Min-Cut?

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

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

Note: For any $x \geq 1$:

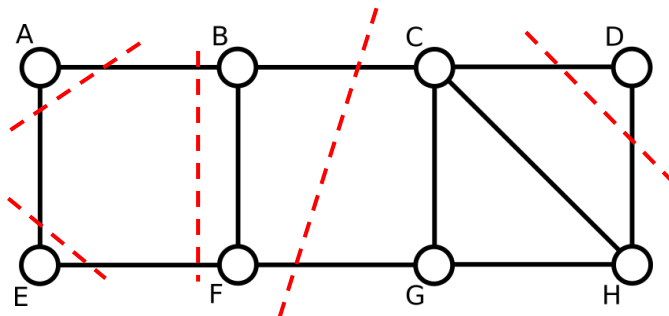
$$\frac{1}{4} \leq \left(1 - \frac{1}{x} \right)^{cx} \leq \left(\frac{1}{e} \right)^c$$

With $k \in O(n^2 \log n) \rightarrow$ Total runtime of Karger's Algorithm is in **$O(n^4 \log n)$**

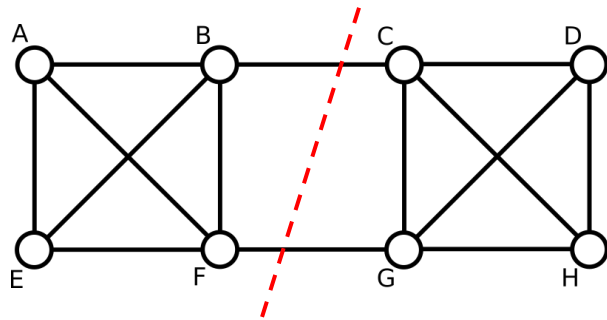
(with a Min-Cut with a high probability)

Karger's Algorithm for Min-Cut — Remarks

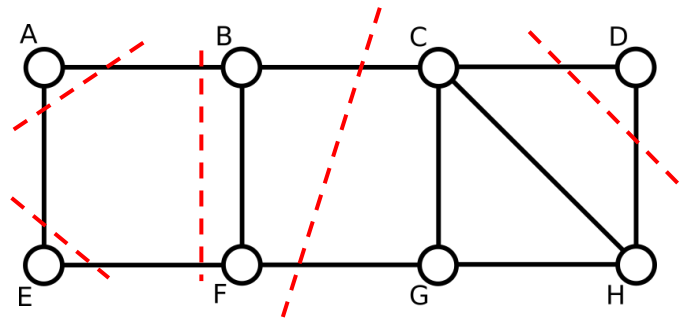
- In general, a graph has multiple possible Min-Cuts



- 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

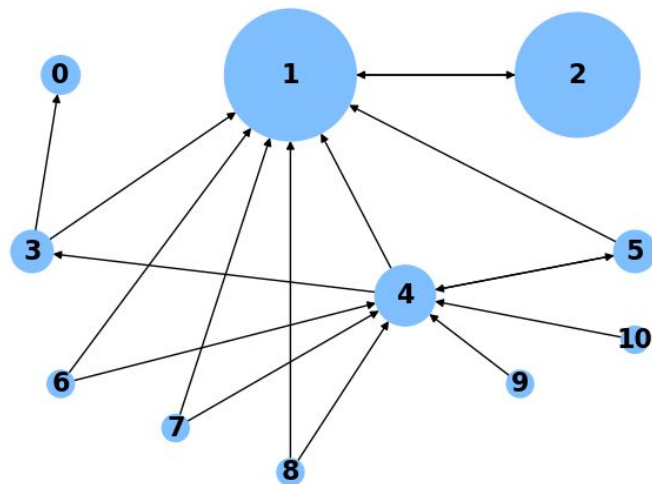
- What makes a node important?

(or more important compared to other nodes)

- Wide range of applications

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

Example: PageRank of 10 web pages



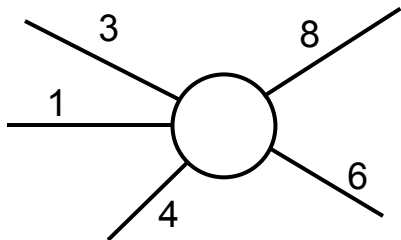
Degree Centrality

- Centrality of a node only dependent on direct neighborhood edges

Undirected graph

Sum of weights of connected edges

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

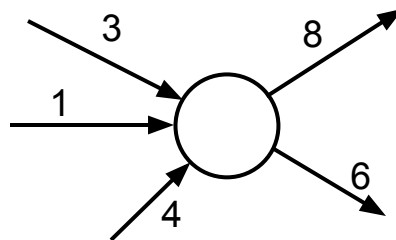


$$c_d(v) = 22$$

Directed graph

Sum of weights of incoming edges

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



$$c_{d_in}(v) = 8$$

Sum of weights of outgoing edges

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

$$c_{d_out}(v) = 14$$

Note: For unweighted graphs, $A[i, j] = 1$ for existing edges \rightarrow 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

- 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)

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

λ — some normalization constant

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

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

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:

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

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

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

Repeat

$$t = t + 1$$

$$x_t = Ax_{t-1}$$

$$x_t = x_t / \|x_t\| \quad \# \text{ Normalize vector}$$

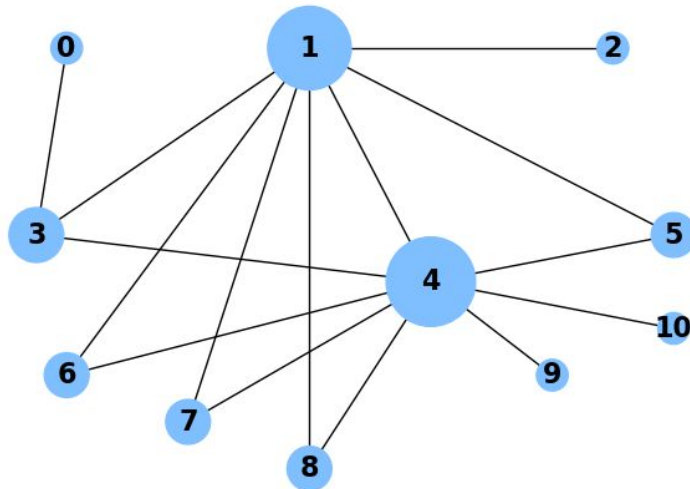
$$\delta = \|x_t - x_{t-1}\| \quad \# \text{ Calculate difference}$$

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

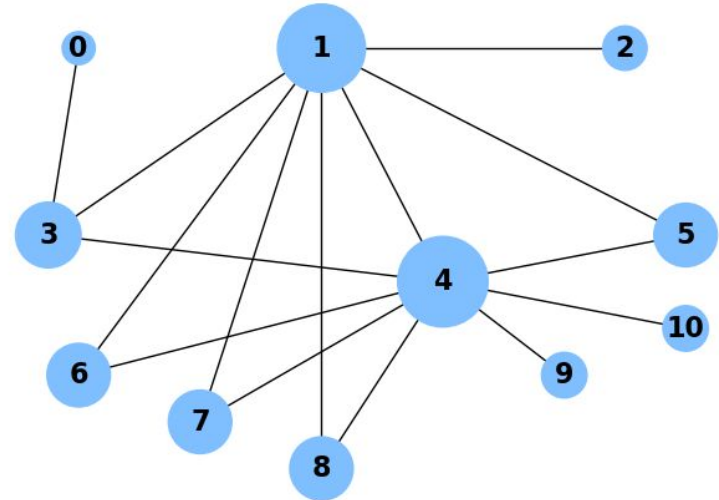
Return x_t

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

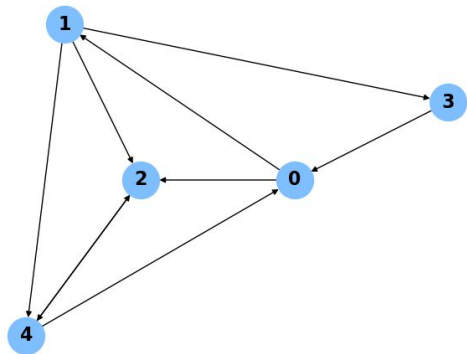
$$PR(v_j) = \sum_{i \rightarrow j} \frac{1}{outdegree(i)} PR(v_i)$$



Matrix notation for all vertices

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

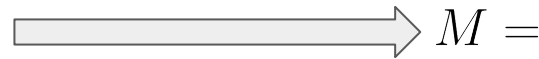
PageRank — Transition Matrix M



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

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



Since M is column-stochastic,
the largest eigenvalue $\lambda = 1$.

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

Common Eigenvector equation:

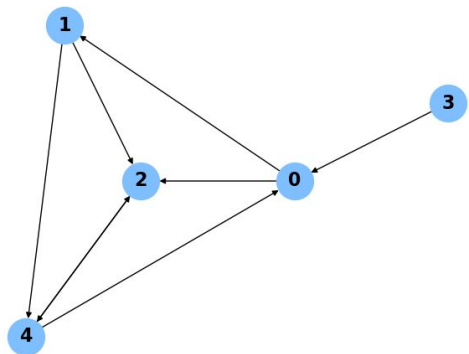
$$A\vec{x} = \lambda\vec{x}$$

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

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

Computing PageRank Scores — Challenge

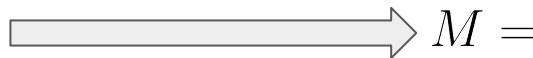
- Requirement: Graph must be strongly connected
 - Each node can be reached from any other node
 - Requirement does not hold for the web graph → dangling nodes
 - Computing PageRank scores using Power Method not working



Adjacency matrix A

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

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



Transition matrix M

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

Computing PageRank Scores — Handling Dangling Nodes

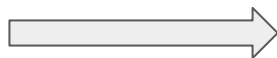
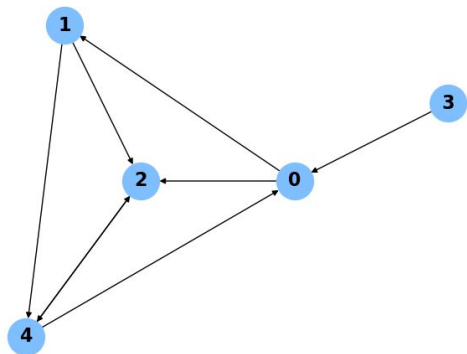
- PageRank implements **Random Surfer** model
 - α probability of following a link on a page to the next
 - $(1 - \alpha)$ probability of jumping to a random page (i.e., not following a link)
- Any page can be reached, whether linked or not!

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

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

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 Iteratio Method without problems



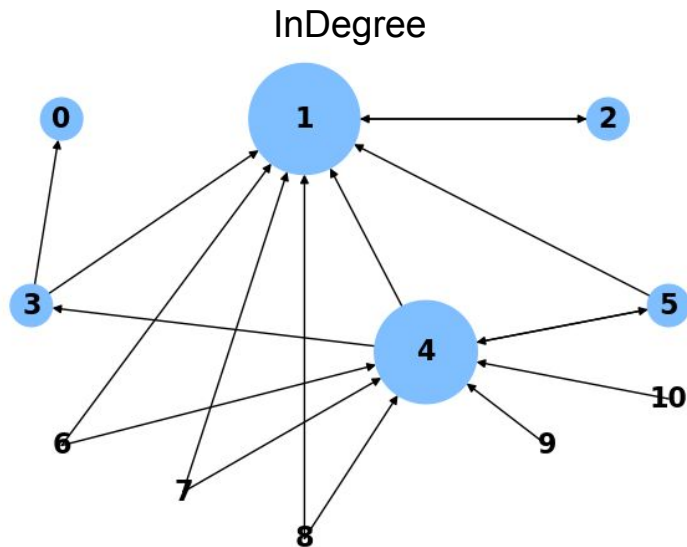
$M =$

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

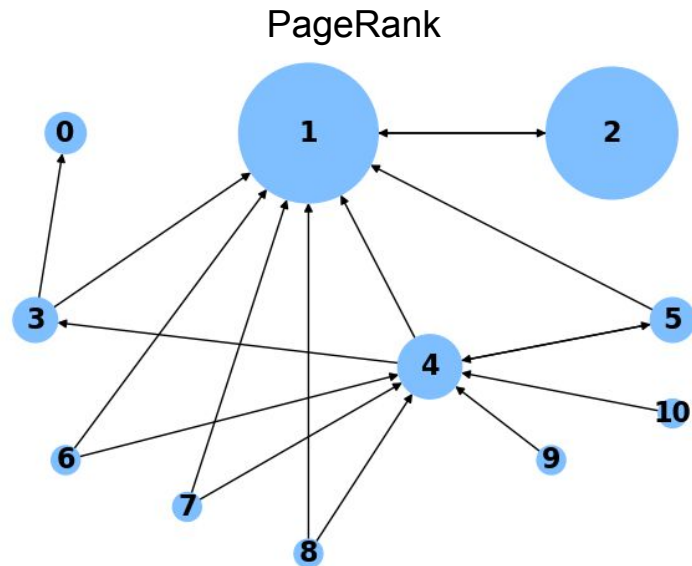
Implied* transition matrix M

***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)



- 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



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 outgoing edges for calculating distances
 - Basic definition applicable to unweighted graph
(generalized definitions for weighted graphs have been proposed)

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

N — number of nodes reachable from v

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

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

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)

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

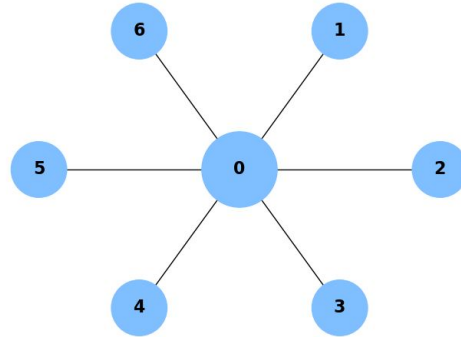
number of shortest paths from s to t passing through node v

total number of shortest paths from s to t

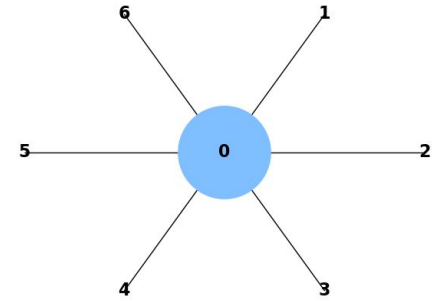
Closeness vs. Betweenness (size of nodes reflect centrality scores)

Star network graph

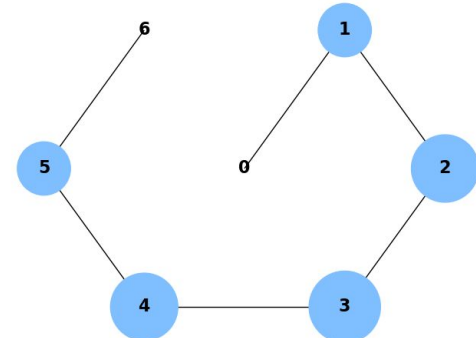
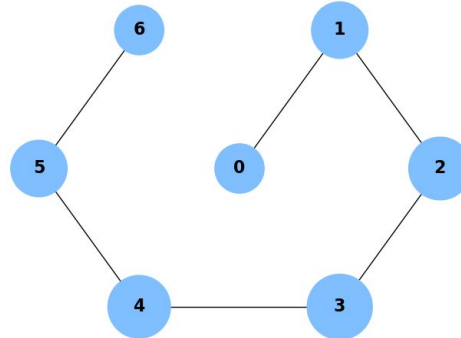
Closeness



Betweenness

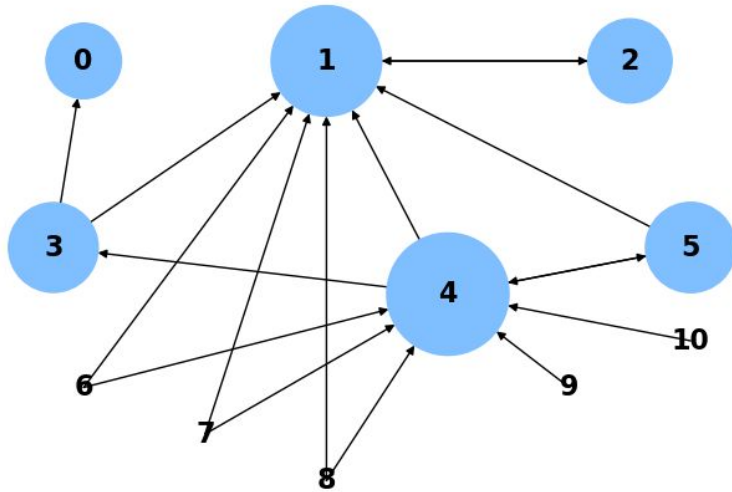


Sequence graph

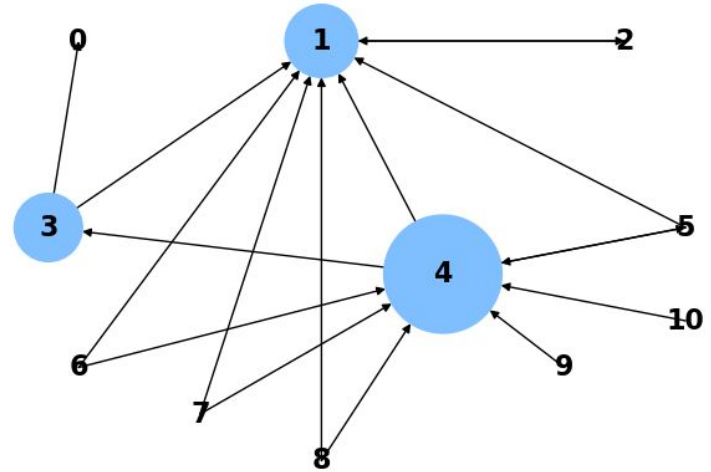


Closeness vs. Betweenness (size of nodes reflect centrality scores)

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
$n^3(\log\log n)/\log n^{1/3}$
$n^3(\log\log n/\log n)^{1/2}$
$n^3/(\log n)^{1/2}$
$n^3(\log\log n/\log n)^{5/7}$
$n^3\log\log n/\log n$
$n^3(\log\log n)^{1/2}/\log n$
$n^3/\log n$
$n^3(\log\log n/\log n)^{5/4}$
$n^3(\log\log n)^3/(\log n)^2$
$n^3(\log\log n)/(\log n)^2$

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



Quick Quiz



Quick Quiz



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**

Summary

- Graph Mining

- Data often comes in form of graphs — incl. very large dataset
(traffic network, social networks, relationship networks, interaction networks, etc.)
- Goal: find interesting patterns based on graph structure
(number of nodes, number of edges, distribution of edges/connections, substructures, etc.)

- In this lecture: 2 types of patterns

- Community — subgraph with nodes more connected among each other than the rest
- Centrality — importance of a node depending its embedding in the graph

Both allow for different definitions → wide range of algorithms

Solutions to Quick Quizzes

