

# INTRODUCTION TO DATA SCIENCE

**JOHN P DICKERSON**

**PREM SAGGAR**

Lecture #14 – 10/15/2018

**CMSC320**

**Mondays & Wednesdays**

**2:00pm – 3:15pm**



**COMPUTER SCIENCE**  
UNIVERSITY OF MARYLAND

# ANNOUNCEMENTS

## Mini-Project #2 is due this week!

- It is linked to from ELMS; also available at:  
<https://github.com/umddb/cmsc320-fall2018/tree/master/project2>
- Deliverable is a .ipynb file submitted to ELMS
- Due **Friday, October 19th**



# ANNOUNCEMENTS

## Midterm is next week!

- Wednesday, October 24<sup>th</sup>
- During class, 75 minutes

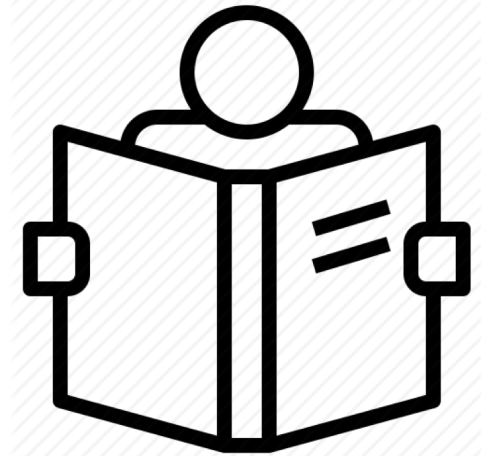
## I will post an **old midterm** on Piazza today

- Note that the order in which certain topics are covered has changed, so some of the questions may not make sense yet!

## I will do a **midterm review** in class next Monday.

## You are allowed **one 8½ x 11” sheet of paper**, single side, handwritten note sheet

- Must be written by you, not copied from somebody else
- It will be turned in at the end of class



# AND NOW!

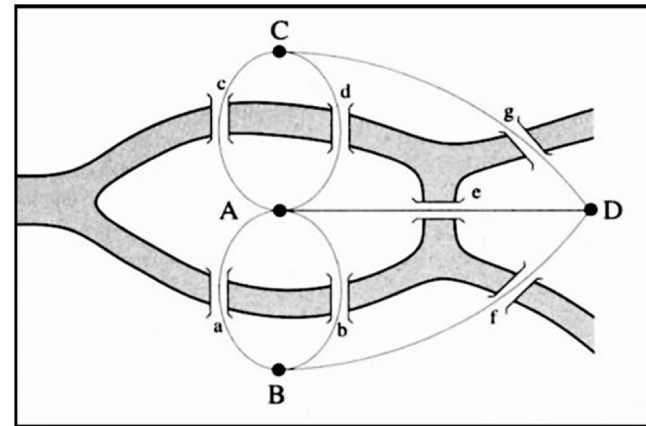
## Graph Processing

- Representing graphs
- Centrality measures
- Community detection

## Natural Language Processing

- Bag of Words, TF-IDF, N-grams
- (If we get to this today ...)

Thank you to: Sukumar Ghosh (Iowa), Lei Tang (Yahoo!), Huan Liu (ASU), Zico Kolter (CMU)



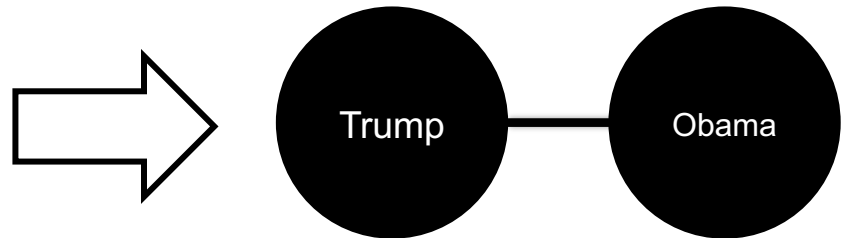


# NETWORKS? GRAPHS?

**Networks** are systems of interrelated objects

**Graphs** are the mathematical models used to represent networks

In data science, we will use algorithms on graphs to answer questions about real-world networks.

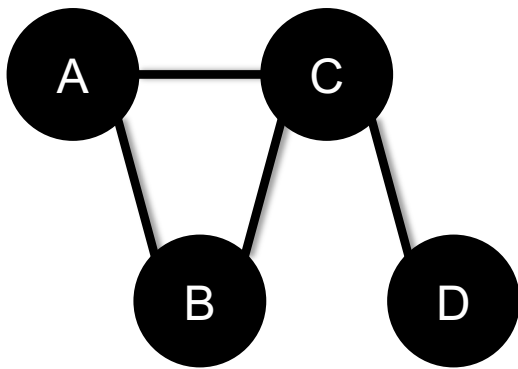


# GRAPHS

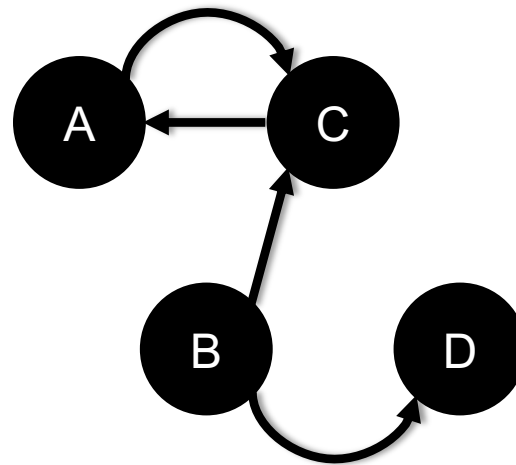
Nodes = Vertices  
Edges = Arcs

A **graph**  $G = (V,E)$  is a set of **vertices**  $V$  and **edges**  $E$

Edges can be undirected or directed



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



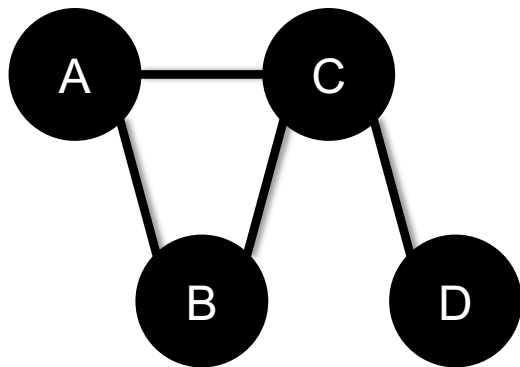
$V = \{A, B, C, D\}$   
 $E = \{(A,C), (C,A), (B,C), (B,D)\}$

Examples of directed vs undirected graphs ??????????????

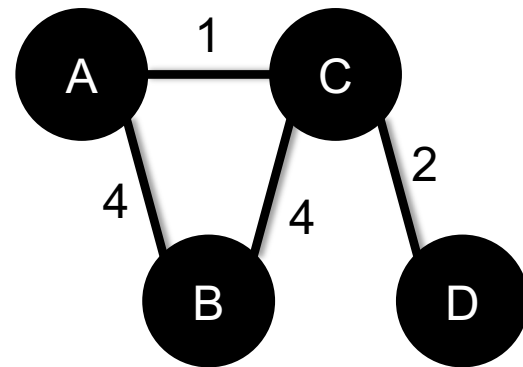
# GRAPHS

Edges can be unweighted or weighted

- Unweighted  $\rightarrow$  all edges have unit weight



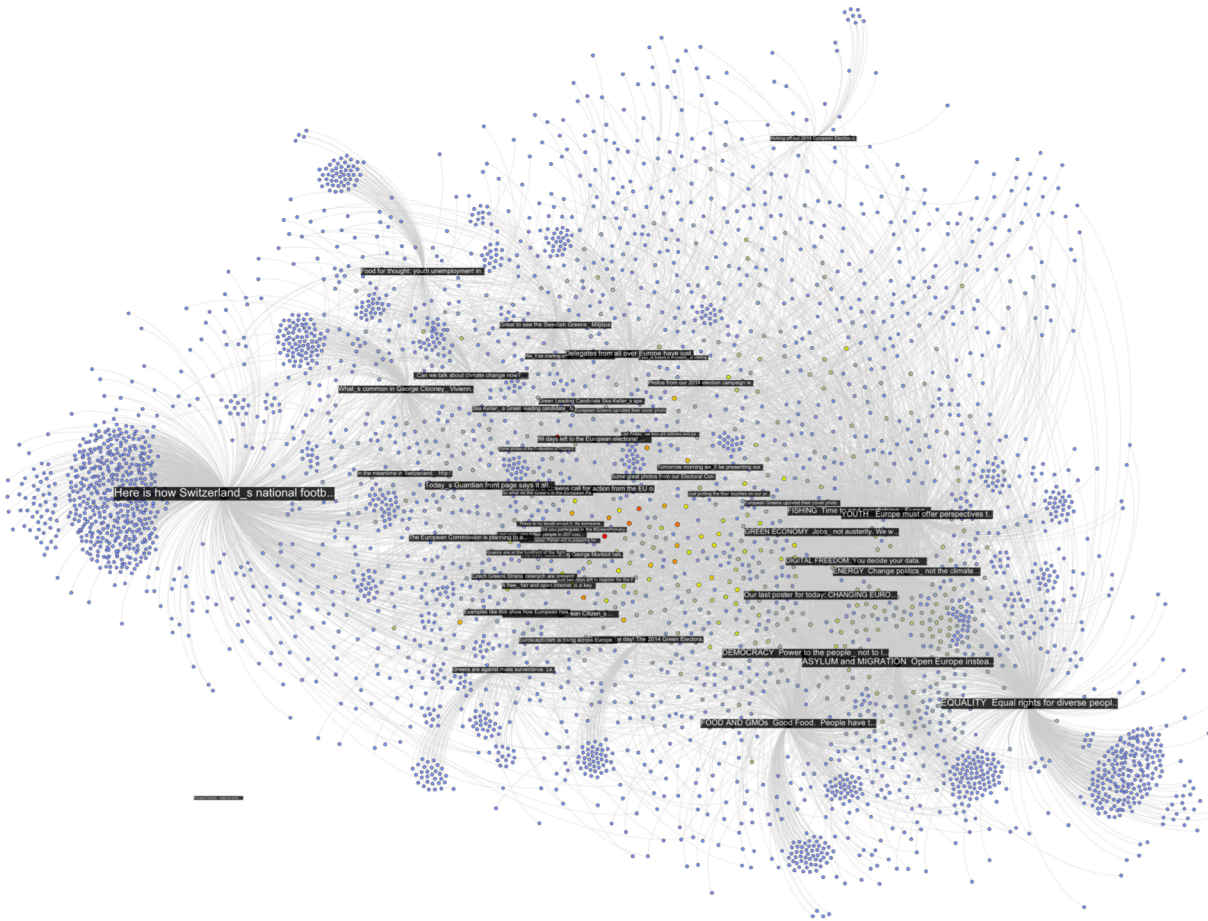
Unweighted



Weighted

Examples of unweighted and weighted graphs ??????????????

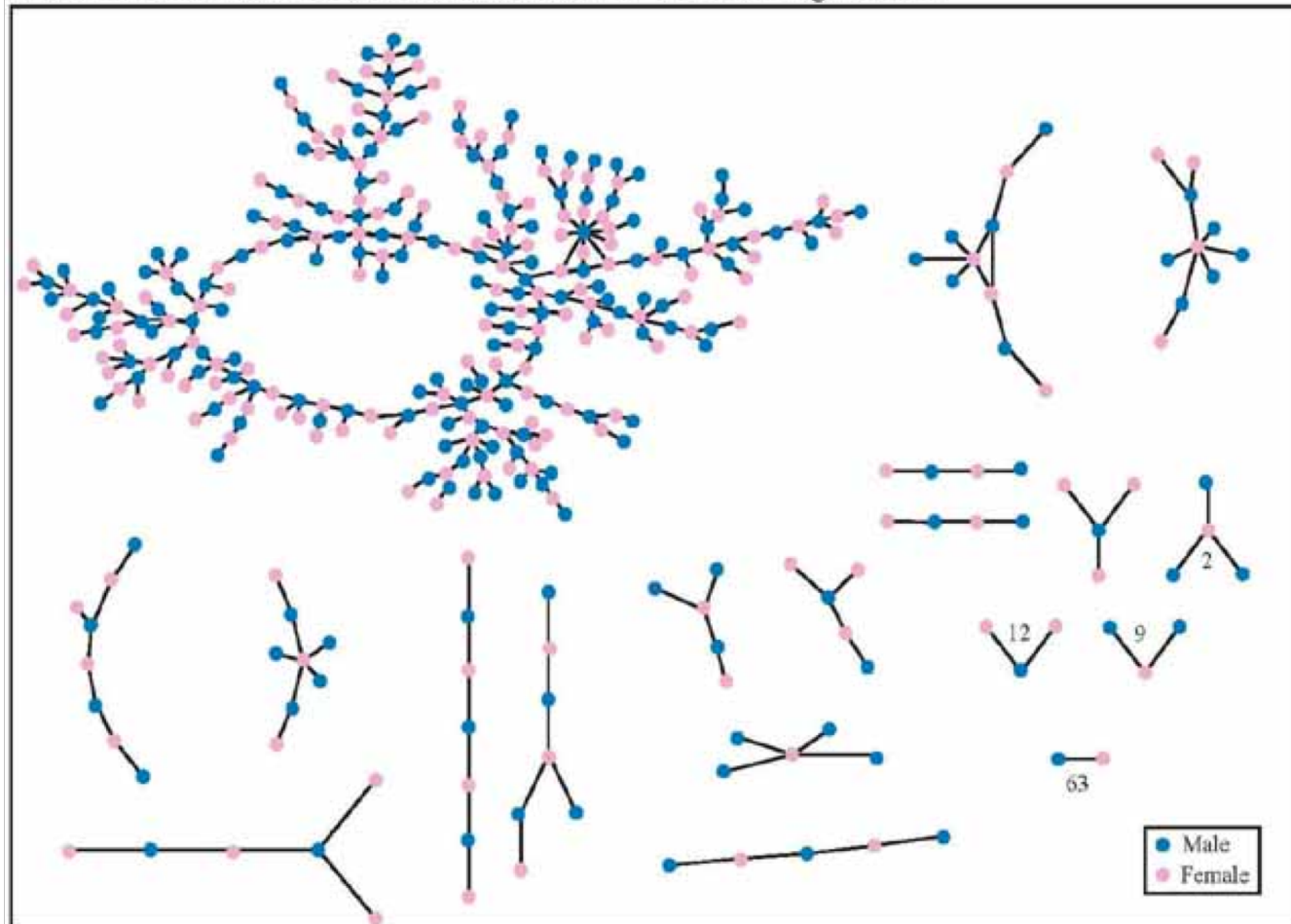
# GRAPHS AND THE NETWORKS THEY REPRESENT



Facebook posts (in black), and users liking or commenting on those posts

# GRAPHS AND THE NETWORKS THEY REPRESENT

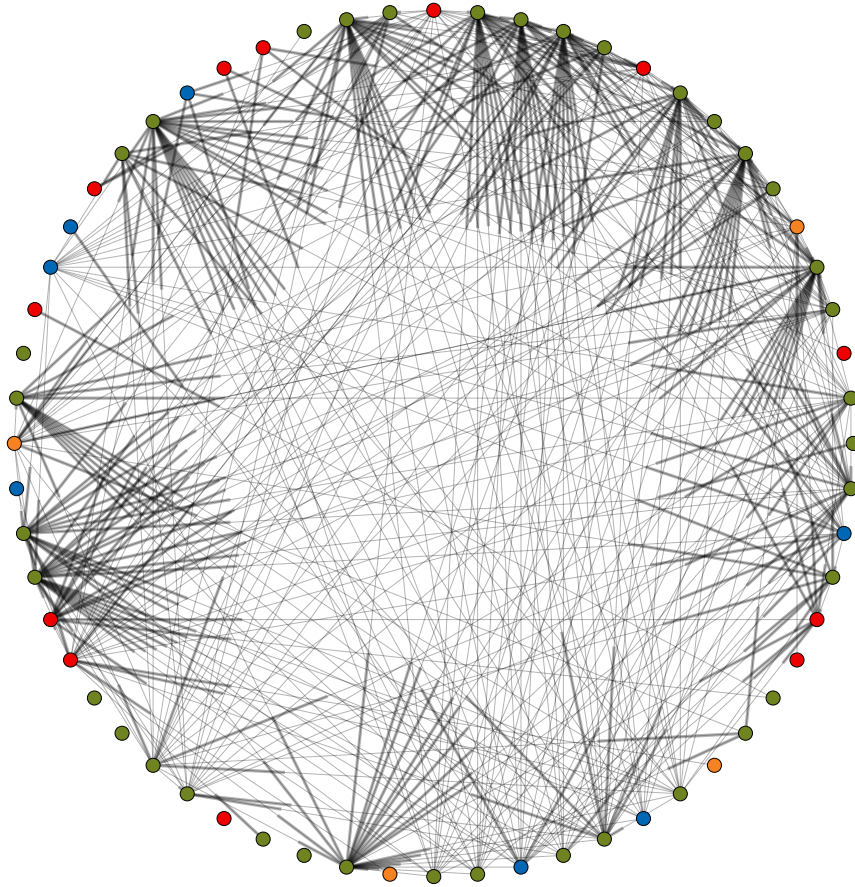
The Structure of Romantic and Sexual Relations at "Jefferson High School"



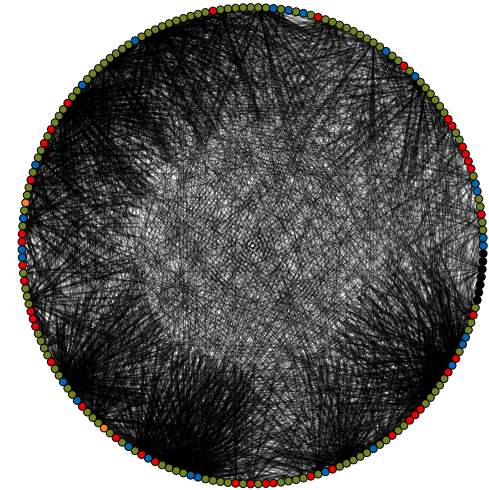
Each circle represents a student and lines connecting students represent romantic relations occurring within the 6 months preceding the interview. Numbers under the figure count the number of times that pattern was observed (i.e. we found 63 pairs unconnected to anyone else).



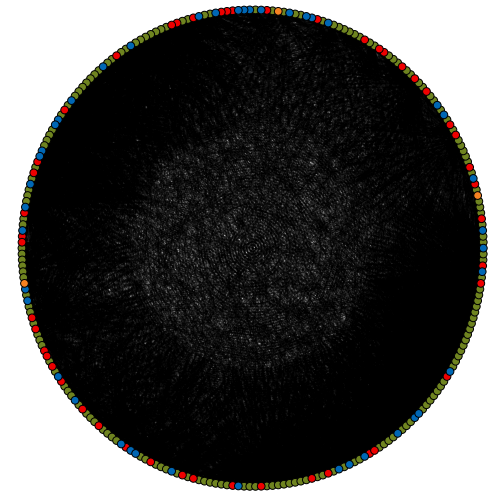
# GRAPHS AND THE NETWORKS THEY REPRESENT



UNOS, 2010-12-08



UNOS, 2012-09-10



UNOS, 2014-06-30

# NETWORKX

**NetworkX** is a Python library for storing, manipulating, and analyzing (small- and medium-sized) graphs

- Uses Matplotlib for rendering
- <https://networkx.github.io/>
- `conda install -c anaconda networkx`

```
import networkx as nx

G=nx.Graph()
G.add_node("spam")
G.add_edge(1,2)

print(list(G.nodes()))
print(list(G.edges())) [(1, 2)]
```

```
[1, 2, 'spam']
[(1,2)]
```

# STORING A GRAPH

Three main ways to **represent** a graph in memory:

- Adjacency lists
- Adjacency dictionaries
- Adjacency matrix

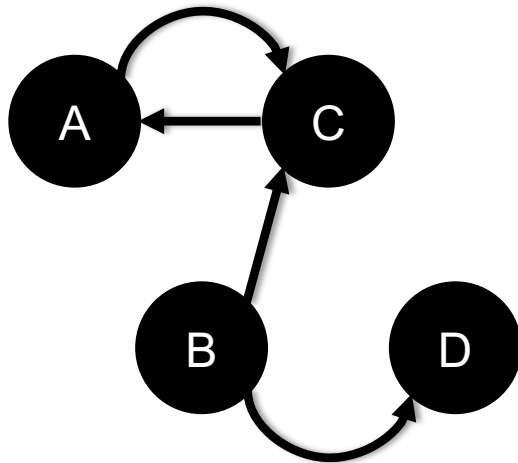
The storage decision should be made based on the expected use case of your graph:

- Static analysis only?
- Frequent updates to the structure?
- Frequent updates to semantic information?



# ADJACENCY LISTS

For each vertex, store an array of the vertices it connects to



Vertex	Neighbors
A	[C]
B	[C, D]
C	[A]
D	[]

**Pros:** ??????????

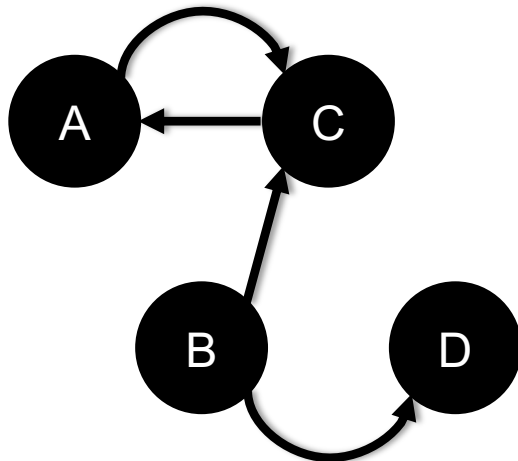
- Iterate over all outgoing edges; easy to add an edge

**Cons:** ??????????

- Checking for the existence of an edge is  $O(|V|)$ , deleting is hard

# ADJACENCY DICTIONARIES

For each vertex, store a dictionary of vertices it connects to



Vertex	Neighbors
A	{C: 1.0}
B	{C: 1.0, D: 1.0}
C	{A: 1.0}
D	{}

**Pros:** ??????????

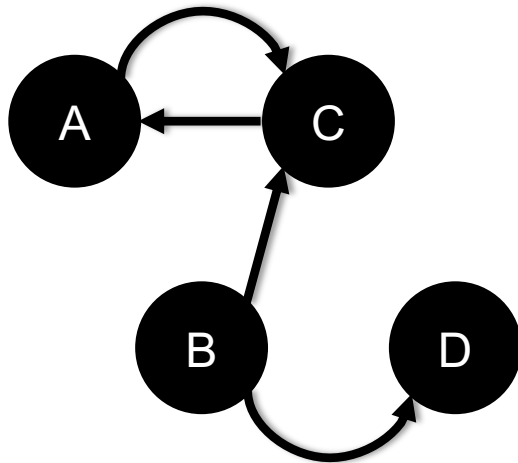
- $O(1)$  to add, remove, query edges

**Cons:** ??????????

- Overhead (memory, caching, etc)

# ADJACENCY MATRIX

Store the connectivity of the graph in a matrix



		From			
		A	B	C	D
To	A	0	0	1	0
	B	0	0	0	0
	C	1	1	0	0
	D	0	1	0	0

Cons: ??????????

- $O(|V|^2)$  space regardless of the number of edges

Almost always stored as a **sparse matrix**

# NETWORKX STORAGE

NetworkX uses an adjacency dictionary representation

- Built-ins for reading from/to SciPy/NumPy matrices

```
# Make a directed 3-cycle
G=nx.DiGraph()
G.add_edges_from([('A','B'), ('B', 'C'), ('C', 'A')])

# Get all out-edges of vertex 'B'
print(G['B'])

# Loop over vertices
for v in G.nodes(): print(v)

# Loop over edges
for u,v in G.edges(): print(u, v)
```

# ASIDE: GRAPH DATABASES

Traditional relational databases store relations between entities directly in the data (e.g., foreign keys)

- Queries search data, JOIN over relations

**Graph databases** directly relate data in the storage system using edges (relations) with attached semantic properties

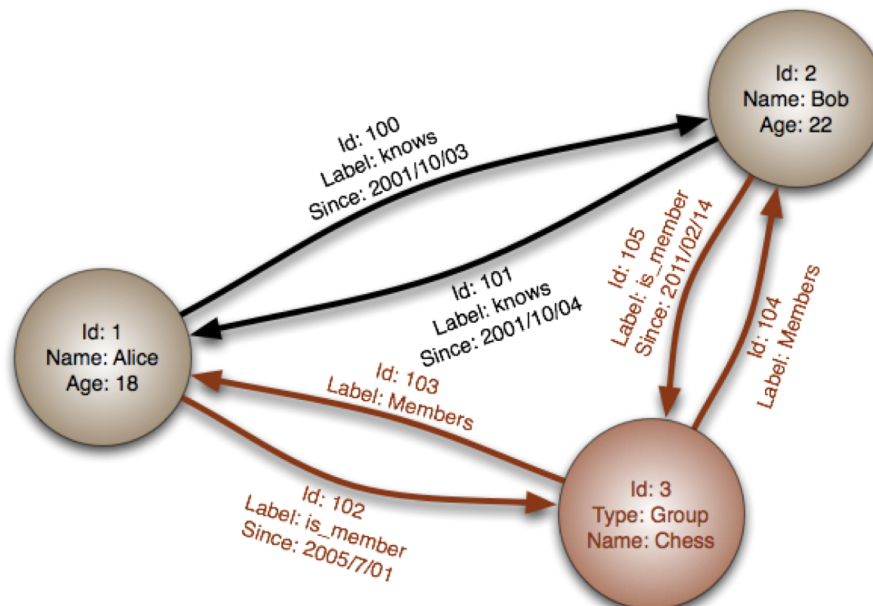
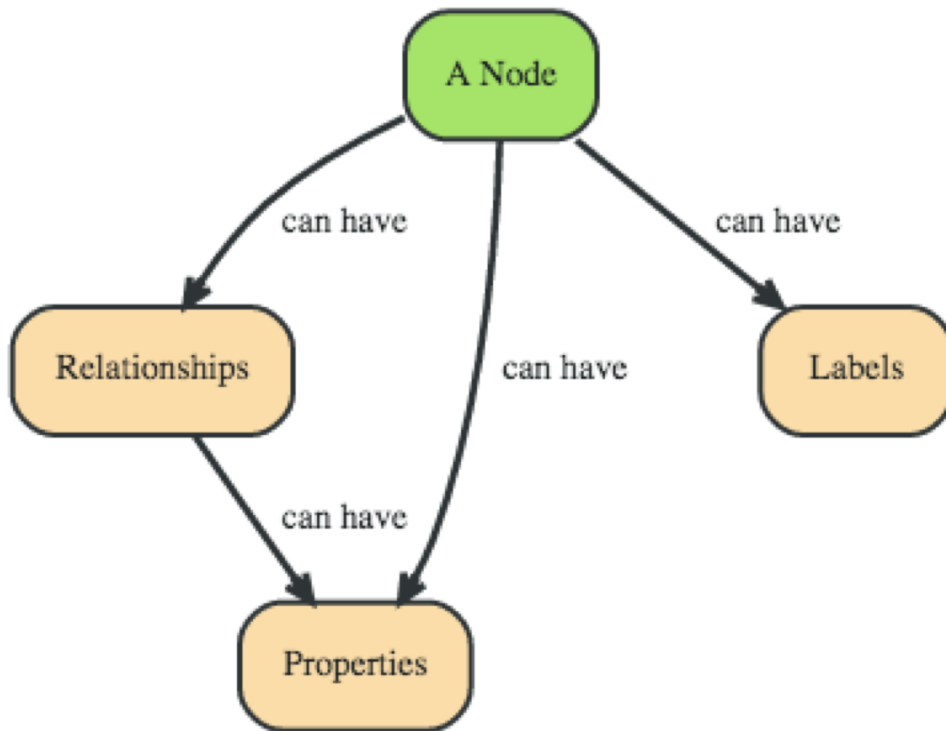


Image thanks to Wikipedia

# EXAMPLE GRAPH DATABASE

**Two people**, John and Sally, are **friends**.  
**Both John and Sally have read the book**, Graph Databases.



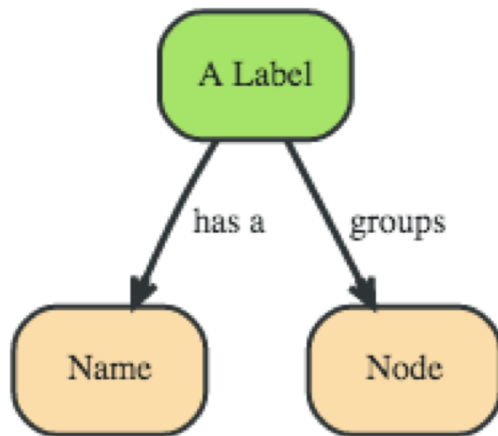
**Nodes ????????????**

- John
- Sally
- Graph Databases

Thanks to: <http://neo4j.com>

# EXAMPLE GRAPH DATABASE

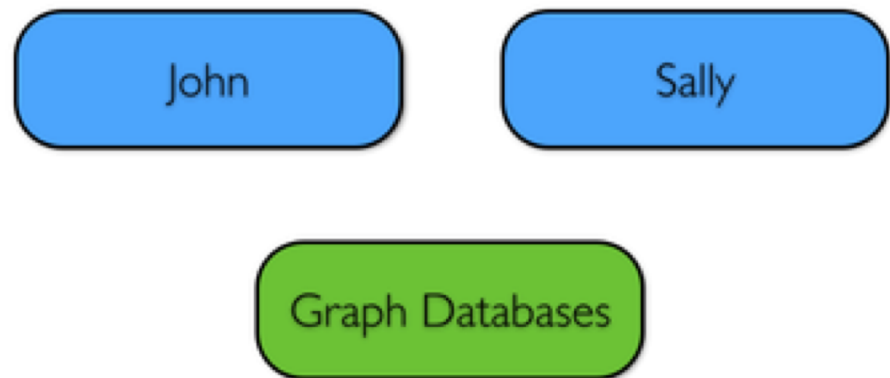
Two *people*, John and Sally, are **friends**.  
Both John and Sally have **read** the *book*, Graph Databases.



A named construct that **groups** nodes into sets

Labels ????????????

- Person
- Book



Next: assign labels to the nodes

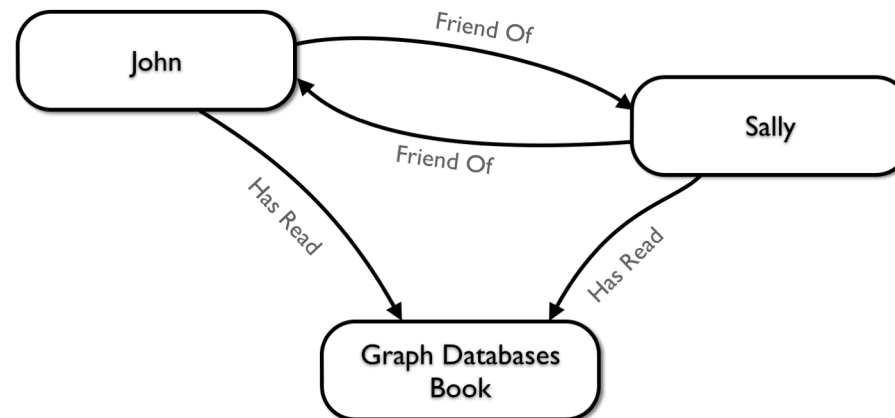
# EXAMPLE GRAPH DATABASE

Two *people*, John and Sally, are **friends**.

Both John and Sally have **read** the *book*, Graph Databases.

Relationships ??????????

- John is a **friend of** Sally; Sally is a **friend of** John
- John has **read** Graph Databases; Sally has **read** Graph Databases

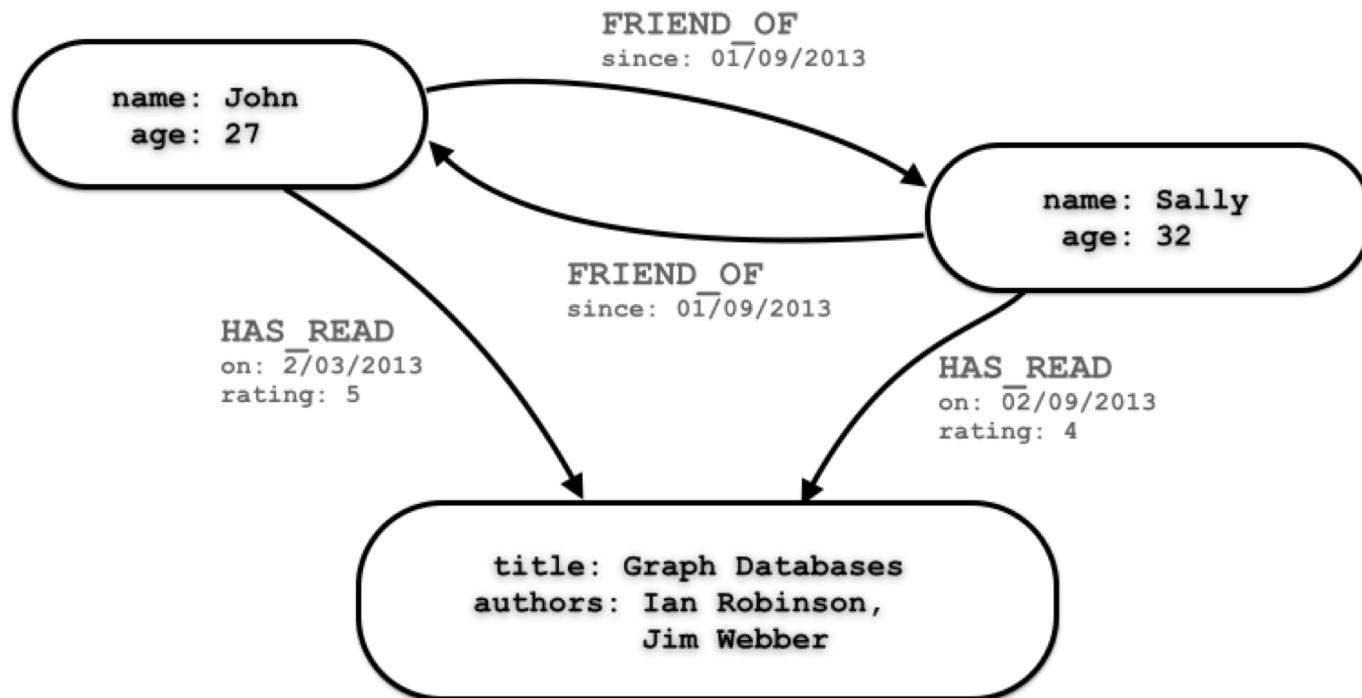




# EXAMPLE GRAPH DATABASE

Can associate **attributes** with entities in a key-value way

- Attributes on nodes, relations, labels



# EXAMPLE GRAPH DATABASE

**Querying** graph databases needs a language other than SQL

**Recall:** graph databases explicitly represent relationships

- Adhere more to an object-oriented paradigm
- May be more suitable for managing ad-hoc data
- May scale better, depending on the query types (no JOINS)

```
# When did Sally and John become friends?  
MATCH (sally:Person { name: 'Sally' })  
MATCH (john:Person { name: 'John' })  
MATCH (sally)-[r:FRIEND_OF]-(john)  
RETURN r.since AS friends_since
```

Cypher query

# BULBFLOW



Many graph databases out there:

- List found here: [https://en.wikipedia.org/wiki/Graph\\_database](https://en.wikipedia.org/wiki/Graph_database)

**neo4j and Titan are popular, easy-to-use solutions**

- <https://neo4j.com/>
- <http://titan.thinkaurelius.com/>



**Bulbflow is a Python framework that connects to several backing graph-database servers like neo4j**

- <http://bulbflow.com/>
- <https://github.com/espeed/bulbs>

# THE VALUE OF A VERTEX



# IMPORTANCE OF VERTICES

Not all vertices are equally important

## Centrality Analysis:

- Find out the most important node(s) in one network
- Used as a feature in classification, for visualization, etc ...

## Commonly-used Measures

- Degree Centrality
- Closeness Centrality
- Betweenness Centrality
- Eigenvector Centrality

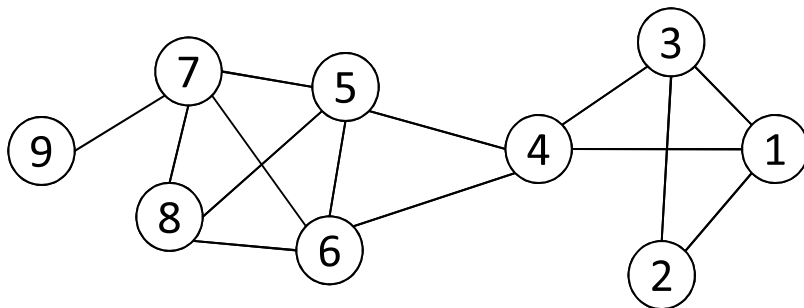
# DEGREE CENTRALITY

The importance of a vertex is determined by the number of vertices adjacent to it

- The larger the degree, the more important the vertex is
- Only a small number of vertex have high degrees in many real-life networks

**Degree Centrality:**  $C_D(v_i) = d_i = \sum_j A_{ij}$

**Normalized Degree Centrality:**  $C'_D(v_i) = d_i / (n - 1)$



For vertex 1, degree centrality is 3;  
Normalized degree centrality is  $3/(9-1)=3/8$ .

# CLOSENESS CENTRALITY

“Central” vertices are important, as they can reach the whole network more quickly than non-central vertices

Importance measured by how **close** a vertex is to other vertices

Average Distance: 
$$D_{avg}(v_i) = \frac{1}{n-1} \sum_{j \neq i}^n g(v_i, v_j)$$

Closeness Centrality:

$$C_C(v_i) = \left[ \frac{1}{n-1} \sum_{j \neq i}^n g(v_i, v_j) \right]^{-1} = \frac{n-1}{\sum_{j \neq i}^n g(v_i, v_j)}$$

# CLOSENESS CENTRALITY

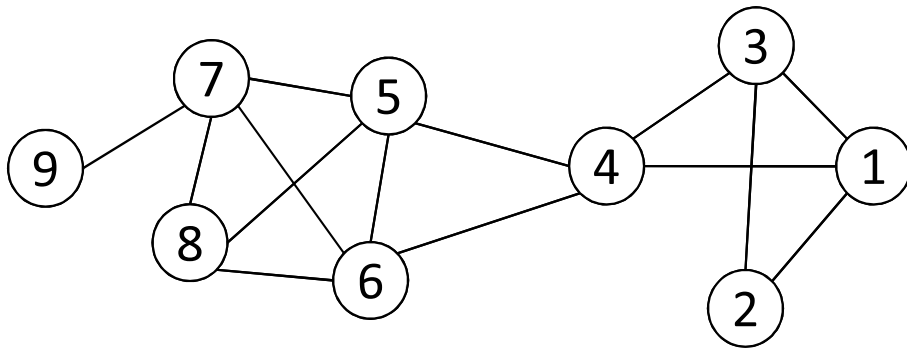


Table 2.1: Pairwise geodesic distance

Node	1	2	3	4	5	6	7	8	9
1	0	1	1	1	2	2	3	3	4
2	1	0	1	2	3	3	4	4	5
3	1	1	0	1	2	2	3	3	4
4	1	2	1	0	1	1	2	2	3
5	2	3	2	1	0	1	1	1	2
6	2	3	2	1	1	0	1	1	2
7	3	4	3	2	1	1	0	1	1
8	3	4	3	2	1	1	1	0	2
9	4	5	4	3	2	2	1	2	0

$$C_C(3) = \frac{9 - 1}{1 + 1 + 1 + 2 + 2 + 3 + 3 + 4} = 8/17 = 0.47,$$

$$C_C(4) = \frac{9 - 1}{1 + 2 + 1 + 1 + 1 + 2 + 2 + 3} = 8/13 = 0.62.$$

Vertex 4 is more central than vertex 3



# BETWEENNESS CENTRALITY

Vertex **betweenness** counts the number of shortest paths that pass through one vertex

Vertices with high betweenness are important in communication and information diffusion

Betweenness Centrality: 
$$C_B(v_i) = \sum_{v_s \neq v_i \neq v_t \in V, s < t} \frac{\sigma_{st}(v_i)}{\sigma_{st}}$$

$\sigma_{st}$  : The number of shortest paths between s and t

$\sigma_{st}(v_i)$  : The number of shortest paths between s and t that pass  $v_i$

# BETWEENNESS CENTRALITY

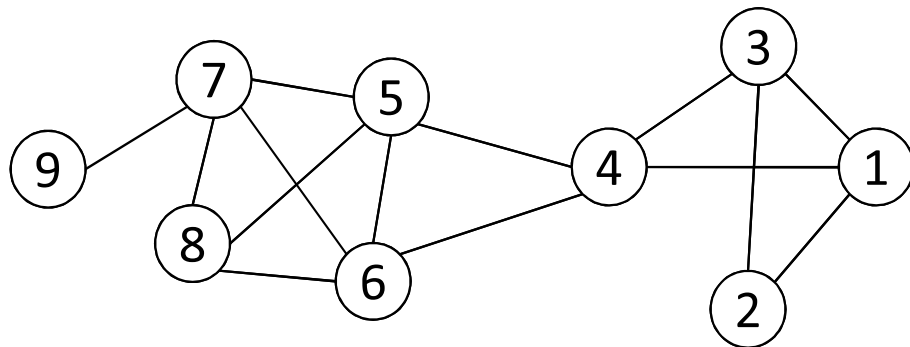


Table 2.2:  $\sigma_{st}(4)/\sigma_{st}$

	$s = 1$	$s = 2$	$s = 3$
$t = 5$	1/1	2/2	1/1
$t = 6$	1/1	2/2	1/1
$t = 7$	2/2	4/4	2/2
$t = 8$	2/2	4/4	2/2
$t = 9$	2/2	4/4	2/2

$\sigma_{st}$  : The number of shortest paths between  $s$  and  $t$

$\sigma_{st}(v_i)$  : The number of shortest paths between  $s$  and  $t$  that pass  $v_i$

$$C_B(v_i) = \sum_{v_s \neq v_i \neq v_t \in V, s < t} \frac{\sigma_{st}(v_i)}{\sigma_{st}}$$

What is the betweenness centrality for node 4 ??????????

# EIGENVECTOR CENTRALITY

A vertex's importance is determined by the **importance of the friends** of that vertex

If one has many important friends, he should be important as well.

$$C_E(v_i) \propto \sum_{v_j \in N_i} A_{ij} C_E(v_j)$$

$$x \propto Ax \quad \longrightarrow \quad Ax = \lambda x.$$

The centrality corresponds to the top eigenvector of the adjacency matrix  $A$ .

A variant of this eigenvector centrality is the PageRank score.

# NETWORKX: CENTRALITY

**Many other centrality measures implemented for you!**

- <https://networkx.github.io/documentation/development/reference/algorithms/centrality.html>

**Degree, in-degree, out-degree**

**Closeness**

**Betweenness**

- Applied to both edges and vertices; hard to compute

**Load: similar to betweenness**

**Eigenvector, Katz (provides additional weight to close neighbors)**

# **STRENGTH OF RELATIONSHIPS**



# WEAK AND STRONG TIES

In practice, connections are not of the same strength

Interpersonal social networks are composed of strong ties (close friends) and weak ties (acquaintances).

Strong ties and weak ties play different roles for **community formation** and **information diffusion**

**Strength of Weak Ties** [Granovetter 1973]

- Occasional encounters with distant acquaintances can provide important information about new opportunities for job search

# CONNECTIONS IN SOCIAL MEDIA

**Social media allows users to connect to each other more easily than ever.**

- One user might have thousands of friends online
- Who are the most important ones among your 300 Facebook friends?

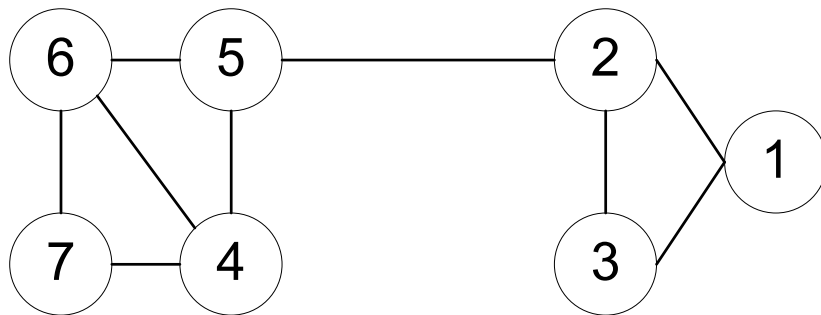
**Imperative to estimate the strengths of ties for advanced analysis**

- Analyze network topology
- Learn from User Profiles and Attributes
- Learn from User Activities

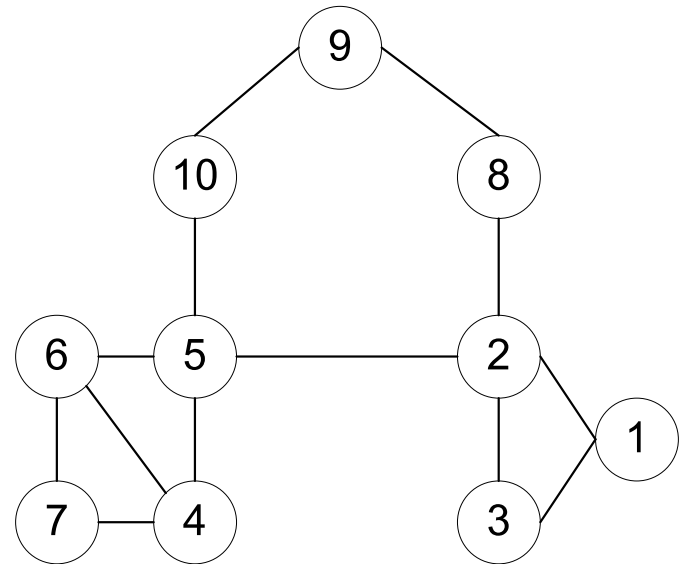
# LEARNING FROM NETWORK TOPOLOGY

**Bridges** connecting two different communities are weak ties

An edge is a bridge if its removal results in disconnection of its terminal vertices



Bridge edge(s) ??????



Bridge edge(s) ??????



# “SHORTCUT” BRIDGE

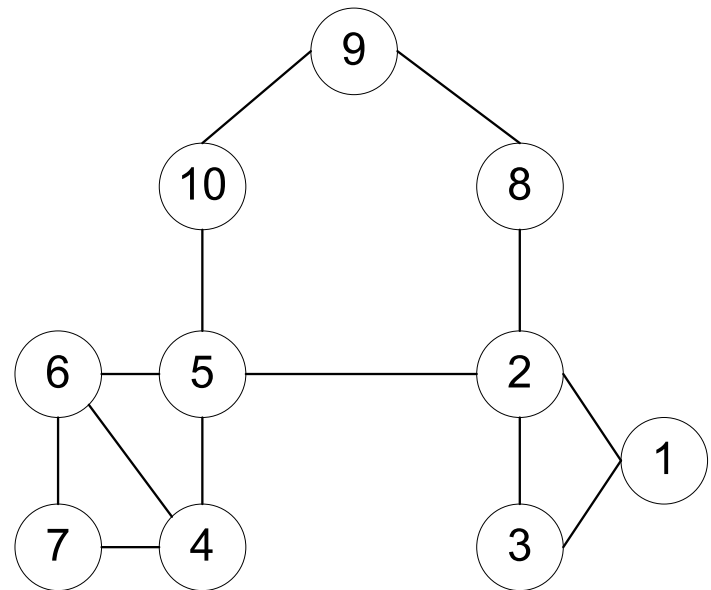
Bridges are rare in real-life networks

Idea: relax the definition by checking if the distance between two terminal vertices increases if the edge is removed

- The larger the distance, the weaker the tie is

**Example:**

- $d(2,5) = 4$  if  $(2,5)$  is removed
- $d(5,6) = 2$  if  $(5,6)$  is removed
- $(5,6)$  is a stronger tie than  $(2,5)$



# NEIGHBORHOOD OVERLAP

Tie strength can be measured based on neighborhood overlap; the larger the overlap, the stronger the tie is.

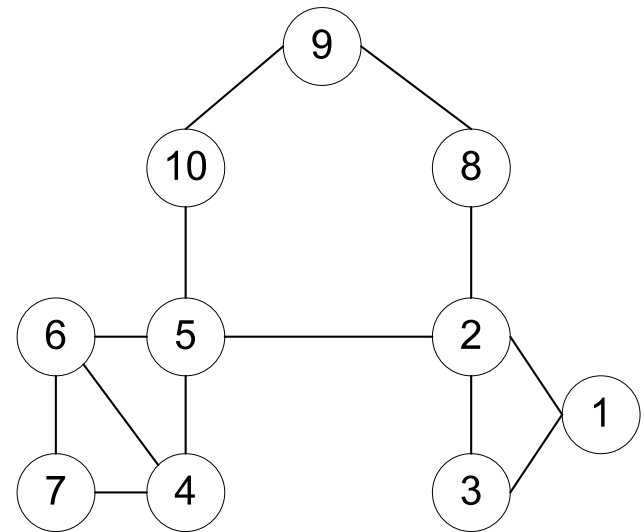
$$\begin{aligned} \text{overlap}(v_i, v_j) &= \frac{\text{number of shared friends of both } v_i \text{ and } v_j}{\text{number of friends who are adjacent to at least } v_i \text{ or } v_j} \\ &= \frac{|N_i \cap N_j|}{|N_i \cup N_j| - 2} \end{aligned}$$

*(-2 in the denominator is to exclude  $v_i$  and  $v_j$ )*

**Example:**

$$\text{overlap}(2, 5) = 0,$$

$$\text{overlap}(5, 6) = \frac{|\{4\}|}{|\{2, 4, 5, 6, 7, 10\}| - 2} = 1/4$$



# LEARNING FROM PROFILES AND INTERACTIONS

## **Twitter: one can follow others without followee's confirmation**

- The real friendship network is determined by the frequency two users talk to each other, rather than the follower-followee network
- The real friendship network is more influential in driving Twitter usage

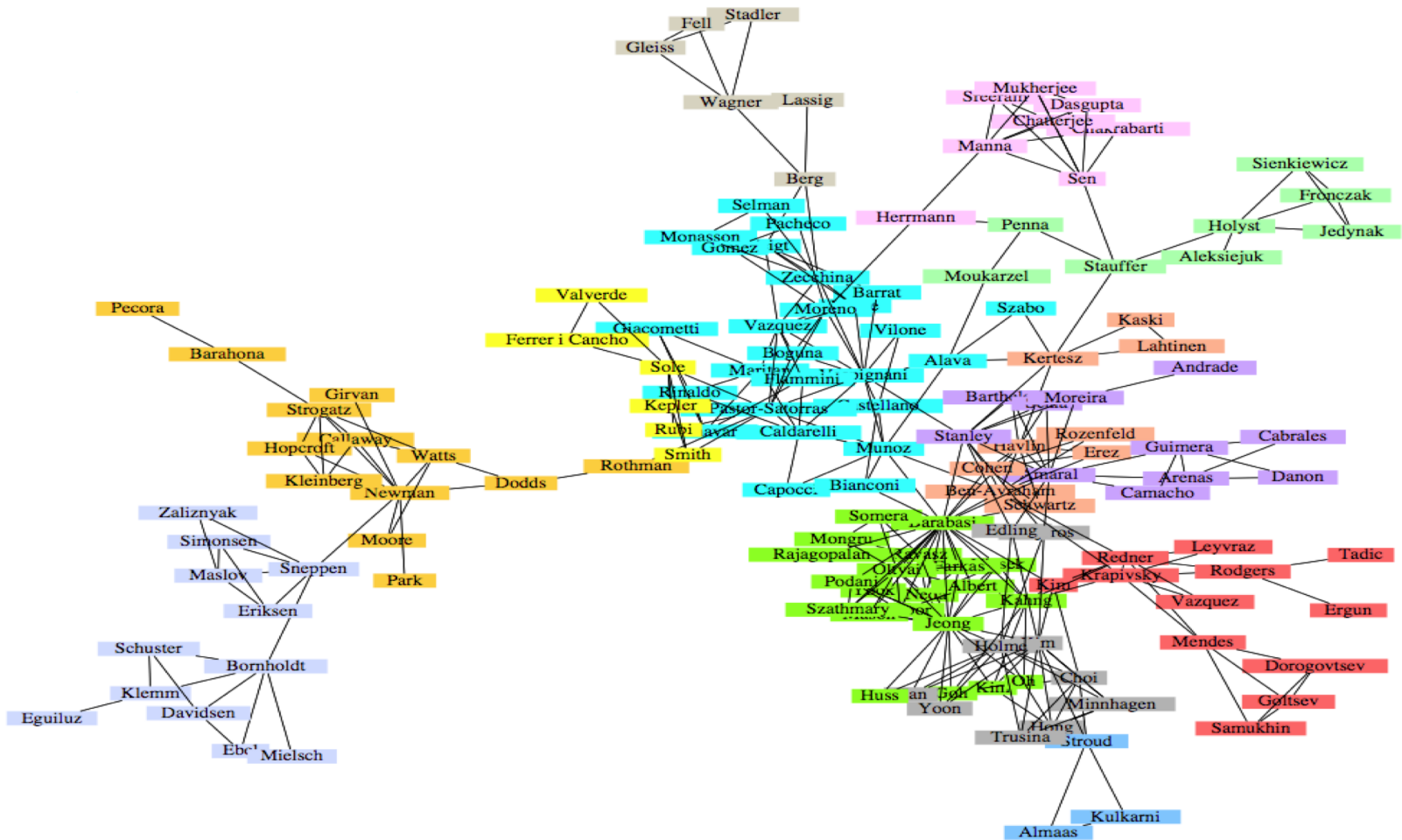
## **Strengths of ties can be predicted accurately based on various information from Facebook**

- Friend-initiated posts, message exchanged in wall post, number of mutual friends, etc.

## **Learning numeric link strength by maximum likelihood estimation**

- User profile similarity determines the strength
- Link strength in turn determines user interaction
- Maximize the likelihood based on observed profiles and interactions

# COMMUNITY DETECTION



A co-authorship network of **physicists** and **mathematicians**  
(Courtesy: Easley & Kleinberg)

# WHAT IS A COMMUNITY?

Informally: “tightly-knit region” of the network.

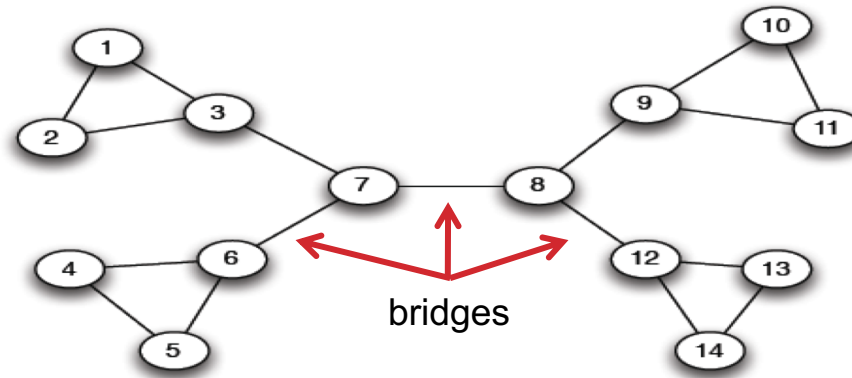
- How do we identify this region?
- How do we separate tightly-knit regions from each other?

It depends on the definition of **tightly knit**.

- Regions can be nested
- Examples ??????????
- How do bridges fit into this ??????????

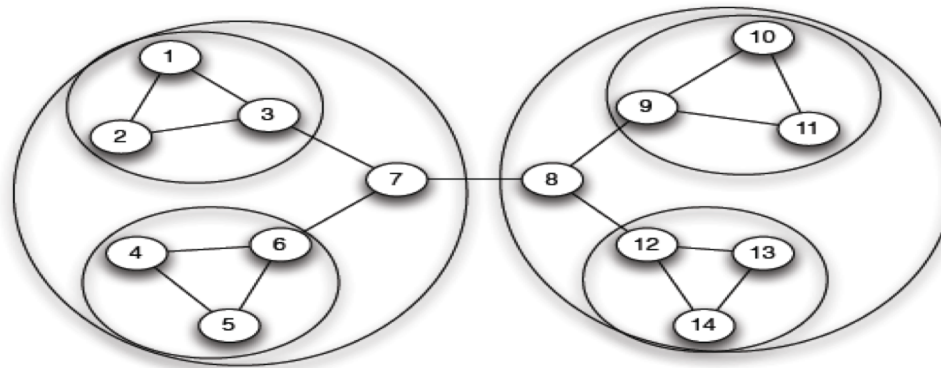


# WHAT IS A COMMUNITY?



Removal of a bridge separates the graph into disjoint components

(a) *A sample network*



(b) *Tightly-knit regions and their nested structure*

An example of a nested structure of the communities  
(Courtesy: Easley & Kleinberg)

# COMMUNITY DETECTION

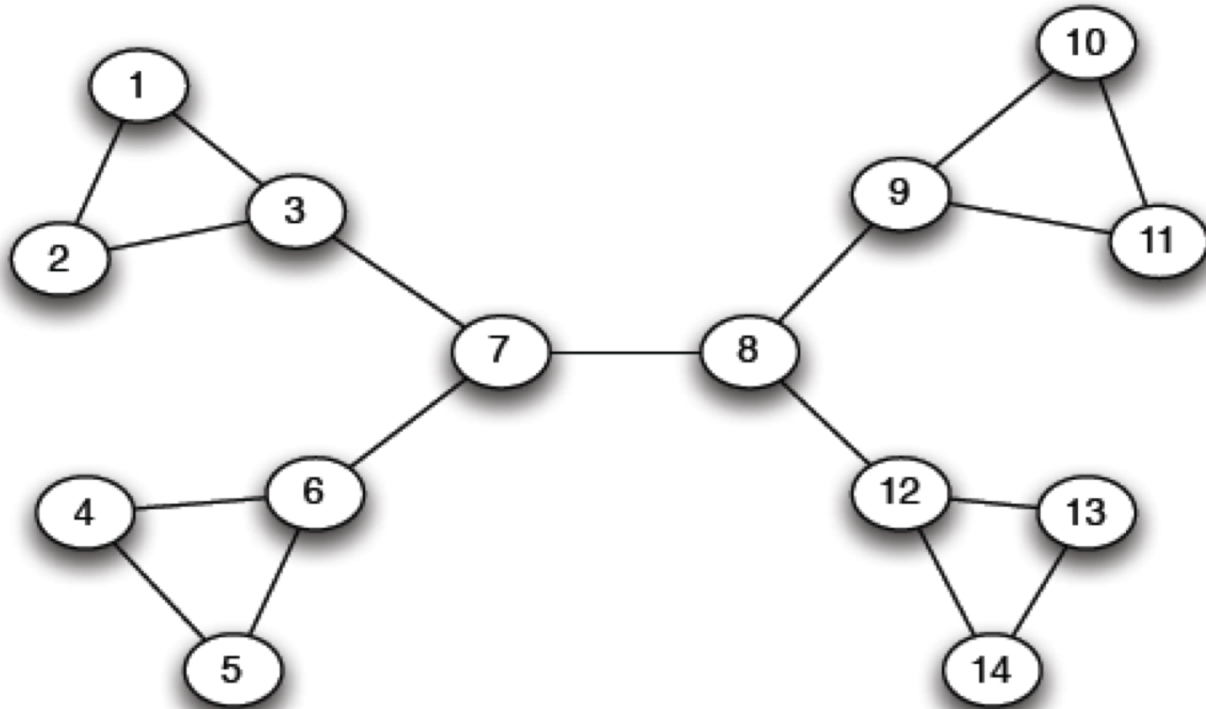
## Girvan-Newman Method

- Remove the edges of highest betweenness first.
- Repeat the same step with the remainder graph.
- Continue this until the graph breaks down into individual nodes.

**As the graph breaks down into pieces, the tightly knit community structure is exposed.**

**Results in a **hierarchical partitioning** of the graph**

# GIRVAN-NEWMAN METHOD



$$\text{Betweenness}(7-8) = 7 \cdot 7 = 49$$

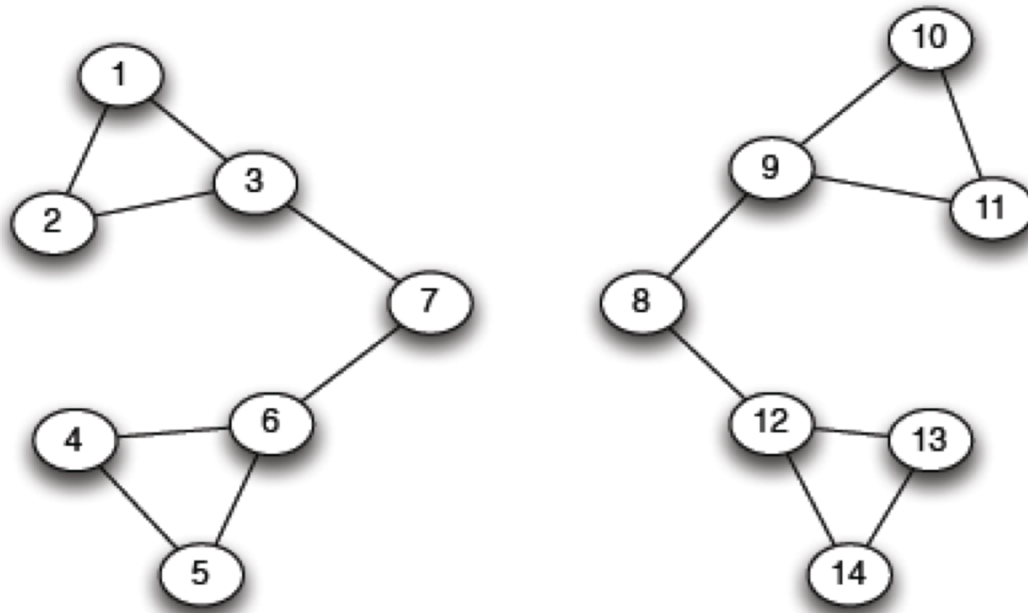
$$\text{Betweenness}(1-3) = 1 \cdot 12 = 12$$

$$\text{Betweenness}(3-7) = \text{Betweenness}(6-7) =$$

$$\text{Betweenness}(8-9) = \text{Betweenness}(8-12) = 3 \cdot 11 = 33$$



# GIRVAN-NEWMAN METHOD



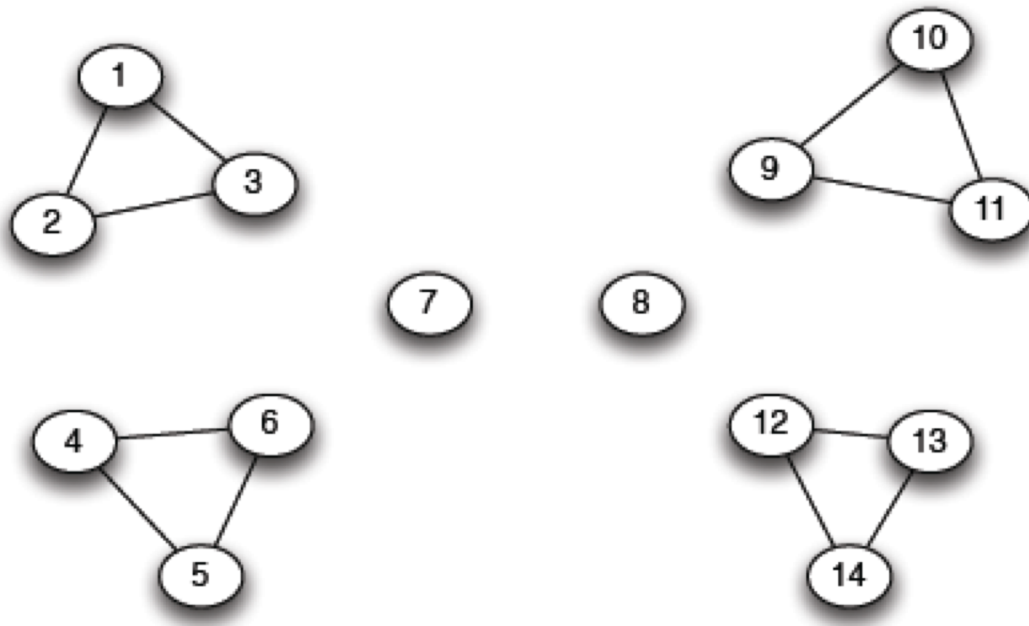
(a) *Step 1*

$$\text{Betweenness}(1-3) = 1*5=5$$

$$\text{Betweenness}(3-7) = \text{Betweenness}(6-7) =$$

$$\text{Betweenness}(8-9) = \text{Betweenness}(8-12) = 3*4 = 12$$

# GIRVAN-NEWMAN METHOD

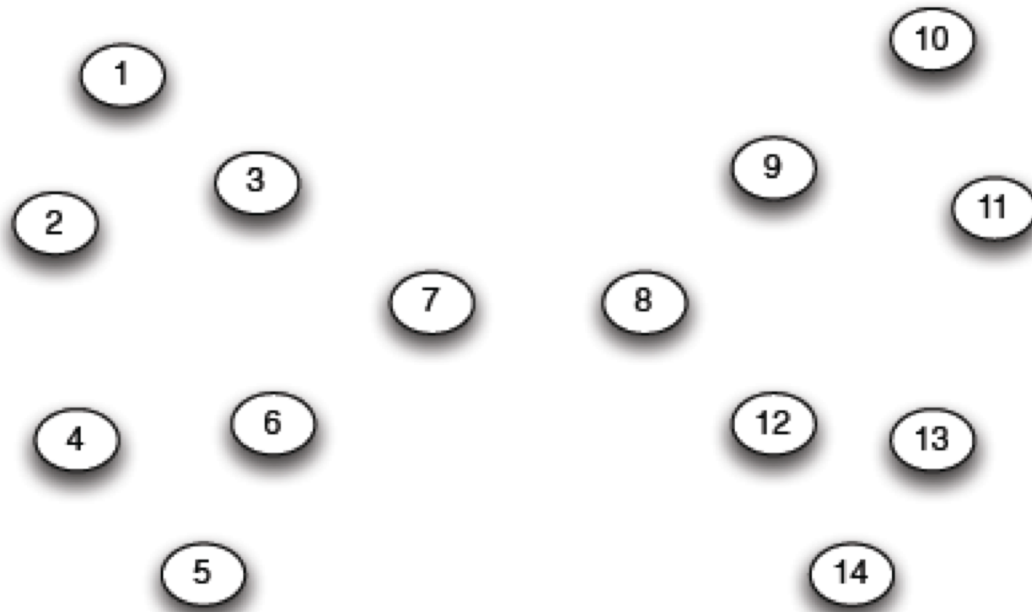


(b) *Step 2*

????????????????????

Betweenness of every edge = 1

# GIRVAN-NEWMAN METHOD



```
G=nx.Graph( )
```

```
# Returns an iterator over partitions at  
# different hierarchy levels  
nx.girvan_newman(G)
```

# NETWORKX: VIZ

Can render via Matplotlib or GraphViz

```
import matplotlib.pyplot as plt

G=nx.Graph()
nx.draw(G, with_labels=True)

# Save to a PDF
plt.savefig("my_filename.pdf")
```

Many different layout engines, aesthetic options, etc

- <https://networkx.github.io/documentation/networkx-1.10/reference/drawing.html>
- <https://networkx.github.io/documentation/development/gallery.html>

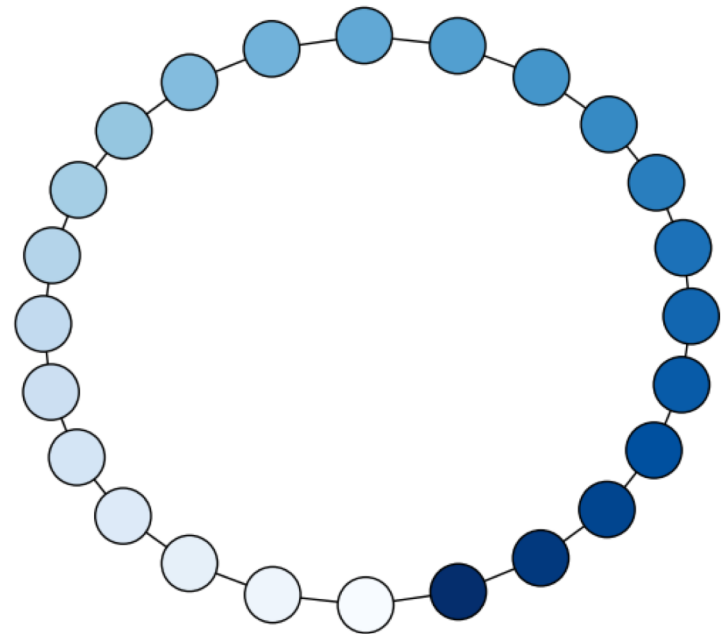
# NETWORKX: VIZ

```
# Cycle with 24 vertices
G=nx.cycle_graph(24)

# Compute force-based layout
pos=nx.spring_layout(G,
                    iterations=200)

# Draw the graph
nx.draw(G,pos,
        node_color=range(24),
        node_size=800,
        cmap=plt.cm.Blues)

# Save as PNG, then display
plt.savefig("graph.png")
plt.show()
```



# NETWORKX: VIZ

```
# Branch factor 3, depth 5
G = nx.balanced_tree(3, 5)

# Circular layout
pos = graphviz_layout(G,
                      prog='twopi', args='')

# Draw 8x8 figure
plt.figure(figsize=(8, 8))
nx.draw(G, pos,
        node_size=20,
        alpha=0.5,
        node_color="blue",
        with_labels=False)

plt.axis('equal')
plt.show()
```

