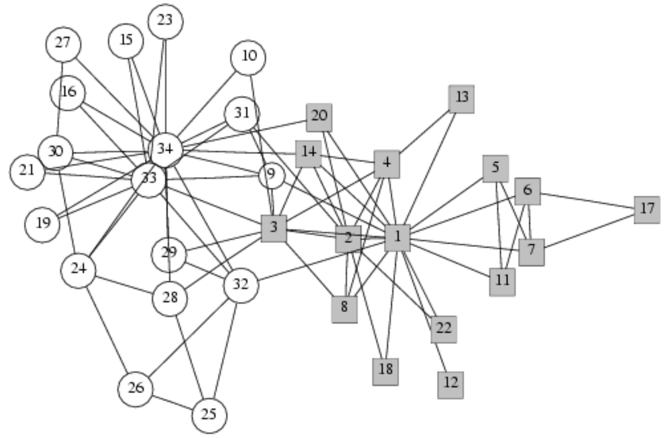
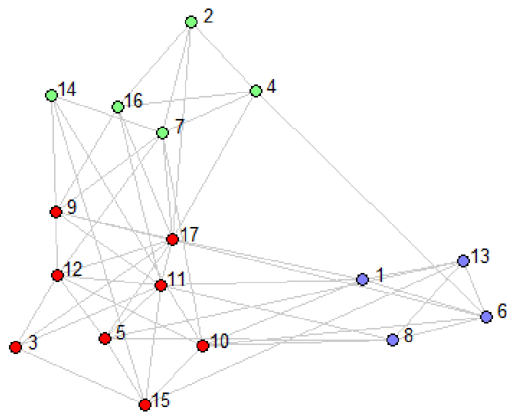




Analyzing Network Data in Python using NetworkX and DyNetworkX

SBP-BRiMS 2021 Tutorial

Tanner Hilsabeck, Hung Do, and Kevin S. Xu (University of Toledo)

Tutorial website:

https://github.com/IdeasLabUT/SBP-BRiMS-2021-NetworkX-Tutorial

About Us

Tanner Hilsabeck

- 2nd year master's student in Comp. Sci. & Engr. at University of Toledo.
- Graduate research assistant

Hung Do

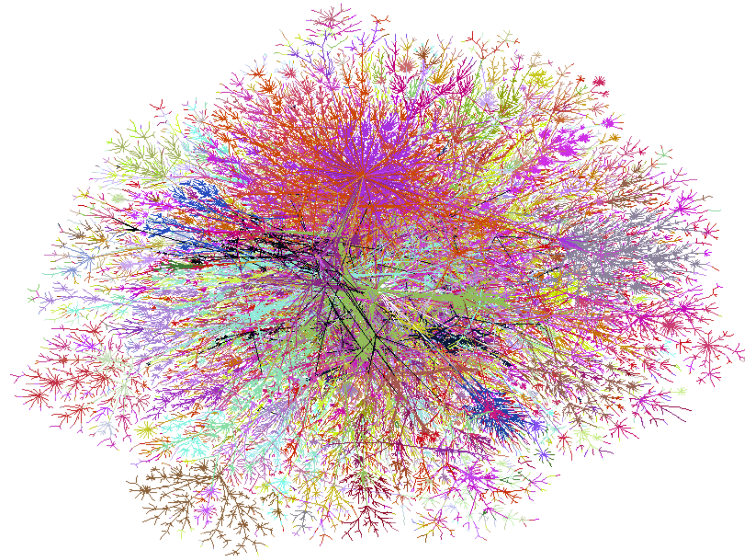
- Data Science student at University of Toledo
- Developer of DyNetworkX

Kevin S. Xu

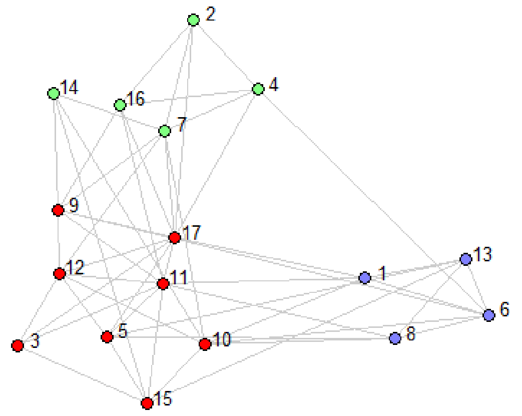
- Assistant professor at University of Toledo
- 3 years research experience in industry
- NSF CAREER award, 2021
- Research interests:
  - Machine learning
  - Network science
  - Statistical signal processing
  - Biomedical informatics

Networks are everywhere…

- Many complex physical, biological, and social systems are naturally represented by networks
- A network is represented by a graph $G = (V, E)$
    - V: vertices, nodes, or actors typically representing people or groups of people
    - E: edges, links, or ties denoting relationships between nodes
    - Nodes and edges may also contain attributes
        - Name, ID, time, location, etc.
- Networks may change over time: dynamic or temporal networks
- Analyzing network representations of complex systems can reveal many insights about how they are organized



The Internet
(Burch and Cheswick, 1998)



Dynamic social network
(Nordlie, 1958; Newcomb, 1961)

Outline

Part 1: Analyzing Network Data in Python (~75 min)

- Measures of network structure
  - Centrality, transitivity, homophily
- Introduction to Python and NetworkX
  - Network analysis using Python
- Python demo 1: Centrality analysis
- Common network analysis tasks
  - Community detection
  - Link prediction
- Python demo 2: Link prediction

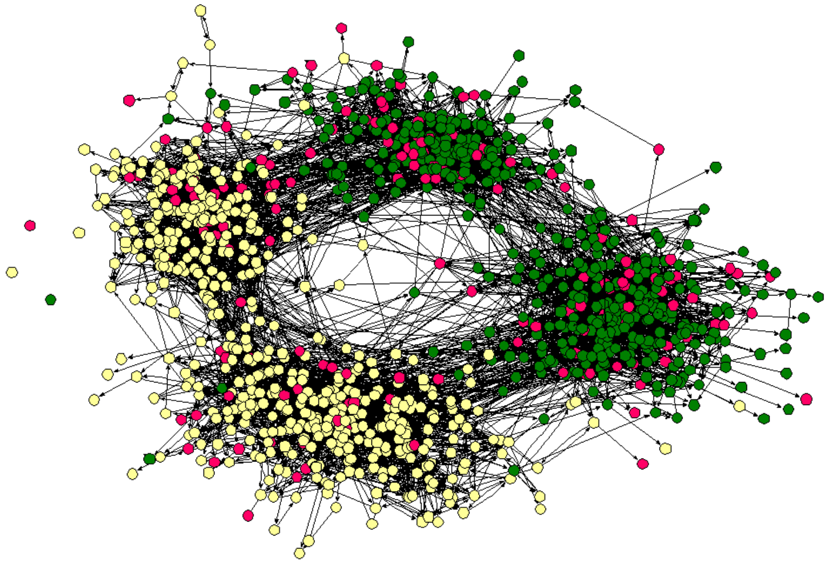
Part 2: Dynamic Networks in Python (~60 min)

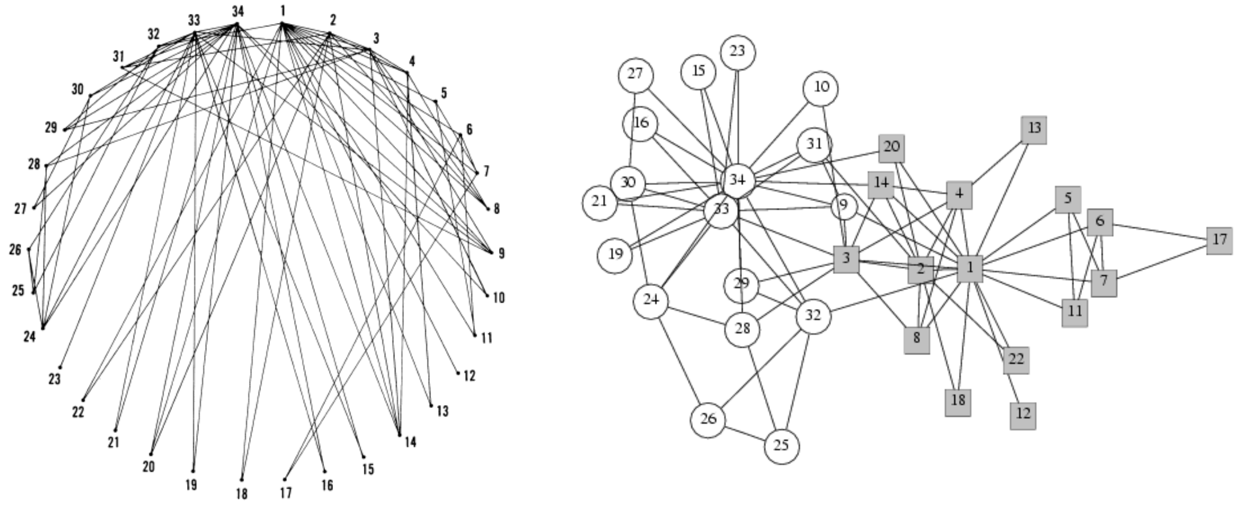
- Different representations of dynamic networks
  - Discrete-time snapshots (panel data)
  - Continuous-time relational event data
- Introduction to DyNetworkX
  - Data structures and algorithms for dynamic network analysis in Python
- Python demo 3: Dynamic centrality analysis
- Motifs in dynamic networks
- Python demo 4: Temporal motif analysis

Part 1 : Analyzing Network Data in Python

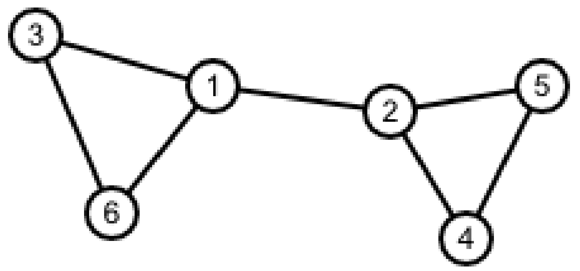
Visual analysis of networks

- Visualization can often provide some insights about networks
  - Example: separation of school friendships by race
- Graphs have no natural representation in a geometric space
  - Two identical graphs drawn differently
  - Moral: insights may be different depending on the type of visualization
  - Are these insights about the network itself or the visualization algorithm?
- Gaining insights from dynamic network visualizations (animations) is even more challenging
- Need more advanced analysis tools ➔ focus of this tutorial


School friendships (Moody, 2001)


Zachary's Karate club (Zachary, 1977)

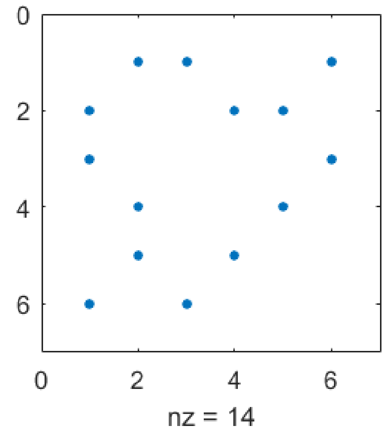
Mathematical representations of graphs

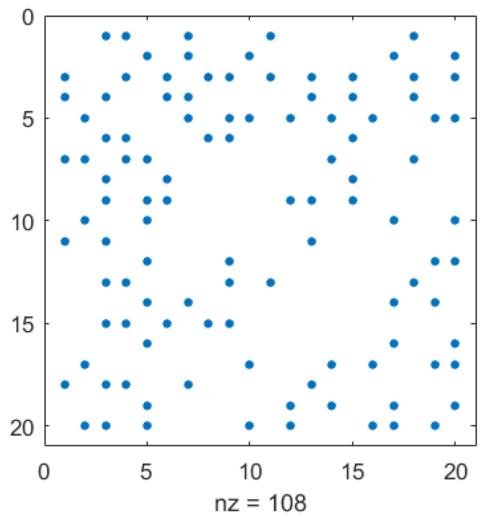


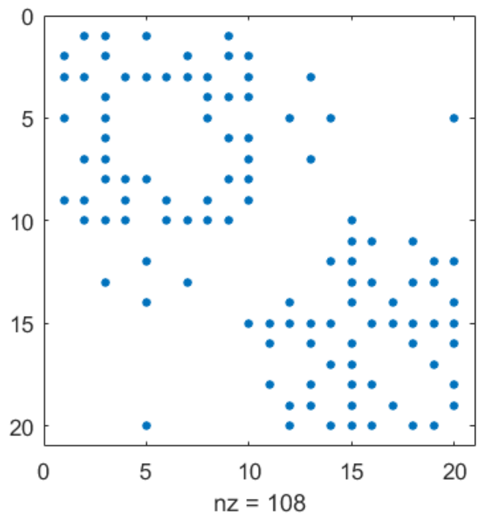
$n = 6$
$m = 7$

- Number of nodes: n
- Number of edges: m
- Represent graph by $n \times n$ adjacency matrix A
  - $a_{ij} = 1$ if there is an edge between nodes i and j
  - $a_{ij} = 0$ otherwise
  - Easily extended to directed and weighted graphs
- Row and column permutations do not change graph

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







Outline

Part 1: Analyzing Network Data in Python (~75 min)

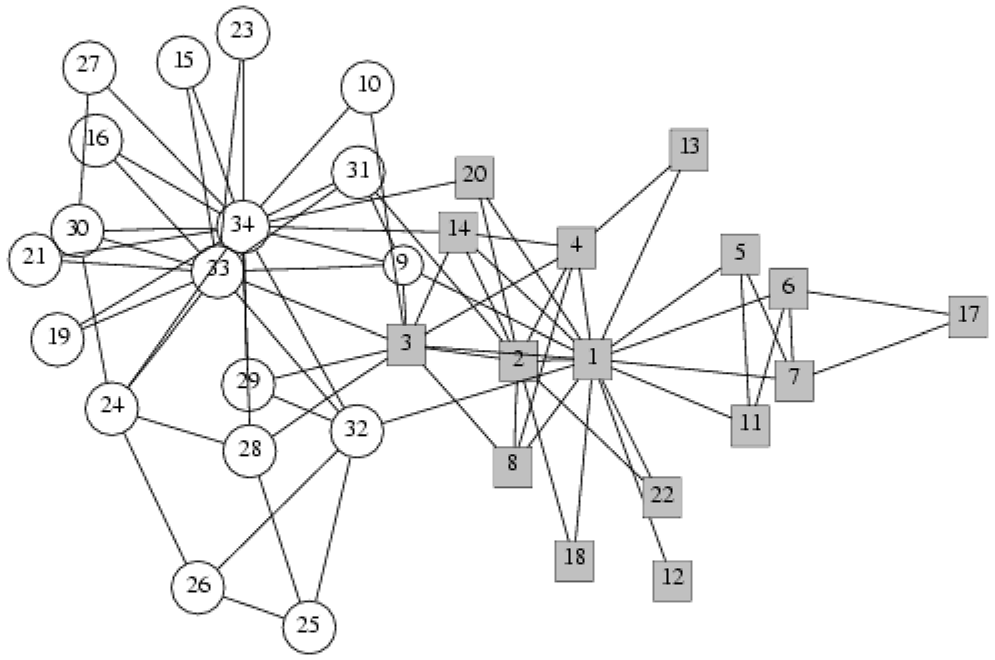
- Measures of network structure
  - Centrality, transitivity, homophily
- Introduction to Python and NetworkX
  - Network analysis using Python
- Python demo 1: Centrality analysis
- Common network analysis tasks
  - Community detection
  - Link prediction
- Python demo 2: Link prediction

Part 2: Dynamic Networks in Python (~60 min)

- Different representations of dynamic networks
  - Discrete-time snapshots (panel data)
  - Continuous-time relational event data
- Introduction to DyNetworkX
  - Data structures and algorithms for dynamic network analysis in Python
- Python demo 3: Dynamic centrality analysis
- Motifs in dynamic networks
- Python demo 4: Temporal motif analysis

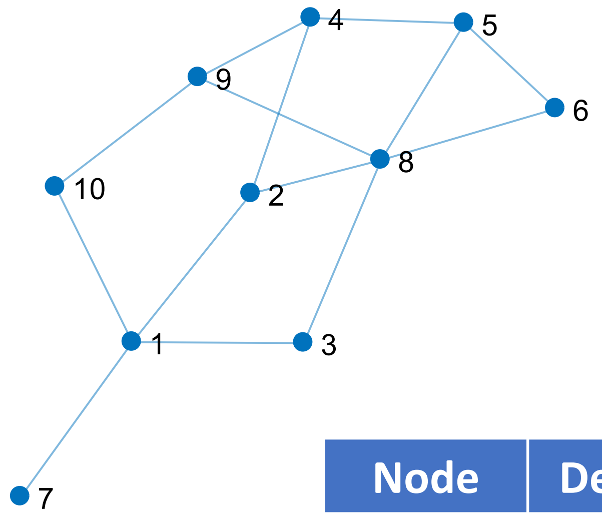
Centrality

- Who are the most important or central nodes in a network?
- Many measures of centrality have been proposed
  - Degree centrality
  - Eigenvector centrality and variants (e.g. PageRank)
  - Closeness centrality
  - Betweenness centrality
- Different measures correspond to different meanings of importance



Degree centrality

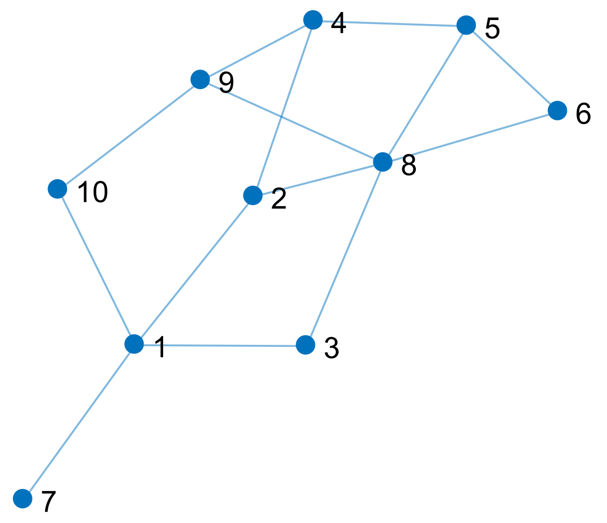
- Simplest centrality measure: degree centrality
  - Degree of a node: # of edges connected to it
  - Also called # of neighbors
- Sometimes normalized by maximum possible degree $n-1$
  - Max. normalized degree centrality is 1
- Nodes in a directed network have different in-degrees and out-degrees
  - Typically use in-degree for directed network
- Weakness of degree centrality: it assumes all neighbors are equally important!



Node	Degree
1	4
2	3
3	2
4	3
5	4
6	2
7	1
8	5
9	3
10	2

Eigenvector centrality

- Basic idea: give nodes credit for having edges with other important nodes
  - But how do we do this when we don't know how important other nodes are?
- Iterative construction of eigenvector centrality
  1. Initialize each node with centrality $1/n$
  2. For each node, set centrality to sum of centralities of all neighbor nodes
  3. Normalize all centralities to sum to 1
  4. Repeat steps 2 and 3 until centralities converge
- Centralities converge to leading eigenvector of graph adjacency matrix
- PageRank centrality: variant of eigenvector centrality designed for directed networks



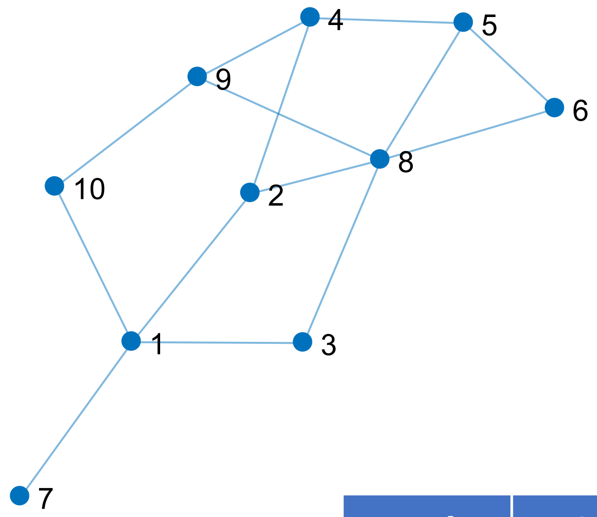
$$\begin{array}{ccc}
 & \text{Iterations} \rightarrow & \text{Degree} \\
\text{Nodes} \downarrow & \begin{bmatrix}
0.10 & 0.14 & \dots & 0.10 & 0.10 \\
0.10 & 0.11 & \dots & 0.12 & 0.12 \\
0.10 & 0.07 & \dots & 0.08 & 0.09 \\
0.10 & 0.11 & \dots & 0.11 & 0.11 \\
0.10 & 0.11 & \dots & 0.12 & 0.12 \\
0.10 & 0.07 & \dots & 0.09 & 0.09 \\
0.10 & 0.04 & \dots & 0.03 & 0.03 \\
0.10 & 0.18 & \dots & 0.17 & 0.17 \\
0.10 & 0.11 & \dots & 0.11 & 0.11 \\
0.10 & 0.07 & \dots & 0.07 & 0.07
\end{bmatrix} & \begin{array}{c} 4 \\ 3 \\ 2 \\ 3 \\ 4 \\ 2 \\ 1 \\ 5 \\ 3 \\ 2 \end{array}
\end{array}$$

Closeness centrality

- Basic idea: a central node should have short distances to other nodes
- Length of geodesic (shortest) path between 2 nodes is usually used as distance measure
  - Call this geodesic distance d_{ij}
- Shorter paths ➔ higher centrality
- Closeness centrality definition

$$C_i = \frac{n-1}{\sum_j d_{ij}}$$

- Has problems with multiple components in a network and low dynamic range



Closeness centrality of node 1:

$$C_1 = \frac{10-1}{4(1)+3(2)+2(3)} = \frac{9}{16}$$

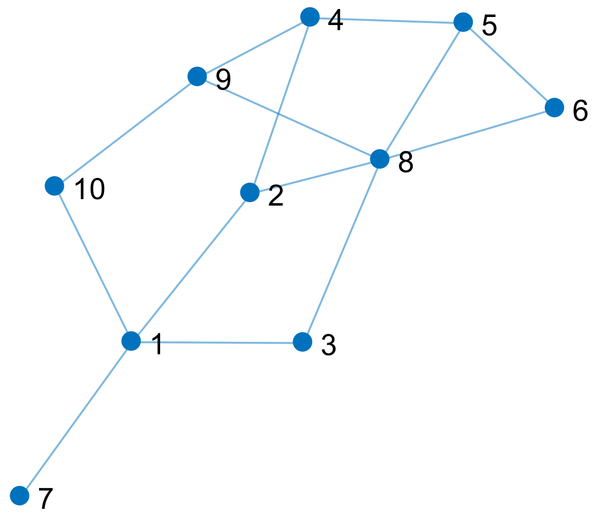
Node	Distance
2	1
3	1
4	2
5	3
6	3
7	1
8	2
9	2
10	1

Betweenness centrality

- Basic idea: central nodes should lie on many paths between nodes
  - These nodes may serve as "bottlenecks" between groups of nodes
- Betweenness centrality definition

$$x_i = \sum_{s<t} \frac{n_{st}^i}{g_{st}}$$

  - Sum over all $n(n-1)/2$ node pairs
  - n_{st}^i: number of geodesic paths $s \to t$ through node i
  - g_{st}: total number of geodesic paths $s \to t$
- Sometimes normalized to max of 1



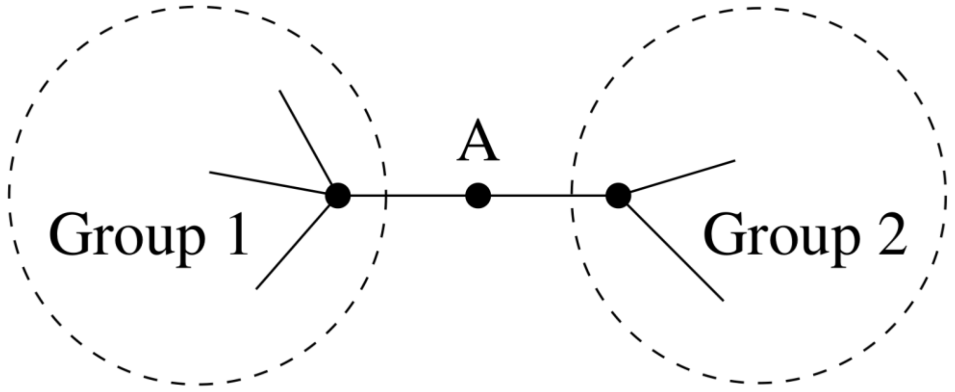
To compute term inside summation for $i = 2$ and $(s, t) = (1,6)$:
2 shortest paths from 1 to 6:
- 1 → 2 → 8 → 6
- 1 → 3 → 8 → 6

$$n_{st}^i = 1$$
$$g_{st} = 2$$
$$\Rightarrow \frac{n_{st}^i}{g_{st}} = \frac{1}{2}$$

Repeat for all $s < t$ to compute betweenness centrality for node 2

Comparison of centrality measures

- Assumptions of centrality measures
  - Degree centrality: important nodes have many edges
  - Eigenvector centrality: important nodes have edges to other important nodes
  - Closeness centrality: important nodes are close to other nodes
  - Betweenness centrality: important nodes are hops between other nodes
- Centrality measures should be interpreted in a relative manner
  - Different definitions and software packages uses different normalizations



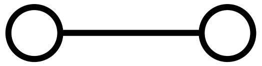
Example from Newman (2018):
Node A can have low degree, eigenvector, and closeness centralities but high betweenness centrality!

Newman, M. (2018). *Networks (2nd edition): Oxford university press.*

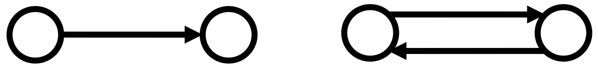
Network motifs

- Motifs or graphlets: subgraphs of a larger graph that frequently appear
- Typically consider 2 and 3-node motifs
  - Can extend analysis to motifs with more than 3 nodes
- Frequencies of observing motifs provide some insight to network structure
  - Overexpressed motif appears more often than expected by chance

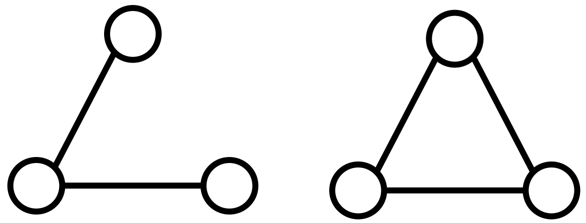
2-node undirected motifs:



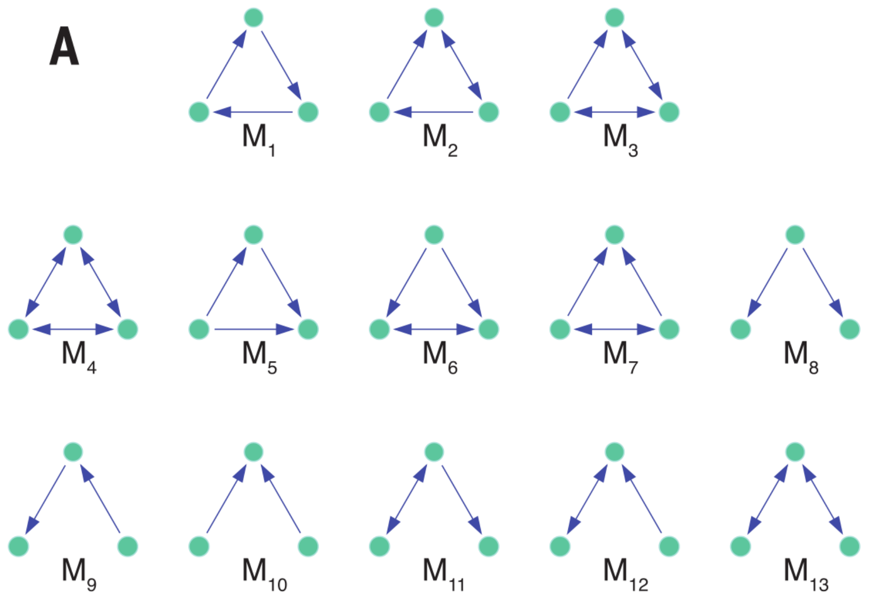
2-node directed motifs:



3-node undirected motifs:

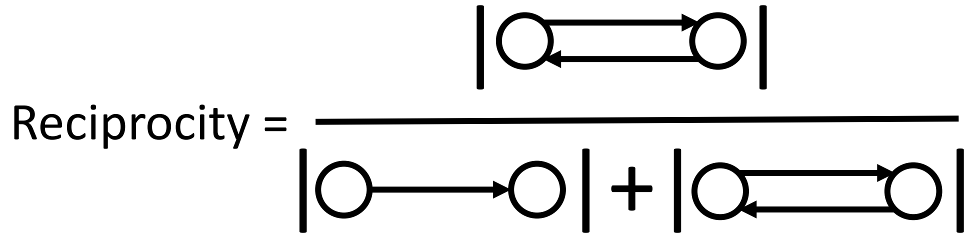


3-node directed motifs (Benson et al., 2016):

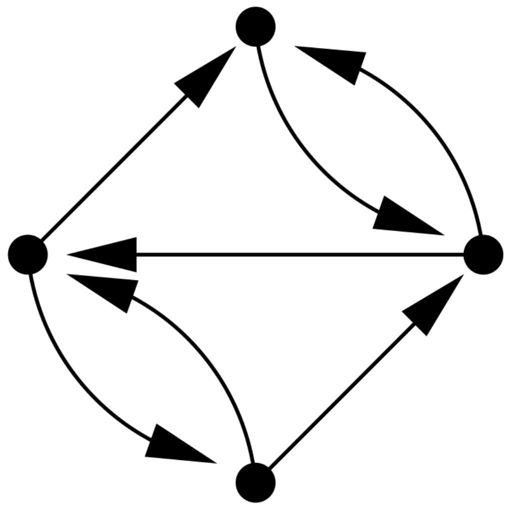


Reciprocity

- Applies only to directed graphs
- Measures frequency of reciprocal (mutual) relationships
- Reciprocity: fraction of all edges that are reciprocated
    - Relative frequency of 2-node loop motif compared to all 2-node motifs
    - Double count each loop!
- Can also define reciprocity for a single node



Simple example from Newman (2018):

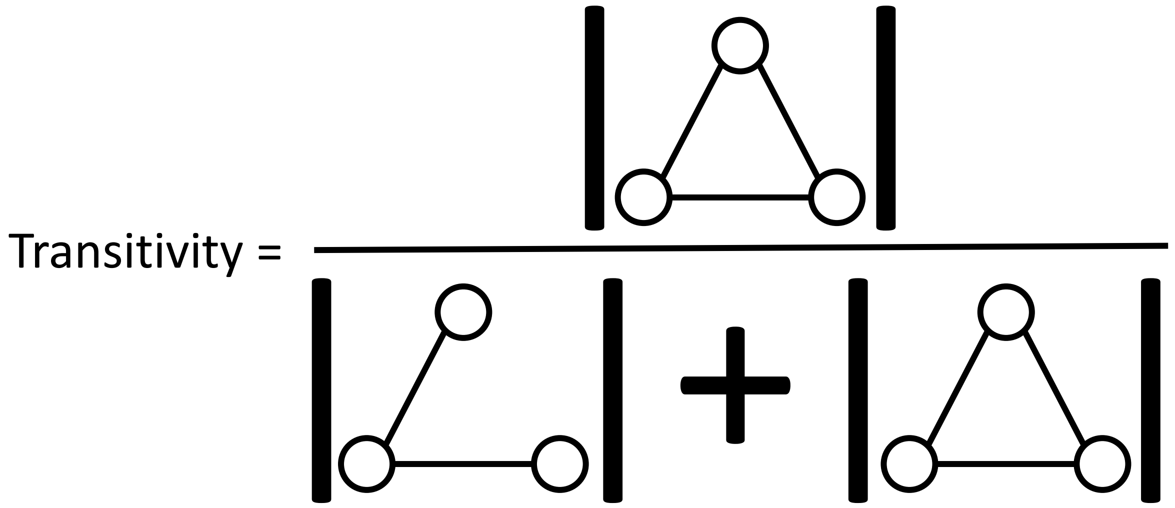


$$\text{Reciprocity} = \frac{4}{7}$$

Newman, M. (2018). *Networks (2nd edition): Oxford university press.*

Transitivity

- If $a \circ b$ and $b \circ c$ implies that $a \circ c$ then the relation $\circ$ is transitive
- Partial transitivity or “clustering” is common in social networks
- Transitivity: fraction of all possible triangles in graph
  - Possible triangle contains 2 edges of a triangle
  - Triple count each triangle!
  - Also known as (global) clustering coefficient
- (Local) clustering coefficient: fraction of all possible triangles involving a node u
  - Average local clustering coefficient is sometimes used as a global clustering coefficient

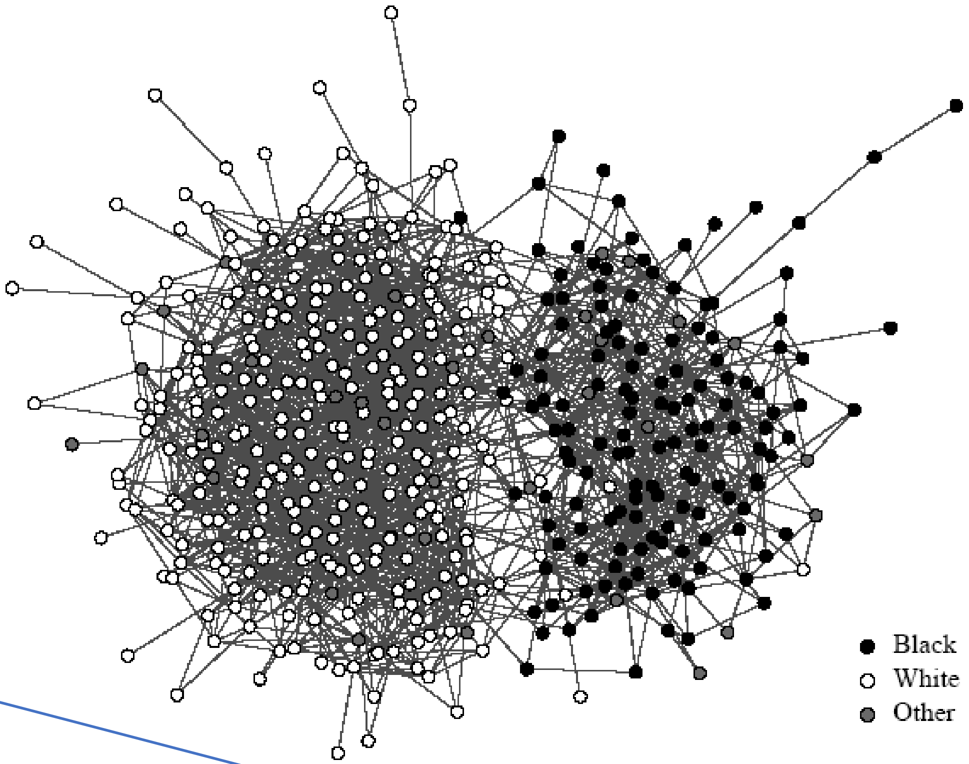


Homophily and assortative mixing

- In many networks, nodes form edges with other similar nodes
  - Called homophily or assortative mixing
- Most common setting: assortative mixing by some group or type g
  - Example: Add Health friendship network by race (Newman, 2018)
  - Some networks show disassortative mixing (e.g. heterosexual relationships by sex)
- Modularity: measure of assortative mixing

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{\deg(i)\deg(j)}{2m} \right) \delta_{g_i g_j}$$

  - Difference between actual # of edges within same group and expected # of edges
- Higher Q ➔ more assortative mixing



1 if i & j are in same group, 0 otherwise

Newman, M. (2018). *Networks (2nd edition): Oxford university press.*

Outline

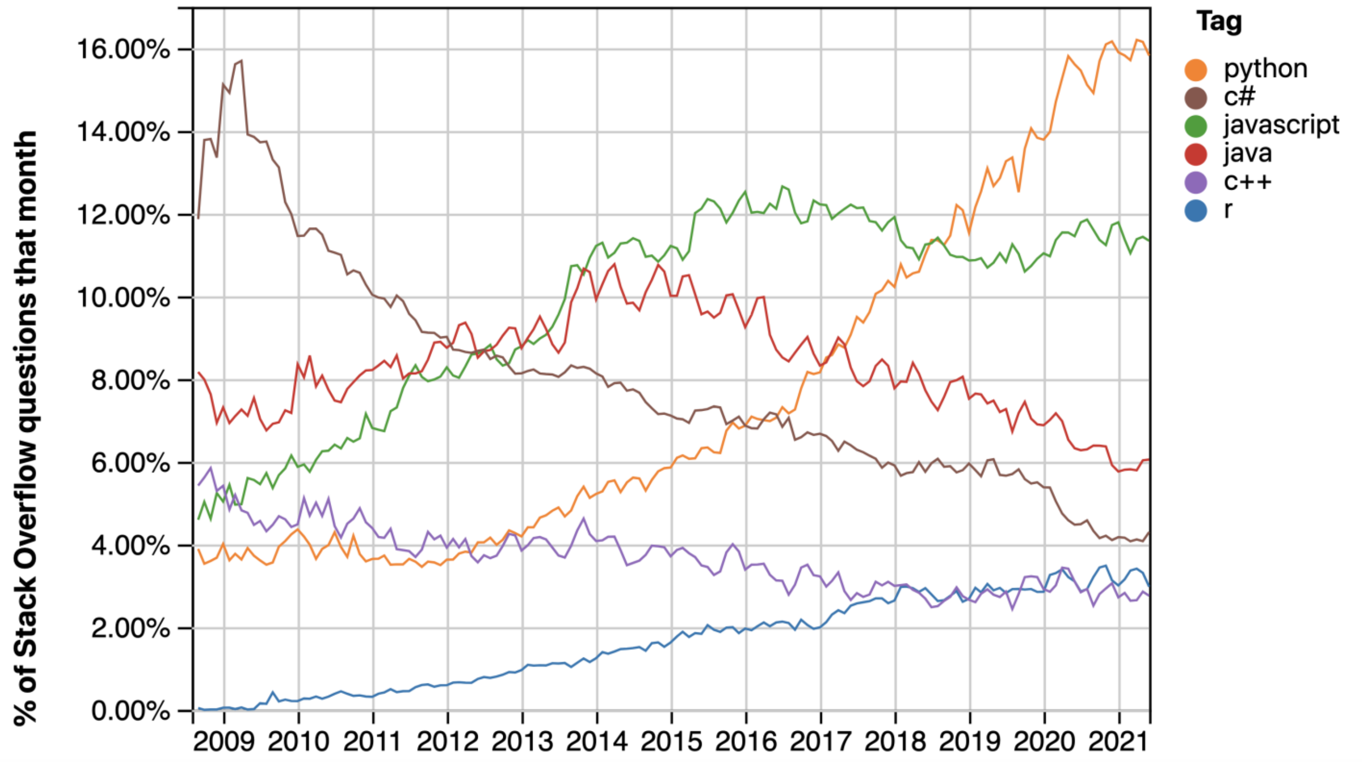
Part 1: Analyzing Network Data in Python (~75 min)

- Measures of network structure
  - Centrality, transitivity, homophily
- Introduction to Python and NetworkX
  - Network analysis using Python
- Python demo 1: Centrality analysis
- Common network analysis tasks
  - Community detection
  - Link prediction
- Python demo 2: Link prediction

Part 2: Dynamic Networks in Python (~60 min)

- Different representations of dynamic networks
  - Discrete-time snapshots (panel data)
  - Continuous-time relational event data
- Introduction to DyNetworkX
  - Data structures and algorithms for dynamic network analysis in Python
- Python demo 3: Dynamic centrality analysis
- Motifs in dynamic networks
- Python demo 4: Temporal motif analysis

Introduction to Python



(Stack Overflow Insights, 2021)

First released in 1991 by Guido van Rossum.

The Pillars of Python:

- Easy and Intuitive
- Open source
- Understandable in plain English
- Suitable for everyday tasks

(Python Institute, 2021)

Stack Overflow Insights, 2021 - https://insights.stackoverflow.com/trends?tags=r%2Cpython%2Cjavascript%2Cjava%2Cc%2B%2B%2Cc%23
Python Institute 2021 - https://pythoninstitute.org/what-is-python/

Easy and Intuitive

C++:

```
#include <iostream>
int main() {
    std::cout << "Hello World!";
}
```

Java:

```
import java.io.*;
public static void main(String[] args) {
    System.out.println("Hello World!");
}
```

Python:

```
print("Hello World!")
```

Python is designed to be easy and intuitive.

This makes it an excellent first language of choice.

No semicolons.
No type declaration.
No pointers.

Open Source

Example data analysis stack:

Input and Data Storage:

Pandas – provides methods to read data from formats such as: CSV, JSON, HTML, Excel, SQL and more. Stores data as a Dataframe object.

NetworkX – provides methods to read data from formats such as: Adjacency List, Edge List, GML, and JSON. Stores datas Graph objects.

Modeling and Computations:

Scikit-learn – commercial-grade machine learning and data analysis suite. Includes variety of models for classification, regression, clustering, and more.

Plotting and Visuallization:

MatplotLib – capable of producing static, animated, and interactive plots. Serves as the backbone for many other plotting packages.

"Information wants to be free" Stewart Brand (1984).

Thousands of developers increase the capabilities of the language with small independent libraries called packages.

Understandable in Plain English

Pythonic:

```
with open("data_file.txt", "r") as file:
    for line in file:
        print(line)
```

Not Pythonic:

```
file = open("data_file.txt", "r")
lines = file.readlines()
i = 0
while i < len(lines):
    print(line)
file.close()
```

Python's easy of use does not stop at its intuitive syntax.

The Python community encourages developers to write in a "pythonic" format that is easy to understand.

Suitable for Everyday Tasks

The lack of declarations makes Python the perfect language of choice for rapid prototyping.

```
def computeSpeed(distances, times):
    if not isInstance(distances, Iterable) and not isInstance(times, Iterable):
        return distances/times
    if len(distances) != len(times):
        raise IndexError("Distances and Times must be the same length!")
    return [distances[i]/times[i] for i in range(len(distances))]
```

Introduction to NetworkX

NetworkX is the most popular package associated with networks for Python.

NetworkX is often included in package managers, such as Anaconda.

Comprised of four classes, NetworkX is able to store and compute metrics across a wide variety of networks.

		Directed?	
		Yes	No
Multiple Edges?	Yes	MultiDiGraph	MultiGraph
	No	DiGraph	Graph

Introduction to NetworkX

Some of the most common NetworkX methods include:

```
# Add a single edge between u and v with optional attributes
G.add_edge(u_of_edge, v_of_edge, **attr)

# Find number of edges or number of nodes
G.number_of_edges()
G.number_of_nodes()

# Iterate over edges or nodes
for edge in G.edges(): # All edges
for edge in G.edges([1, 2, 3]): # All edges connected to nodes 1, 2 or 3.
for node in G.nodes():
```

Introduction to NetworkX

Some of the most common NetworkX methods include:

```
# Calculate degree of nodes
G.degree() # All nodes
G.degree([1, 2, 3]) # Only nodes 1, 2, and 3

# Example Centralities
G.degree_centrality()
G.betweenness_centrality()

# Calculate components
nx.algorithms.components.connected_components(G)
```

Other network analysis packages in Python

- Many other Python packages for working with graphs and networks
  - graph-tool
  - igraph
  - networkit
  - SNAP
- Almost all written in C or C++ but have Python APIs

- NetworkX is written in Python
  - Easy to learn and use
  - Easy to contribute to
  - Largest community of developers
  - At least 10x slower than C/C++ based packages
  - Scales to networks of ~100,000 nodes

Benchmark of popular graph/network packages v2, 2021 - https://www.timlrx.com/blog/benchmark-of-popular-graph-network-packages-v2

Outline

Part 1: Analyzing Network Data in Python (~75 min)

- Measures of network structure
    - Centrality, transitivity, homophily
- Introduction to Python and NetworkX
    - Network analysis using Python
- Python demo 1: Centrality analysis
- Common network analysis tasks
    - Community detection
    - Link prediction
- Python demo 2: Link prediction

Part 2: Dynamic Networks in Python (~60 min)

- Different representations of dynamic networks
    - Discrete-time snapshots (panel data)
    - Continuous-time relational event data
- Introduction to DyNetworkX
    - Data structures and algorithms for dynamic network analysis in Python
- Python demo 3: Dynamic centrality analysis
- Motifs in dynamic networks
- Python demo 4: Temporal motif analysis

Demo 1: Centrality analysis using NetworkX

Zachary’s Karate Club network

Outline

Part 1: Analyzing Network Data in Python (~75 min)

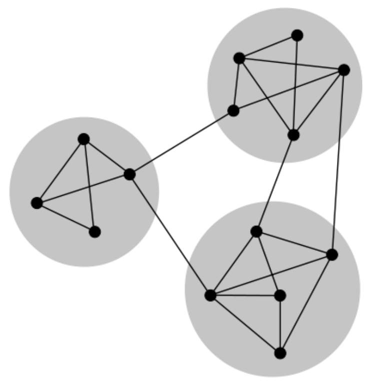
- Measures of network structure
  - Centrality, transitivity, homophily
- Introduction to Python and NetworkX
  - Network analysis using Python
- Python demo 1: Centrality analysis
- Common network analysis tasks
  - Community detection
  - Link prediction
- Python demo 2: Link prediction

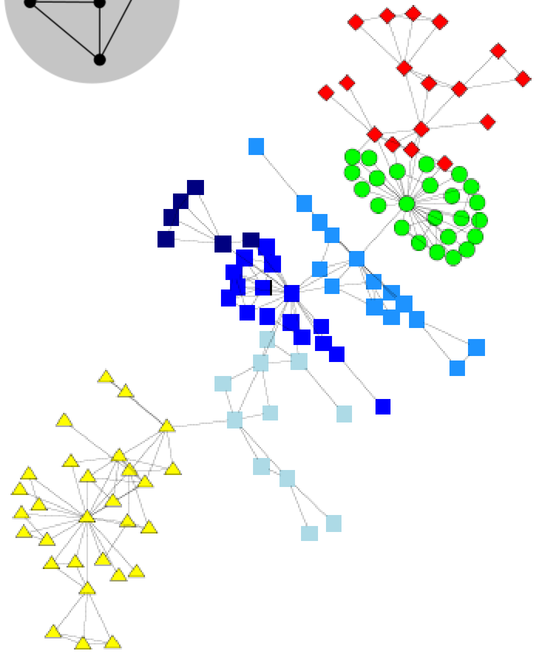
Part 2: Dynamic Networks in Python (~60 min)

- Different representations of dynamic networks
  - Discrete-time snapshots (panel data)
  - Continuous-time relational event data
- Introduction to DyNetworkX
  - Data structures and algorithms for dynamic network analysis in Python
- Python demo 3: Dynamic centrality analysis
- Motifs in dynamic networks
- Python demo 4: Temporal motif analysis

Community detection

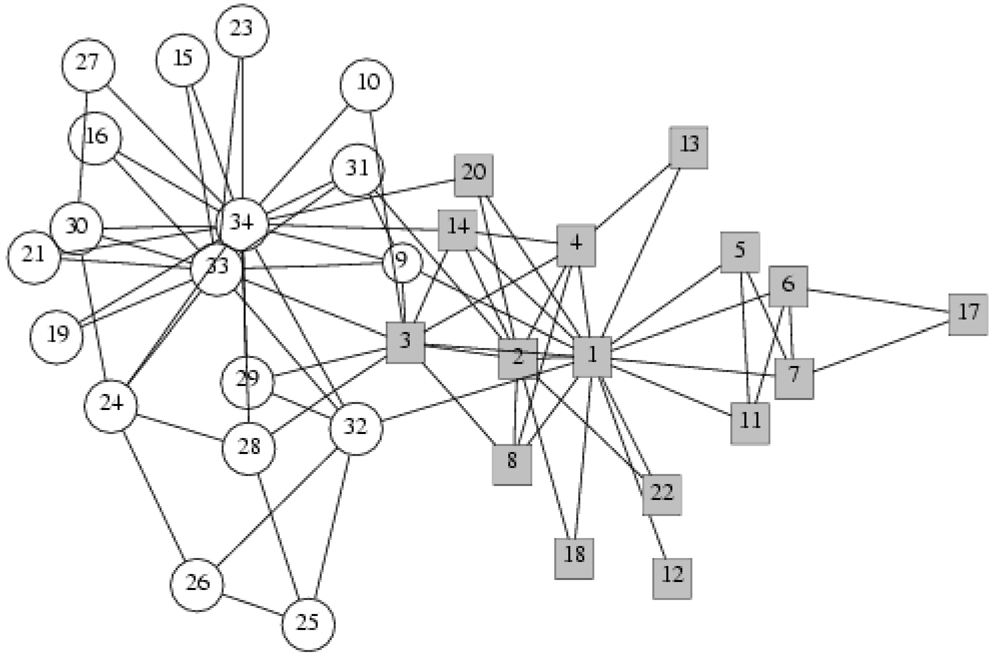
- Many networks have community structure
  - Division of nodes into groups such that there are
    - Many edges within groups
    - Not many edges between groups
- Community structure helps us understand organization of the network
- How can we detect these communities from just the network structure?





Modularity maximization

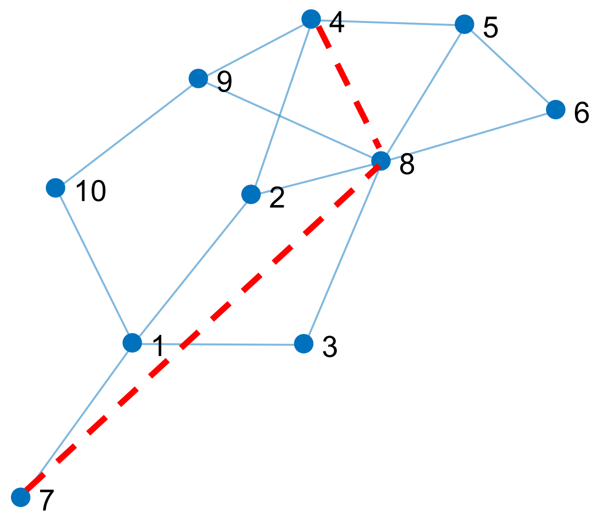
- Recall modularity: measure of assortative mixing
$$Q = \frac{1}{2m}\sum_{ij}\left(A_{ij} - \frac{\deg(i)\deg(j)}{2m}\right)\delta_{g_i g_j}$$
  - But we don't know the groups or communities!
- Modularity maximization: assign nodes to groups to maximize Q
  - NP-hard problem!
  - Lots of heuristics proposed to approximately maximize Q
- Greedy maximization algorithm (Clauset et al., 2004) correctly recovers community structure in Zachary's Karate club network



Clauset, A., Newman, M. E., & Moore, C. Finding community structure in very large networks. *Physical Review E 70*(6), 2004.

Link prediction

- Link prediction: prediction of pairs of nodes that don't have a link (edge) but ought to have a link
- Applications of link prediction
  - Recommendation in on-line social networks
    - Key element in People You May Know (LinkedIn and Facebook), Who to Follow (Twitter), etc.
  - Imputation of missing data
    - In a partially observed network, which unobserved pairs of nodes are most likely to have an edge?
  - Validation of models for network formation
    - If a model for network formation is accurate, it should generate accurate predictions of edges that actually form in the future

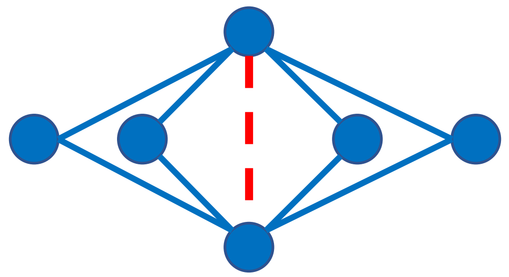


Link prediction similarity scores

- For any two nodes (u, v) without a link, compute a “similarity score” s_{uv}
- What is a good way to calculate similarity between two nodes?
  - Assortative mixing: use groups or types if we have them
    - But we don’t always have this!
  - Use graph structure itself!
- Preferential attachment: popular nodes are likely to form links

$$s_{uv} = \deg(u)\deg(v)$$

- Common neighbors: two people with no link but lots of common neighbors are good candidates to form a link
  - Forming such a link closes many triangles



- Similarity between people u and v = number of common neighbors between u and v

More similarity scores

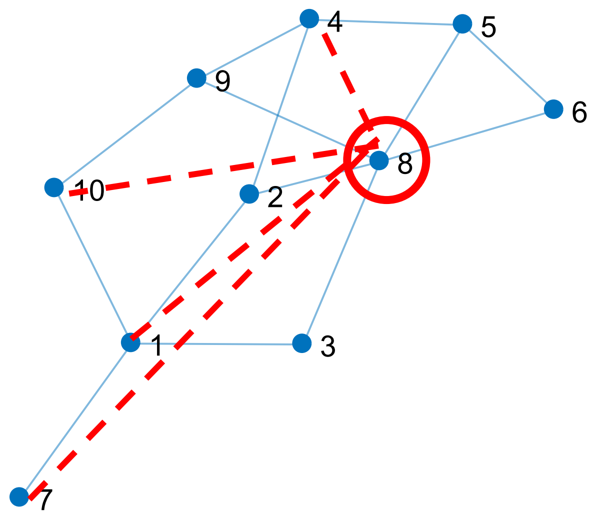
- Jaccard coefficient:

$$s_{uv} = \frac{\text{\# of common neighbors between } u, v}{\text{max. possible \# of common neighbors between } u, v}$$

- Adamic-Adar: high-degree common neighbors are less useful than low-degree common neighbors for link prediction
    - First discovered by Adamic and Adar in 2003 at HP
    - Recall: degree of a node is the number of neighbors (connections, friends, etc.) the node has
    - Example: two people having the town mayor as a common friend should be less similar than two people having some obscure common friend
- Similarity between nodes u and v is sum of $1/\log[\deg(w)]$ over all common neighbors w
    - Log factor for slower decay

Demonstration of Adamic-Adar

- Adamic-Adar similarity scores for node 8:
  - Node 1: $1/\log 3 + 1/\log 2 = 5.4$
  - Node 4: $1/\log 3 + 1/\log 3 + 1/\log 3 = 6.3$
  - Node 7: 0
  - Node 10: $1/\log 3 = 2.1$
- Recommend node 4 most frequently!
  - Same as common neighbors recommendation in this case



Evaluating link prediction accuracy

- Compute similarity score (prediction) s_{uv} for all (u, v) without an edge
- Compare predictions to future network
  - Let m' denote # of new edges formed
  - Choose m' highest similarity scores s_{uv} and check how many (u, v) node pairs formed edges
- Good link predictor typically has $\sim 10\%$ accuracy!
  - Compare to random guessing: expected accuracy is typically very low $\sim 0.2\%$
- If no future network is available, evaluate using cross validation
  - Randomly remove m' edges from the network and treat them as “new” edges
  - Average prediction accuracy over multiple random removals

Predictions $m' = 5$	Formed edge?
4	☑
3	☒
2.5	☑
2	☒
1.5	☒

Accuracy = 2/5

Outline

Part 1: Analyzing Network Data in Python (~75 min)

- Measures of network structure
  - Centrality, transitivity, homophily
- Introduction to Python and NetworkX
  - Network analysis using Python
- Python demo 1: Centrality analysis
- Common network analysis tasks
  - Community detection
  - Link prediction
- Python demo 2: Link prediction

Part 2: Dynamic Networks in Python (~60 min)

- Different representations of dynamic networks
  - Discrete-time snapshots (panel data)
  - Continuous-time relational event data
- Introduction to DyNetworkX
  - Data structures and algorithms for dynamic network analysis in Python
- Python demo 3: Dynamic centrality analysis
- Motifs in dynamic networks
- Python demo 4: Temporal motif analysis

Demo 2: Link prediction using NetworkX

Zachary’s Karate Club network

Part 2: Dynamic Networks in Python

Outline

Part 1: Analyzing Network Data in Python (~75 min)

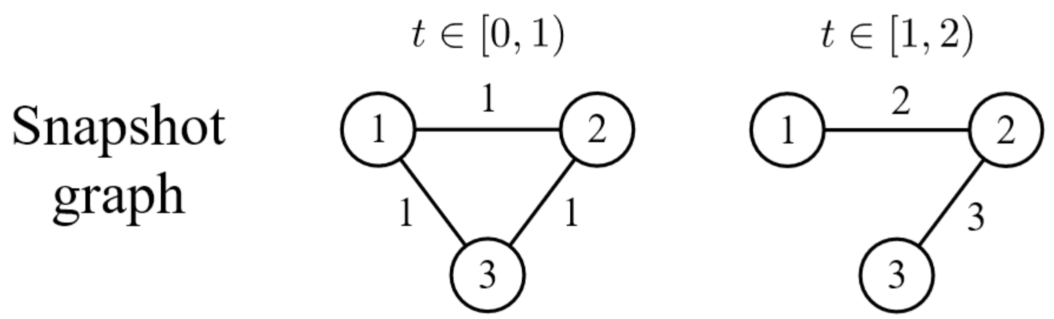
- Measures of network structure
  - Centrality, transitivity, homophily
- Introduction to Python and NetworkX
  - Network analysis using Python
- Python demo 1: Centrality analysis
- Common network analysis tasks
  - Community detection
  - Link prediction
- Python demo 2: Link prediction

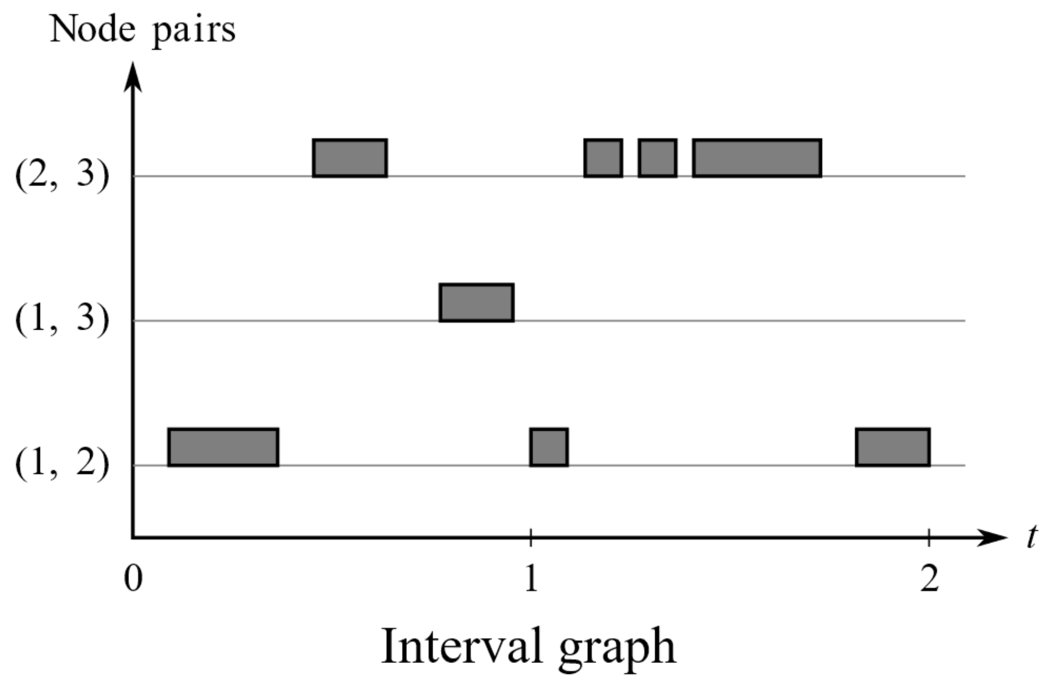
Part 2: Dynamic Networks in Python (~60 min)

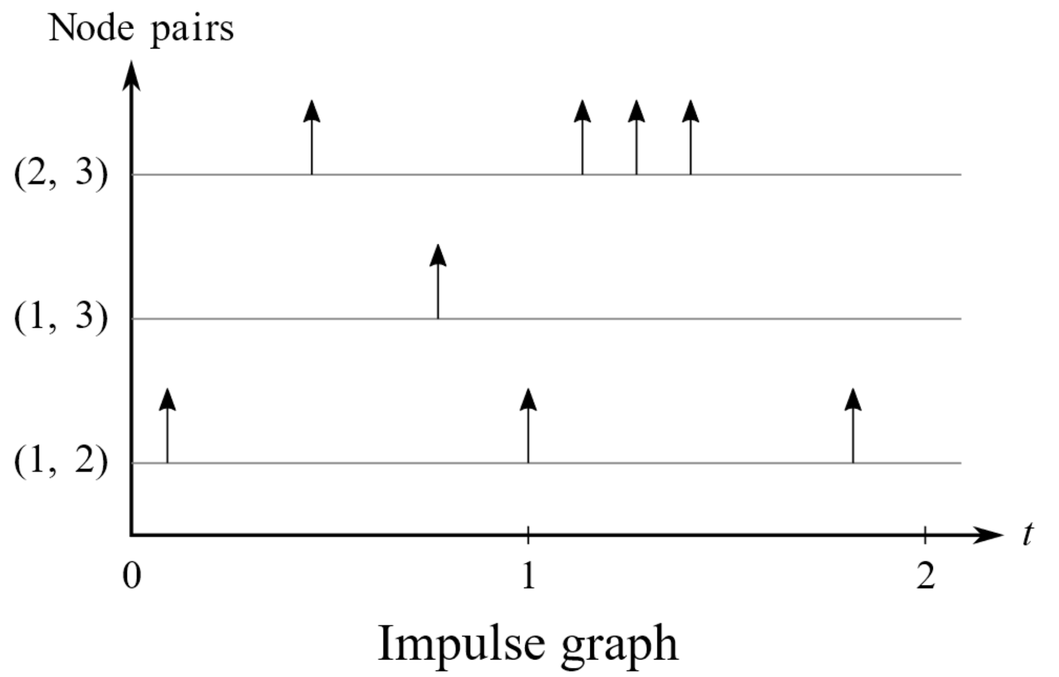
- Different representations of dynamic networks
  - Discrete-time snapshots (panel data)
  - Continuous-time relational event data
- Introduction to DyNetworkX
  - Data structures and algorithms for dynamic network analysis in Python
- Python demo 3: Dynamic centrality analysis
- Motifs in dynamic networks
- Python demo 4: Temporal motif analysis

Dynamic network representations

- Most complex systems represented by networks change continually over time
  - Need representations of temporal or dynamic networks
- Toy example: 3 different representations of 3-node network with repeated interactions over time

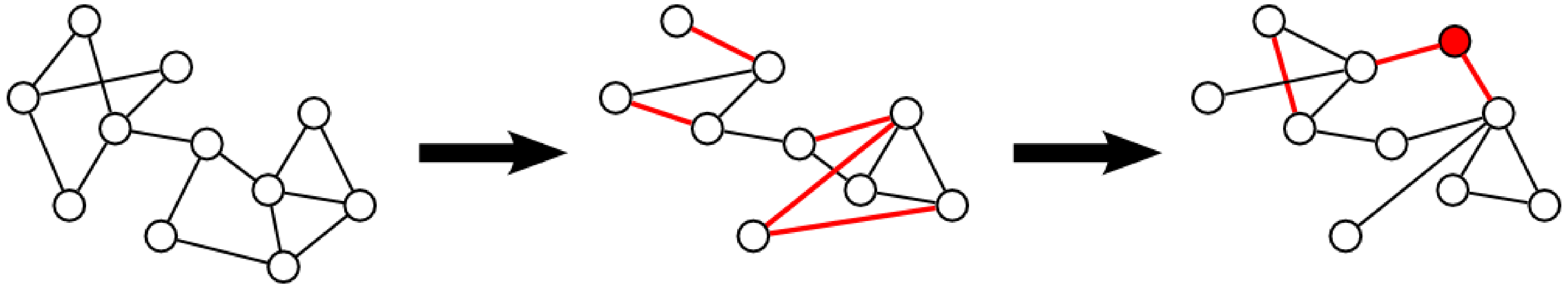






Discrete-time dynamic networks

- Network snapshots at discrete time steps
  - Nodes and edges can both appear and disappear over time



  - Adjacency matrix dimensions may change over time

$$A^{t-1} = \begin{bmatrix} 0 & 1 & \dots & 0 \\ 1 & 0 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 0 \end{bmatrix} \qquad A^{t} = \begin{bmatrix} 0 & 1 & \dots & 1 \\ 0 & 0 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \dots & 0 \end{bmatrix} \qquad A^{t+1} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 1 \\ 1 & 1 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 1 & \dots & 0 \end{bmatrix}$$

- Also known as network panel data

Continuous-time relational event networks

- Relational event data with fine-grained timestamps
  - Facebook wall posts (Viswanath et al., 2009)
- Represent impulse events (durationless) as triplets (i, j, t)
  - Examples: messages or emails between people
- Represent interval events (with duration) as $(i, j, t_{start}, t_{end})$ or $(i, j, t_{start}, \Delta t)$
  - Examples: phone calls or physical proximity between people
- Converting to discrete-time dynamic networks loses information

Sender	**Receiver**	**Timestamp**
1595	1021	1100626783
4581	5626	1100627183
3806	991	1100640075
521	533	1100714520
521	3368	1100716404
8734	527	1100724840
1017	1015	1100828851
17377	1021	1100832283
2926	726	1100838067

Dynamic network analysis techniques

Discrete time

- Run snapshot-by-snapshot analysis of network structure
  - Example: examine how centrality measure varies over time
- Time-aware approaches
  - Dynamic community detection
    - Produces community structure that evolves smoothly over time
  - Dynamic link prediction
    - Predicts both new links that are added and existing links that are removed

Continuous time

- Measures of temporal-topological structure
  - Time-respecting paths: only traverse edges that are valid at any given time
  - Shortest path becomes fastest path
  - Geodesic distance becomes latency
- Motifs in dynamic networks
  - Subgraphs where edges need to follow time constraints

Outline

Part 1: Analyzing Network Data in Python (~75 min)

- Measures of network structure
  - Centrality, transitivity, homophily
- Introduction to Python and NetworkX
  - Network analysis using Python
- Python demo 1: Centrality analysis
- Common network analysis tasks
  - Community detection
  - Link prediction
- Python demo 2: Link prediction

Part 2: Dynamic Networks in Python (~60 min)

- Different representations of dynamic networks
  - Discrete-time snapshots (panel data)
  - Continuous-time relational event data
- Introduction to DyNetworkX
  - Data structures and algorithms for dynamic network analysis in Python
- Python demo 3: Dynamic centrality analysis
- Motifs in dynamic networks
- Python demo 4: Temporal motif analysis

Introduction to DyNetworkX

While NetworkX can support dynamic networks through the use of edge attributes. It was not originally designed with this purpose in mind.

DyNetworkX is package, inspired by NetworkX, with the intent to restore the easy and inituitiveness of NetworkX on dynamic networks.

```
# query edges overlapping weekend
saturday = 432,000 # arbitrary weekend start
sunday = 604,800 # arbitrary weekend end
```

NetworkX:

```
# stored as {u: {v: {"begin":_, "end":_}}}
for u, v, d in G.edges(data=True):
    if d["begin"] < sunday and d["end"] > saturday):
        # do work...
```

DyNetworkX:

```
# stored as (u, v, begin, end)
for edge in G.edges(begin=saturday, end=sunday):
    # do work...
```

Introduction to DyNetworkX

NetworkX's data structure is like a phone book sorted by last name.

It's great for looking up people who might have the same last name as you, but what if you wanted to find people with a similar phone number to yours?

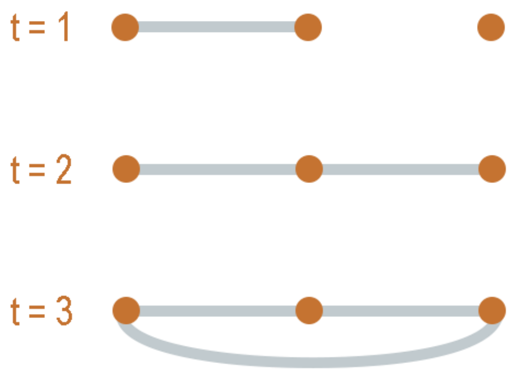
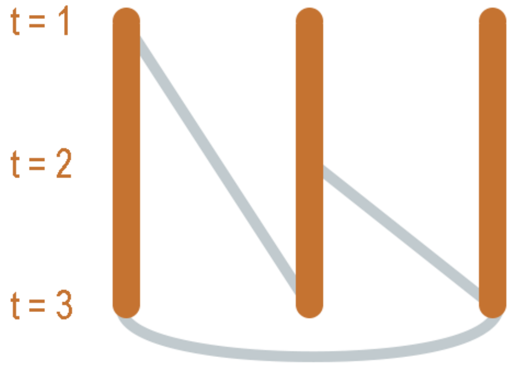
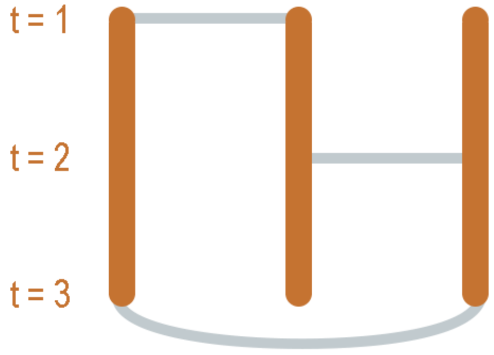
You need a phone book sorted by phone number too!

Smith, John	555-123-0589
Smith, Mark	555-987-3267
Smith, Robert	555-123-6092

555-123-0588	Doe, Jane
555-123-0589	Smith, John
555-123-0590	Jones, Ashley

Introduction to DyNetworkX

DyNetworkX features a single discrete representation class, SnapshotGraph, and two continuous representations, IntervalGraph and ImpulseGraph.

SnapshotGraph	IntervalGraph	ImpulseGraph
		
Stores dynamic network as a series of static networks. Static networks are stored as NetworkX Graphs and support most algorithms provided by NetworkX.	Treats the dynamic network as continuous. Best used for operations requiring rapid filtering of edges in overlapping time windows.	Special use case of IntervalGraph in which duration of edges are instantaneous. Common examples include: • Social Media Direct Messages • Email • Text Messages (SMS)

Introduction to DyNetworkX

DyNetworkX is a work in progress package written by the IdeasLab @ the University of Toledo.

You can learn more about DyNetworkX, and find our contact information, in our documentation.

Latest version can be found at our GitHub, or installed via pip.

GitHub:

github.com/IdeasLabUT/dynetworkx

Documentation:

dynetworkx.readthedocs.io

Installation:

```
pip install dynetworkx
```

Outline

Part 1: Analyzing Network Data in Python (~75 min)

- Measures of network structure
  - Centrality, transitivity, homophily
- Introduction to Python and NetworkX
  - Network analysis using Python
- Python demo 1: Centrality analysis
- Common network analysis tasks
  - Community detection
  - Link prediction
- Python demo 2: Link prediction

Part 2: Dynamic Networks in Python (~60 min)

- Different representations of dynamic networks
  - Discrete-time snapshots (panel data)
  - Continuous-time relational event data
- Introduction to DyNetworkX
  - Data structures and algorithms for dynamic network analysis in Python
- Python demo 3: Dynamic centrality analysis
- Motifs in dynamic networks
- Python demo 4: Temporal motif analysis

Demo 3: Dynamic betweenness centrality in DyNetworkX

Enron email network

Outline

Part 1: Analyzing Network Data in Python (~75 min)

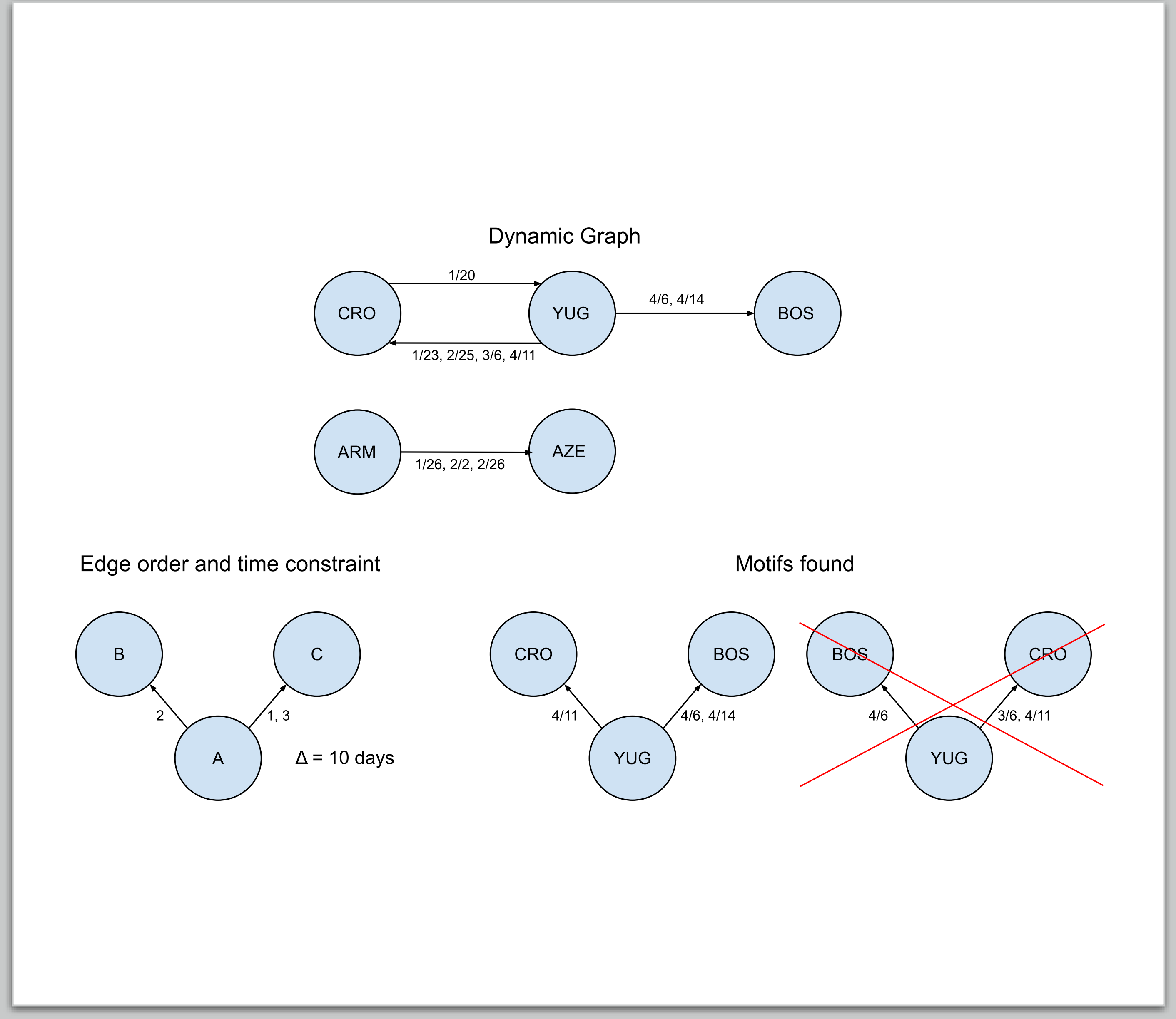
- Measures of network structure
  - Centrality, transitivity, homophily
- Introduction to Python and NetworkX
  - Network analysis using Python
- Python demo 1: Centrality analysis
- Common network analysis tasks
  - Community detection
  - Link prediction
- Python demo 2: Link prediction

Part 2: Dynamic Networks in Python (~60 min)

- Different representations of dynamic networks
  - Discrete-time snapshots (panel data)
  - Continuous-time relational event data
- Introduction to DyNetworkX
  - Data structures and algorithms for dynamic network analysis in Python
- Python demo 3: Dynamic centrality analysis
- Motifs in dynamic networks
- Python demo 4: Temporal motif analysis

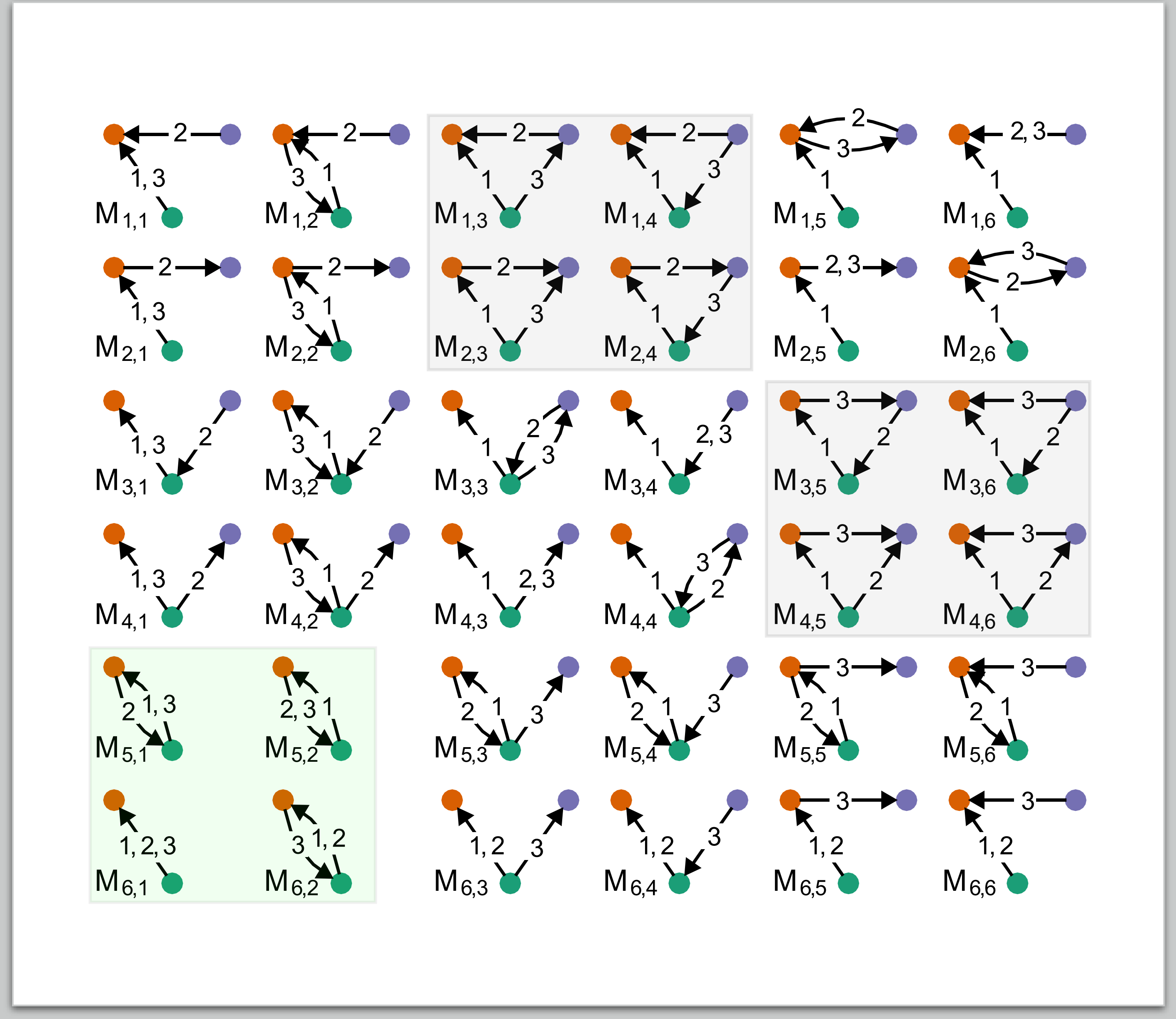
Temporal Motif

- Small patterns within a dynamic network
- Edges have to follow a predefined order
- Edges have to occur within a time constraint



All 2 or 3-node, 3-edge temporal motifs

- Green boxes: 2-node motifs
- Grey boxes: triangle motifs
- All other motifs: different variants of stars



Outline

Part 1: Analyzing Network Data in Python (~75 min)

- Measures of network structure
  - Centrality, transitivity, homophily
- Introduction to Python and NetworkX
  - Network analysis using Python
- Python demo 1: Centrality analysis
- Common network analysis tasks
  - Community detection
  - Link prediction
- Python demo 2: Link prediction

Part 2: Dynamic Networks in Python (~60 min)

- Different representations of dynamic networks
  - Discrete-time snapshots (panel data)
  - Continuous-time relational event data
- Introduction to DyNetworkX
  - Data structures and algorithms for dynamic network analysis in Python
- Python demo 3: Dynamic centrality analysis
- Motifs in dynamic networks
- Python demo 4: Temporal motif analysis

Demo 4: Temporal motifs in DyNetworkX

Militarized Interstate Disputes incident network

Summary

- Python is designed to be easy and intuitive
- NetworkX Python package provides lots of common network analysis tools
  - Mature and well maintained with a large community of developers
- Dynamic network analysis techniques vary depending on the network representation
- DyNetworkX Python package provides efficient data structures for working with dynamic networks
  - Integrates very well with NetworkX
  - Constantly under development—feel free to contribute!
- All materials available at tutorial website https://github.com/IdeasLabUT/SBP-BRiMS-2021-NetworkX-Tutorial