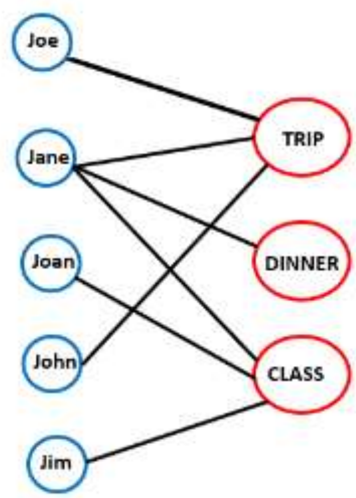
CS 5001 – Applied Social Network Analysis

Bipartite (a.k.a. Two-Mode) Graphs

- Definition: Nodes divided into 2 subsets (i.e., $V = V_1 \cup V_2$), edges from nodes in one set (V_1) to nodes in the other set (V_2)

Ex:



Bipartite graphs are great for modeling recommender systems!

- **Bipartite matrix:** assuming **n** entities (a.k.a., actors) and **m** things (a.k.a., events), use **n x m** matrix where A[i][j] = 1 if entity i has relationship with thing j, else 0

  So graph's set of nodes is $V = V_n \cup V_m$ where $|V_n| = n$ and $|V_m| = m$; V_n and V_m are disjoint

Ex: For the graph above

	TRIP	DINNER	CLASS	Row Total
Joe	1	0	0	1
Jane	1	1	1	3
Joan	0	0	1	1
John	1	0	0	1
Jim	0	0	1	1
Col Total	3	1	3	7

What's the significance of each row total?
What's the significance of each column total?

- **Event-by-Actor matrix:** you can get a slightly different view of things by looking at the **transpose of bipartite matrix A**; we'll call this 2-mode matrix $\mathbf{A^T}$

<u>Ex</u>: For the graph above

	Joe	Jane	Joan	John	Jim	Row Total
TRIP	1	1	0	1	0	3
DINNER	0	1	0	0	0	1
CLASS	0	1	1	0	1	3
Col Total	1	3	1	1	1	7

What's the significance of each <u>row</u> total?
What's the significance of each <u>column</u> total?

- **Actor-by-Actor matrix:** you can get a **social network of the actors** if you multiply the bipartite matrix A by its transpose A^T (i.e., $\mathbf{A * A^T}$); we'll call this one-mode matrix $\mathbf{X^A}$

<u>Ex</u>: For the graph above

1	0	0	*	1	1	0	1	0	=	1	1	0	1	0
1	1	1		0	1	0	0	0		1	3	1	1	1
0	0	1		0	1	1	0	1		0	1	1	0	1
1	0	0								1	1	0	1	0
0	0	1								0	1	1	0	1

Each row <u>and</u> column corresponds to an actor (i.e., Joe, Jane, Joan, John, Jim)

The **<u>diagonal value</u>** equals the row total in the bipartite matrix (i.e., actor's # events); the **avg of the values in the diagonal** gives avg # events attended by each actor (e.g., 7/5 = 1.4)

$\mathbf{X^A}$[i][k] = # events attended by <u>both</u> actor i and actor k

- **Event-by-Event matrix:** you can get a different matrix which shows **association between events** if you multiply the transpose A^T by the bipartite matrix A (i.e., $\mathbf{A^T * A}$); we'll call this one-mode matrix $\mathbf{X^E}$

<u>Ex</u>: For our bipartite graph

1	1	0	1	0	*	1	0	0	=	3	1	1
0	1	0	0	0		1	1	1		1	1	1
0	1	1	0	1		0	0	1		1	1	3
						1	0	0				
						0	0	1				

Each row and column corresponds to an event (i.e., TRIP, DINNER, CLASS)

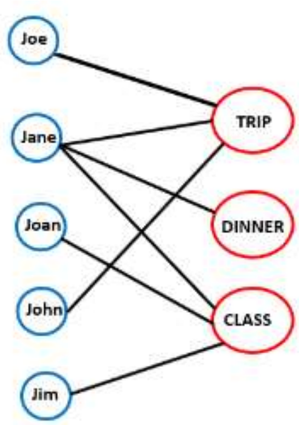
The **<u>diagonal value</u>** equals the # actors participating in that event; the **avg of the values in the diagonal** gives avg # actors participating in each event (e.g., 7/3 = 2.3)

X^E[i][k] = # actors participating in <u>both</u> event i and event k

- **Metrics:**
  - You can compute many of the same metrics that we've discussed for non-bipartite graphs (e.g., density, clustering coefficient, etc.)
  - However, they're somewhat skewed and/or misleading because of the **2 sets of nodes** and the "partitioning" of the edges; is the metric for a node in one set really comparable to the metric's value for a node in the other set???
  - **So there are bipartite versions of some of the metrics!**

Density: extent to which nodes are connected (max value = 1, min value = 0); in bipartite graph **must specify one of the partitions of nodes**, P; then compute as |E| / (|P| * (|V| - |P|)); if a <u>directed</u> graph, <u>divide by 2</u>

<u>Ex</u>:

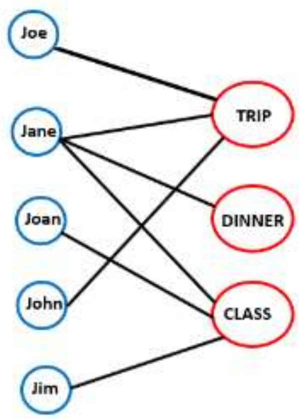


|E| = 7, |V| = 8
If P = {Joe, Jane, Joan, John, Jim},
then **density** = 7/(5*(8-5)) = 7/15
= 0.467

Moderately well connected

Degree centrality: usually just a count of # of connections a node has (i.e., degree) and "normalize" by dividing by |V|-1; **in bipartite graph, must "normalize" by dividing by # nodes in <u>opposite</u> partition**

<u>Ex</u>: For the graph below, partitions are {Joe, Jane, Joan, John, Jim} and {Trip, Dinner, Class}



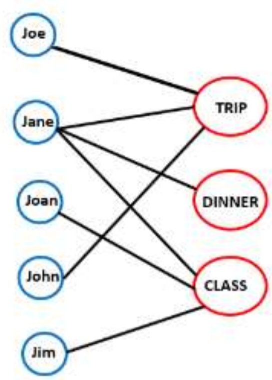
Normalized degree centrality for Jane is 3/3 = 1, and for John is 1/3 = 0.33

Normalized degree centrality for Trip is 3/5 = 0.6, and for Dinner is 1/5 = 0.2

Closeness centrality: considers how fast a node can reach other nodes; **in bipartite graph must specify one of the partitions of nodes** P; then compute closeness centrality for node as follows:

n = |P|
m = |V| - |P|
totSP = sum of lengths of all shortest paths that go through node
if node is in P
  closeness = (m + 2 * (n-1)) / totSP
else closeness = (n + 2 * (m-1)) / totSP
if normalize
  closeness = closeness * (# shortest paths that go through node – 1)/(|V|-1)

Ex:



	Closeness
John	0.579
Joe	0.579
Jim	0.579
Joan	0.579
Jane	1
Trip	0.692
Class	0.692
Dinner	0.529

Let P = {Joe, Jane, Joan, John, Jim},   n = 5, m = 3

Suppose node = Jane

Shortest paths that go through Jane (and their length): Jane -> Jane (0), Jane -> Trip (1), Jane -> Dinner (1), Jane -> Class (1), Jane -> Joe (2), Jane -> John (2), Jane -> Joan (2), Jane -> Jim (2)

totSP = 11 (i.e., sum of the lengths of the SP's, of which there are 8)

Jane's **closeness = (m + 2 * (n-1)) / totSP** = (3 + 2 * (5-1))/ 11 = 11/11 = 1

Normalized closeness = 1 * ((8-1)/(8-1)) = 1

Suppose node = Dinner

Shortest paths that go through Dinner (and their length): Dinner -> Dinner (0), Dinner -> Jane (1), Dinner -> Trip (2), Dinner -> Class (2), Dinner -> Joe (3), Dinner -> John (3), Dinner -> Joan (3), Dinner -> Jim (3)

totSP = 17 (i.e., sum of the lengths of the SP's, of which there are 8)

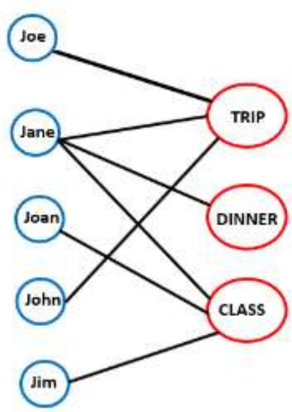
Dinner's **closeness = (n + 2 * (m-1)) / totSP** = (5 + 2 * (3-1))/ 17 = 9/17 = 0.529

Normalized closeness = 0.53 * ((8-1)/(8-1)) = 0.529

Betweenness centrality: considers # of shortest paths that go through a node (i.e., is this node a "bridge"); **in bipartite graph must specify one of the partitions of nodes** P; then compute betweenness centrality for node as follows:

n = |P|
m = |V| - |P|
if node is in P
{
 s = floor((n – 1) / m)
 t = (n – 1) % m
 adjustment = ($(m^2*(s+1)^2)$ + (m*(s+1)*(2*t-s-1)) - (t*((2*s)-t+3))) / 2
}
else
{
 p = floor((m – 1) / n)
 r = (m – 1) % n
 adjustment = ($(n^2*(p+1)^2)$ + (n*(p+1)*(2*r-p-1)) - (r*((2*p)-r+3))) / 2
}
betweenness = betweenness centrality as computed conventionally / adjustment

Ex:



	Betweenness
John	0
Joe	0
Jim	0
Joan	0
Jane	0.938
Trip	0.579
Class	0.579
Dinner	0

Let P = {Joe, Jane, Joan, John, Jim}, n = 5, m = 3

Suppose node = Trip

p = floor((3-1)/5) = floor(2/5) = 0 r = (m-1) % n = 2 % 5 = 2

adjustment = ($(5^2*(0+1)^2)$ + (5*(0+1)*(2*2-0-1)) - (2*((2*0)-2+3))) / 2 = 19 betweenness for Trip would normally be calculated as follows **(note: don't count SP's <u>to</u> Trip, and don't count same pair twice!)**:

SP's from Joe that involve Trip (to Jane, Joan, John, Jim, Dinner, Class): 6

SP's from Jane that involve Trip (to John): 1

SP's from Joan that involve Trip (to John): 1

SP's from John that involve Trip (to Jim, Dinner, Class): 3

SP's from Jim that involve Trip: 0

SP's from Dinner that involve Trip: 0

SP's from Class that involve Trip: 0

Total SP's that involve Trip = 11

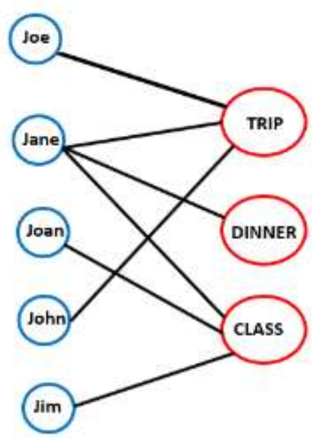
Trip's bipartite betweenness centrality = 11/adjustment = 11/19 = 0.579

Clustering coefficient: the higher the value for a node (max = 1), the closer it and its neighbors resemble a clique; **in bipartite graph** it's kind of like that (but you have mixture of node types…), compute clustering coefficient for node u as follows:

N(u) = set of neighbors of node u, excluding node u
c(u, v) = |N(u) ∩ N(v)| / |N(u) U N(v)| where u, v are nodes

First find X = N(N(u)) (i.e., the neighbors of the neighbors of the node u)
For every node v in X, find c(u, v) and add it to a sum
Divide the sum by |X|

<u>Ex</u>:



	Clustering Coeff
John	0.667
Joe	0.667
Jim	0.667
Joan	0.667
Jane	0.333
Trip	0.267
Class	0.267
Dinner	0.333

Let node u be John

Then N(u) = {Trip}

N(N(u)) = {Joe, Jane} don't include John

Need to compute c(John, Joe) and c(John, Jane)

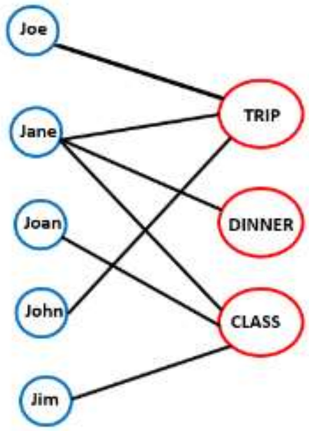
c(John, Joe):
N(John) = {Trip}, N(Joe) = {Trip}
N(John) ∩ N(Joe) = {Trip}, N(John) U N(Joe) = {Trip}
c(John, Joe) = |N(John) ∩ N(Joe)| / |N(John) U N(Joe)| = 1/1 = 1

c(John, Jane):
N(John) = {Trip}, N(Jane) = {Trip, Dinner, Class}
N(John) ∩ N(Jane) = {Trip}, N(John) U N(Jane) = {Trip, Dinner, Class}
c(John, Jane) = |N(John) ∩ N(Jane)| / |N(John) U N(Jane)| = 1/3 = 0.333

Sum = 1.333
Divide by |N(N(u))| = |{Joe, Jane}| = 2
So John's **clustering coefficient** is 1.333/2 = 0.667
<u>Note</u>: {John, TRIP} not "truly" a clique (coefficient isn't 1) because different node types

Python for Bipartite Graphs

Ex:



Some operations on a bipartite graph

import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

First way to create a new undirected graph
B = nx.Graph()

 # Add nodes with an attribute "bipartite"
B.add_nodes_from(['joe','jane','joan','john','jim'], **bipartite=0**)
B.add_nodes_from(['trip','dinner','class'], **bipartite=1**)

Add edges
B.add_edges_from([('joe','trip'),('jane','trip'),('jane','dinner'),('jane','class'),('joan','class'),
('john','trip'),('jim','class')])

Display the graph
First, separate nodes into 2 groups
l, r = nx.bipartite.sets(B)

# Then assign position for node in each group
pos = {}
pos.update((node, (1, index)) for index, node in enumerate(l))
pos.update((node, (2, index)) for index, node in enumerate(r))

nx.draw(B, pos=pos, with_labels=True)
plt.show()

Second way to create a new undirected graph; don't need attribute
B2 = nx.Graph()

# Add nodes
B2.add_nodes_from(['joe','jane','joan','john','jim'])
B2.add_nodes_from(['trip','dinner','class'])

Add edges
B2.add_edges_from([('joe','trip'),('jane','trip'),('jane','dinner'),('jane','class'),('joan','class'
),('john','trip'),('jim','class')])

from networkx.algorithms import bipartite

 # Tests whether bipartite
 # Note: This would NOT work if nodes were both int and <u>same</u> int used in both groups!
bipartite.**is_bipartite**(B2)

 # Make 2 groups of nodes
 # Note: The following does NOT work if graph is disconnected! (Ambiguous solution)
X, Y = **bipartite.sets**(B2)
list(X) # X = [john, jane, joe, joan, jim]
list(Y) # Y = [class, dinner, trip]

 # Degrees of nodes; whichever you list as arg is one you get 1st
degX, degY = **bipartite.degrees**(B2, **X**)
print(degX, degY)

c = **bipartite.color**(B2) # assigns binary 'color' number for each node
print(c["john"]) # node john's color

 # Metrics (in addition to, sometimes different from, nx. ones)
 # most require specifying group of nodes (affects result order)

nx.density(B2)
bipartite.density(B2, X)

nx.clustering(B2)
bipartite.clustering(B2)

nx.degree_centrality(B2)
bipartite.degree_centrality(B2, X)

print(nx.closeness_centrality(B2))
bipartite.closeness_centrality(B2, X)

nx.betweenness_centrality(B2)
bipartite.betweenness_centrality(B2, X)

 # Make a bipartite matrix
# Slightly different order than what was shown in lecture
# because here we're sorting the row and column names
# alphabetically
row_order = sorted(list(X)) # specify rows in matrix
col_order = sorted(list(Y))   # specify rows (optional)
numpyMatrix = **bipartite.biadjacency_matrix**(B2, row_order, column_order=col_order)
M = numpyMatrix.A   # .A gets us an ndarray object
print(M)
print(M[0,0])   # gets us jane, class
print(M[4,2])   # gets us john, trip

 # Make the event-by-actor matrix
# Rows are class, dinner, trip
# Columns are Jane, Jim, Joan, Joe, John
AT = np.**transpose**(M)
print(AT)
print(AT[0,0])   # gets us class, jane
print(AT[2,4])   # gets us trip, john

```
# Make the actor-by-actor matrix
# Rows and columns correspond to Jane, Jim, Joan, Joe, John
XA = M.dot(AT)
print(XA)
print(XA[0][1])         # gets us jane, jim
print(XA[3][1])         # gets us joe, jim

# Make the event-by-event matrix
# Rows and columns correspond to class, dinner, trip
XE = AT.dot(M)
print(XE)
print(XE[0][1])         # gets us class, dinner
```