L23 -- Communities
[Jeff Phillips - Utah - Data Mining]

Social Network == Large (directed) graph

G = (V,E)

------------

Draw Example

------------

Mid 2000s very exciting time.
- People studying networks for years
- Much anecdotal evidence on small graphs 10s to 100s
+ Finally in 2000s, large scale networks --> could see effects
+ Could collect data (explosion of work)

--------------------------------

Example question:
Why do people join groups?

Group C

Two people not in C: X, Y
- X has three friends in C, all connected
- Y has three friends in C, none connected

Who more likely to join?

for
X: safety/trust in friends who know each other
Y: independent support

Answer: X
  --> tightly connected subsets in graphs

---------------------

so: HOW DO WE FIND COMMUNITIES

Option 1:
 Local properties:
  + how many incoming/out-going edges
  + count triangles
    (A,B) and (A,C) ->
      + more likely (B,C)

  + B C trust each other
  + A incentive to bring B,C together
  + if A has few triangles, more depressed (empirical study)
- Easily spoofed

Option 2:
Spectral Clustering
(already covered, L11)

Option 3:
Betweenness

betw((a,b)) = # shortest paths that use edge (a,b)

How to interpret betw(a,b)?
large score is bad (between communities, not within community)

How to calculate (a,b)?

```
  <all-pairs shortest path>
  For each v in V
   1: DFS on entire graph -> build DAG
   2: Walk from each leaf back-up, adding counter to each edge
      (need to split walk up if multiple paths)
```

Explain on Example

What about ties?

How efficient?
O(|V| * |E|)
Very slow. Various sampling attempts, none satisfactory

Use to find communities?
- remove high-betweeness edges...

Also:
High betweenness edges are important for keeping network connected!

Option 4:
Modularity:
 Q = (# edges in group) - (expected number in group)
actual A_{i,j} = {1 if edge, 0 otherwise}
E_{i,j} = d_i * d_j / 2|E|
  d_i = degree if node i
  |E| = number of nodes (allows self edges)

Q(C) = (1/4m) [sum_{ij in C} (A_{i,j} - E_{i,j|)]
     in [-1,1]
      positive if number edges exceed expectation
      Q in [0.3,0.7] significant

  (better statistical ways to look at this SSS)
  (always some high-modularity cluster, but is it significant?)

[bias towards large communities (with > sqrt{|E|} edges)]

How to optimizes modularity directly?
  Use Spectral Clustering!
  + Finding leading eigenvector.
  + Find best split.
    If split increases modularity, recurse
    Else: stop

 (if too slow, use PageRank repetition to estimate eigenvector!)

Alternative: Build bottom-up (Hierarchical clustering)
      + Greedy Nibble: Add one best node at a time, repeat
      + Greedy Chomp: Add (or subtract) all nodes which individually improve modularity
      --> local minimum

To find smaller communities:
    --> Look for complete graphics (cliques)
    --->   complete bipartite graphs K_{s,t}