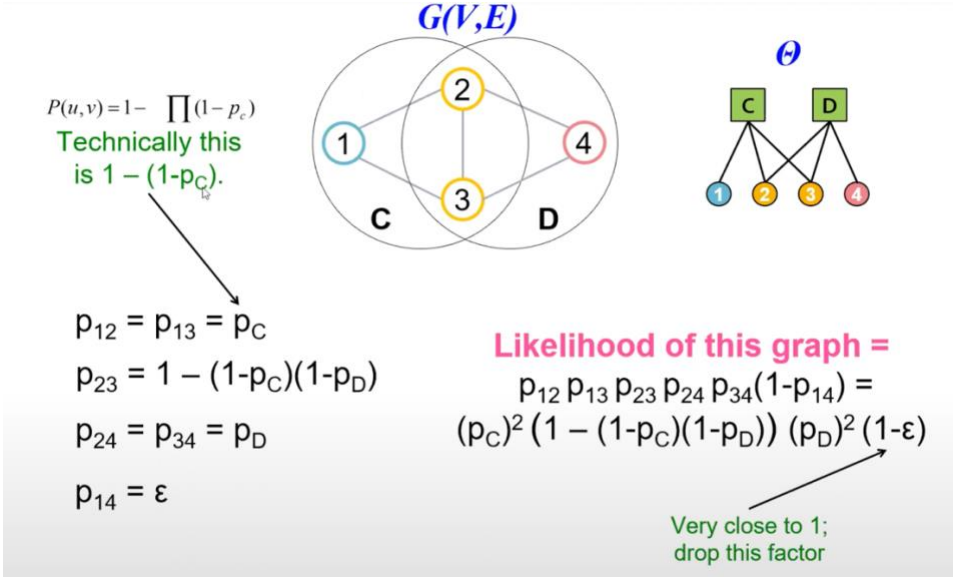
Over-lapping community Detection

- **Recap:**
  - Undirected graph:
    - Max edges = (n-1)*n/2
    - Max number of edges it can have
  - Directed graph:
    - Max edges = n(n-1)
  - With self loop + n
- **Plan of attack:**
  - Given a model, generate the network
  - Given a graph, generate the model
- **Maximum likelihood estimate (MLE):**
  - Probability of the graph: P(Gx|M) = (P^edges)*(1-P)^(max edges-real edges)
  - Find max by taking derivative and set to 0 to find best parameter



- **Community-affiliation graph: Affliction graph model**
  - **Goal**: Given set of information, what is the probability of two nodes have edge/link. With probability generated for each pair, we can then generate a graph for what the model looks like.
  - For example, we know A and B from same community, we want to find probability of A and B have edge.
  - **Generative model: B(V,C,M,{Pc})**
    - Nodes: V
    - Community: C
    - Membership: M

        - Each community is represent by a: Pc
    - **Affiliation graph:**
        - Given: A number of communities and number of individuals
        - Two sets of parameters:
        - General case: If u and v are members of a set M of communities, the probability of an edge between u and v is:

$$p_{uv} = 1 - \prod_{C \text{ in } M} (1 - p_C)$$

    -

- Given $C = \{w, x, y\}$ and $D = \{w, y, z\}$
  - For w and x, $M_{wx} = \{C\}$
  - $P_{wx} = 1 - (1 - P_c) = P_c$
  - And
  - $P_{xy} = P_C$ and $P_{yz} = P_{wz} = P_D$
  - $P_{wy} = 1 - (1 - P_C)(1 - P_D)$
  - P_{xz} = a tiny value (very low probability)

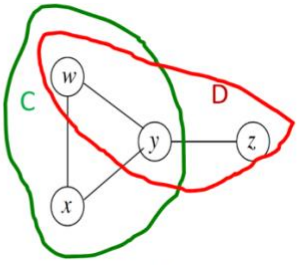


Figure 10.20: A social graph

- The likelihood of this graph given the membership assumption
- $P(G|AGM) = P_{wx} P_{wy} P_{xy} P_{yz} (1-P_{wz})(1-P_{xz})$
  $= (P_C)^2 P_D (P_C + P_D - P_C P_D)(1-P_D)(1-\varepsilon)$

- **BigClam (Big community learning algorithm)**
    - Build upon affiliation graph model
    - Given nodes and the strength they belong to each community. Check if pairs of members in overlapping community have edge. If so, we can construct relation graph between nodes.
    - Relaxation: memberships have strength (avoid discrete membership changes
    - **Maximizing the likelihood** of observed edges.
- **Using log-likelihood transformation** to simplify calculations.
    - Fua: strength of node U to community A. (0 to positive)
    - Probability of connection is proportional to the product of strengths.
        - Pa(U,V) = 1-exp-Summation(FuA*FvA) , where “*” is dot product
        - Using using exp to have smooth output
        - Negative summation ensures high edge probability

- Community A links nodes u, v independently:

$$P_A(u,v) = 1 - \exp(-F_{uA} \cdot F_{vA})$$

- Then prob. at least one common C links them:

$$P(u,v) = 1 - \prod_C (1 - P_C(u,v)) = 1 - \exp(-\sum_C F_{uC} \cdot F_{vC}) = 1 - \exp(-F_u \cdot F_v^T)$$

- **Example F matrix:**

F_u:	0	1.2	0	0.2
F_v:	0.5	0	0	0.8
F_w:	0	1.8	1	0

Node community membership strengths

Then: $F_u \cdot F_v^T = 0.16$

And: $P(u,v) = 1 - exp(-0.16) = 0.14$

But: $P(u,w) = 0.88$

$P(v,w) = \varepsilon$

We assume that if P(edge)=0, then P(edge)= ε

- Bigclam optimization
  - Given a network
  - Need maximize

$$L = \prod_{(u,v)\in E} \left(1 - e^{-\sum_c F_{u,c}F_{v,c}}\right) \prod_{(u,v)\notin E} e^{-\sum_c F_{u,c}F_{v,c}}$$

  - Change product to summation to reduce computation using log

$$\log L = \sum_{(u,v)\in E} \log P(E_{uv} = 1) + \sum_{(u,v)\notin E} \log P(E_{uv} = 0)$$

  -

$$l(F_u) = \sum_{v\in\mathcal{N}(u)} \log(1 - \exp(-F_u F_v^T)) - \sum_{v\notin\mathcal{N}(u)} F_u F_v^T$$

  -
  - Then we compute the gradient of above
- Summary for BigClam
  - Given G(V, E), compute B(C,M,{pc}) such that P(G|B) is maximum
    - But it is too hard to find C, M, and {pc} this way
  - V1.0. Change to a model easier to find
    - Replace C and M by F // binary -> strength
    - Define P(u,v) = 1-exp(-FuAFvA)
    - Compute F such that P(G|F) is maximum
  - V2.0. Speed up the computation of
    - By computing cache summation of Fv