Community Detection
Lecture 3

E0: 259

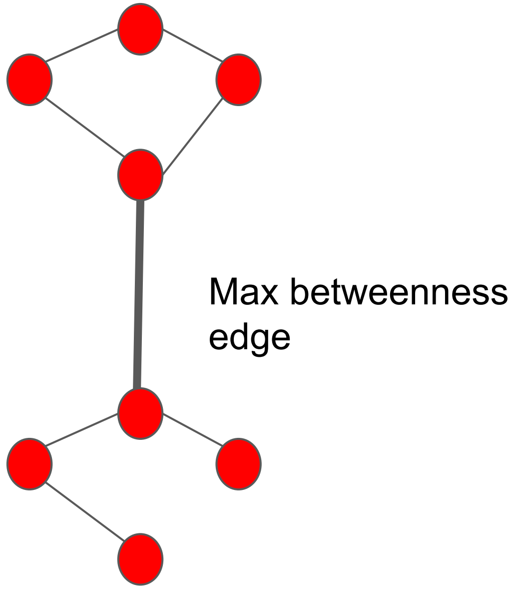
Recall: Betweenness Centrality

σ_{st} = *number of shortest paths between node s and node t in a graph*

$\sigma_{st}(e)$ = *number of shortest paths between node s and node t that pass through edge* $e \in E$

Edge Betweenness Measure

$$C_B(e) = \sum_{s \neq t} \frac{\sigma_{st}(e)}{\sigma_{st}}$$



Intuitively removing max betweenness edges recursively should give good partitions

Computing Betweenness

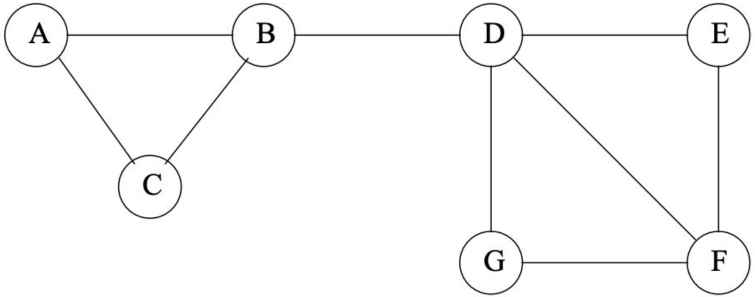
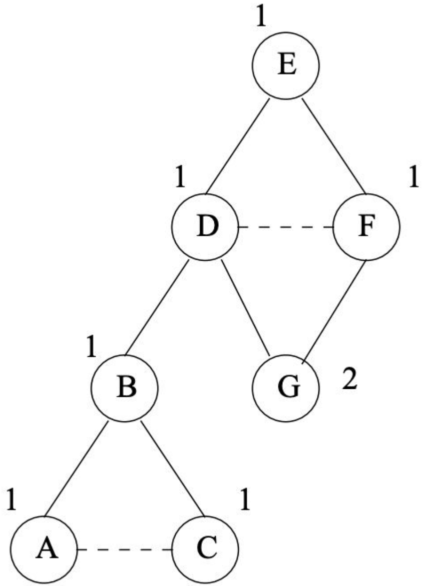
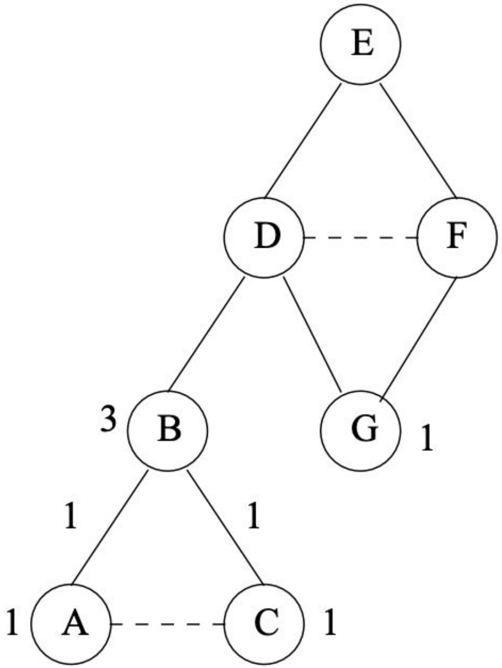


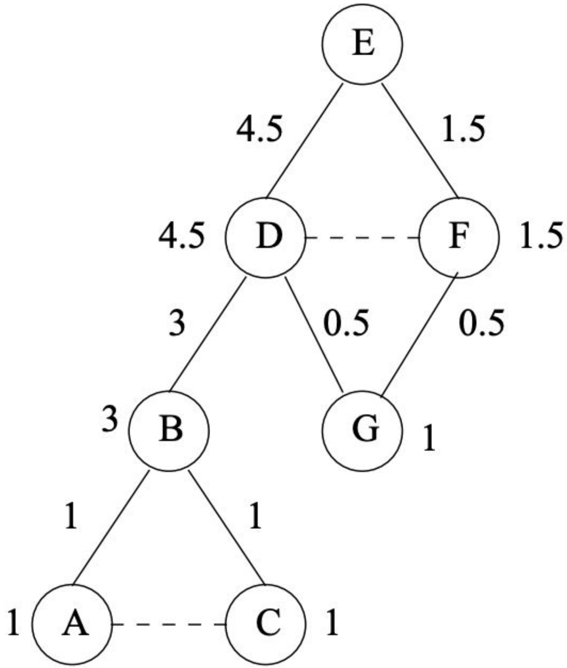
Figure 10.3: Repeat of Fig. 10.1



From: Chapter 10: http://www.mmds.org/

Computing Betweenness Contd.

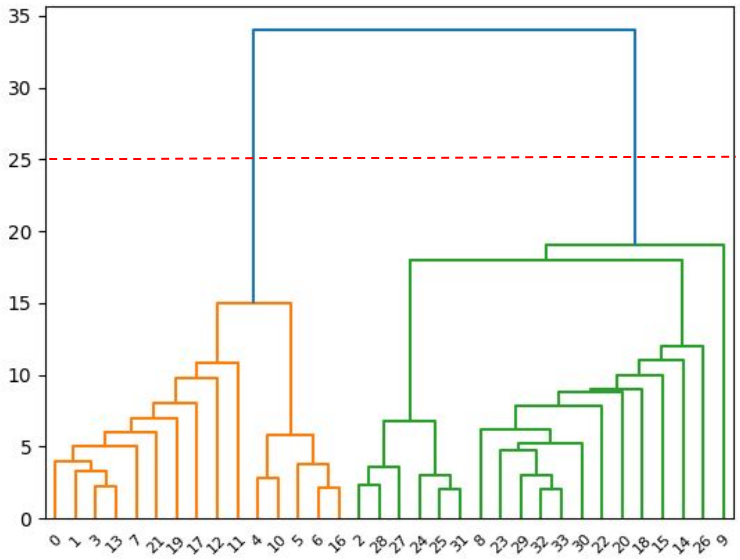




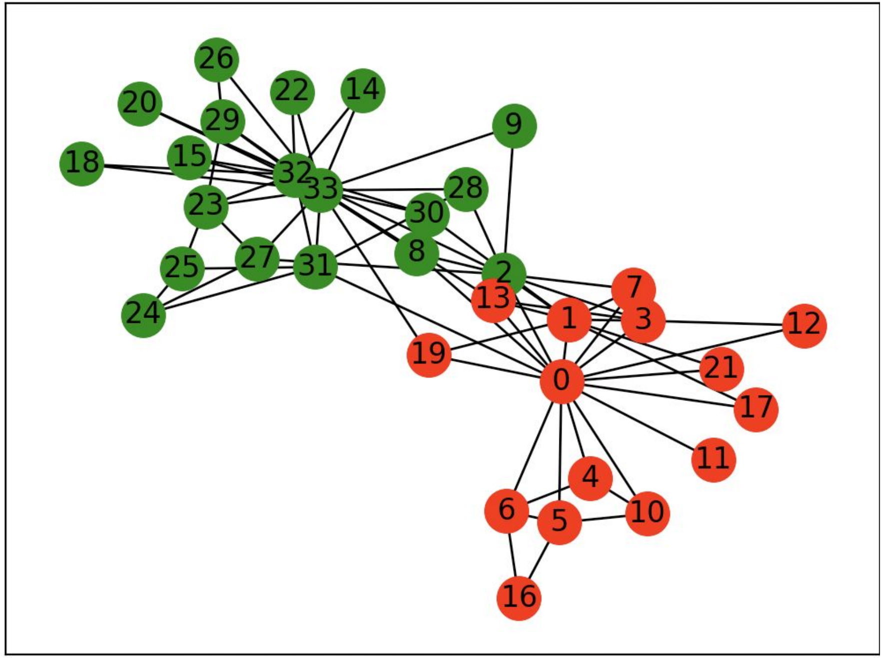
From: Chapter 10: http://www.mmds.org/

- Each Node at leaf of BFS gets credit of 1. *A, C and G get credit of 1*
- Each edge leading to level above accumulates credit from child node in proportion to number of shortest paths to it. E.g AB gets 1, DG gets 0.5

Zachary Karate Club - Girvan Newman



Dendogram



2 Communities

Community Detection Algorithms - Girvan Newman (2004)

Algorithm 1 Girvan Newman Algorithm

Input $G = (V, E)$
Output: Dendogram
while E not ϕ
    compute $C_B(e), \forall e \in E$
    $E = E \backslash e_i$, where $e_i = \arg\max_{e \in E} C_B(e)$

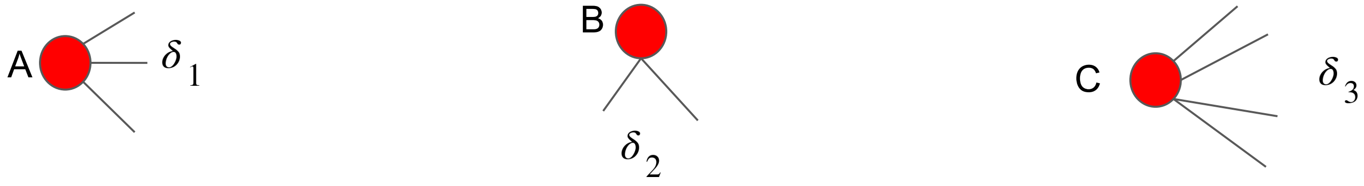
$C_B(e) = \sum_{s \neq t} \frac{\sigma_{st}(e)}{\sigma_{st}}$

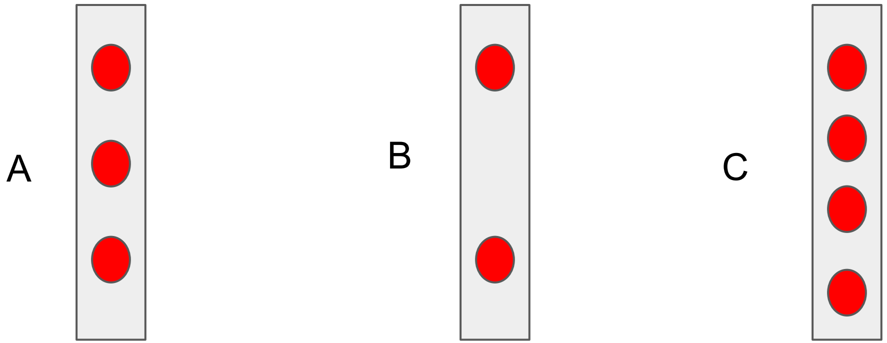
$\sigma_{st}(e)$ is all shortest paths between s and t passing through e

σ_{st} is all shortest paths between s and t

Recall: Configuration Model

- Ideally would like to preserve additional structural properties of the actual subgraph.
  - E.g. the node degrees are identical to the original graph.





- Make δ_i copies of each node i
- Pick 2 nodes from set of eligible nodes
- Put an edge between them
- Remove these 2 nodes from set of eligible nodes
- For large graphs guarantees that expected number of edges is same as original graph

Recall: Configuration Model

Consider a graph $G = (V, E)$, *let* $n = ||V||$ *and* $m = ||E||$,

let δ_i *be degree of node* i

then in the configuration model, expected number of edges between node i *and node* j *is* $\delta_i \frac{\delta_j}{2m}$

Expected number of edges with configuration model:

$$\sum_{i \in V} \sum_{j \in V} \delta_i \frac{\delta_j}{2m} = \frac{1}{2m} \sum_{i \in V} \delta_i \sum_{j \in V} \delta_j = 2m$$

With the configuration model, number of nodes, degree of each node and total number of edges are all preserved

Recall: Modularity with Configuration Model

$$Let\ A\ be\ the\ adjacency\ matrix\ of\ the\ graph.\ A_{ij} = \begin{cases} 1,\ if\ \ i\ and\ j\ have\ an\ edge \\ 0,\ otherwise \end{cases}$$

$$Modularity\ Q = \sum_{p \in P} (\#\ of\ edges\ in\ p - expected\ \#\ of\ edges\ in\ p)$$

$$Modularity\ Q = \frac{1}{2m} \sum_{p \in P} \sum_{i \in p} \sum_{j \in p} \left(A_{ij} - \frac{\delta_i \delta_j}{2m} \right)$$

Louvain Algorithm

- Input is a set of communities, output is a dendogram
- Algorithm that runs in phases.
- Phase 1: greedily generate communities from give graph.
- Phase 2: Cluster communities in phase 1 into super nodes and form new graph. Run Louvain algorithm on new graph.
- Repeat Phase 1 and Phase 2 until denogram left.

Louvain Algorithm - Phase 1

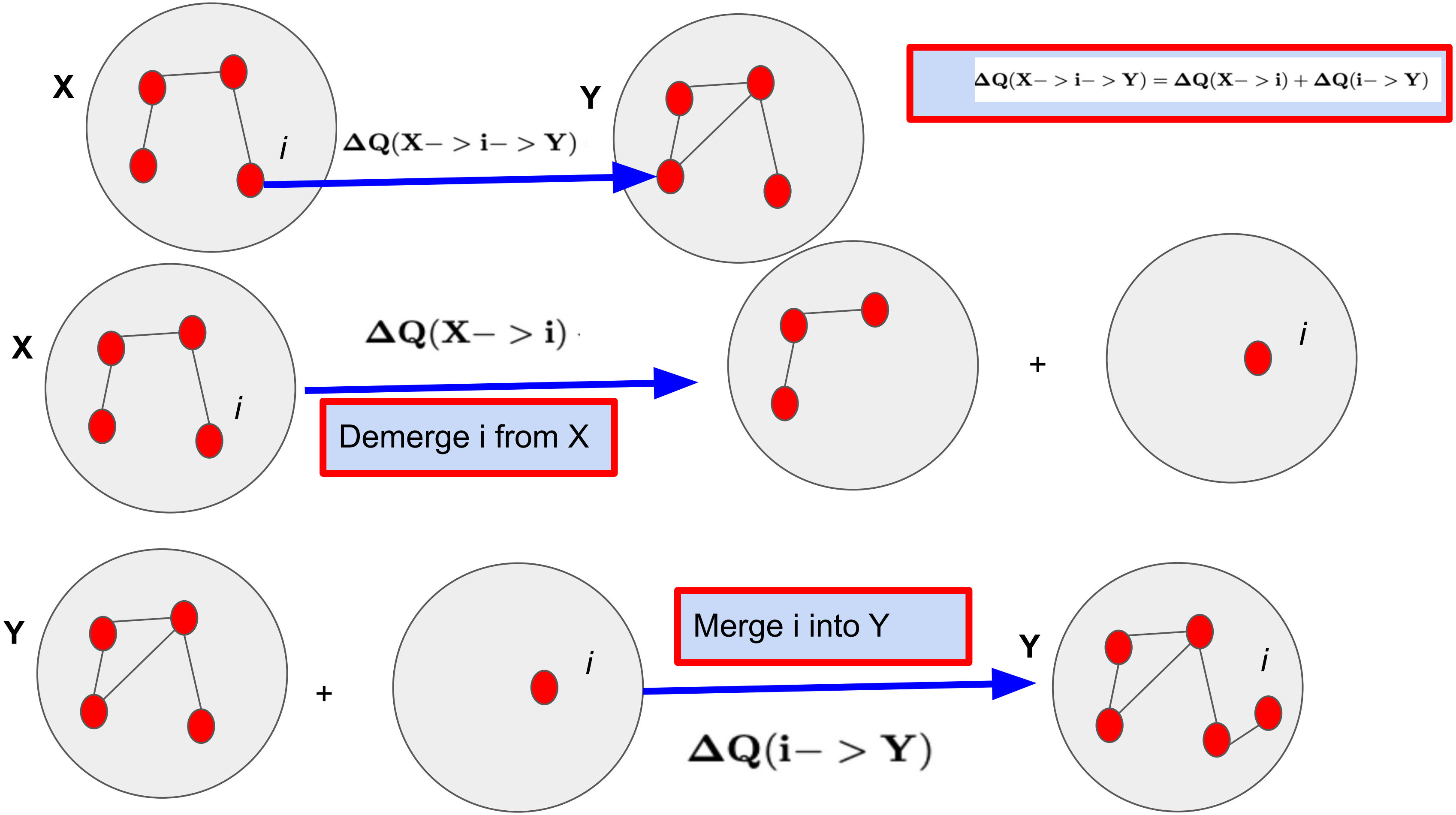
Algorithm 1 Louvain Algorithm Phase 1

Input: $G = (V, E)$
Output: Graph Partitions

Initialize: Assign every node to its own community
Step 1: for $i \in V$
    for $j \in C$
        compute ΔQ_{ij} if i moves to j
Step 2: $i^*, j^* = arg\,\max_{ij} \Delta Q_{ij}$
Step 3: Move node i^* to community j^*

Repeat Step 2 and 3 until no further improvement in modularity is possible.

Computing Change in Modularity



- For any community X we have:

$$\sigma_{in} = \sum_{i,j \ in X} A_{ij}$$

Number of edges within community.
Gets double counted

$$\sigma_X = \sum_{i \in X} k_i$$

Total degree of nodes within community

- Therefore $Q(X) = \frac{1}{2m} \sum_{i,j \in X} (A_{i,j} - \frac{k_i k_j}{2m})$

$$Q(X) = \frac{\sum_{i,j \in X} A_{i,j}}{2m} - \frac{\sum_i k_i}{2m} \frac{\sum_i k_j}{2m}$$

$$Q(X) = \frac{\sigma_{in}}{2m} - (\frac{\sigma_X}{2m})^2$$

Computing Change in Modularity (contd)

- Computing $\mathbf{\Delta Q(i > Y)}$
- Before merging, total modularity of i plus Y is:

$$Q_{before} = Q_{\{i\}} + Q(Y)$$

$$Q(i) = A_{ii} - \frac{k_i^2}{2m}$$

$$Q(i) = 0 - \frac{k_i^2}{2m}$$

$$Q(Y) = \frac{\sigma_{in}}{2m} - (\frac{\sigma_Y}{2m})^2$$

$$Q_{before} = \frac{\sigma_{in}}{2m} - (\frac{\sigma_Y}{2m})^2 - \frac{k_i^2}{2m}$$

- Now compute modularity Q_{after} after merging i to Y

- Define $k_{i,Y} = \sum_{j \in Y} A_{i,j} + \sum_{j \in Y} A_{j,i}$

Number of edges going from *i* to nodes in *Y*

- $Q_{after} = Q_{i \cup Y}$
- $\sigma_{in}(after) = \sigma_{in}(Y) + k_{i,Y}$

Number of edges in the new community {i U Y}

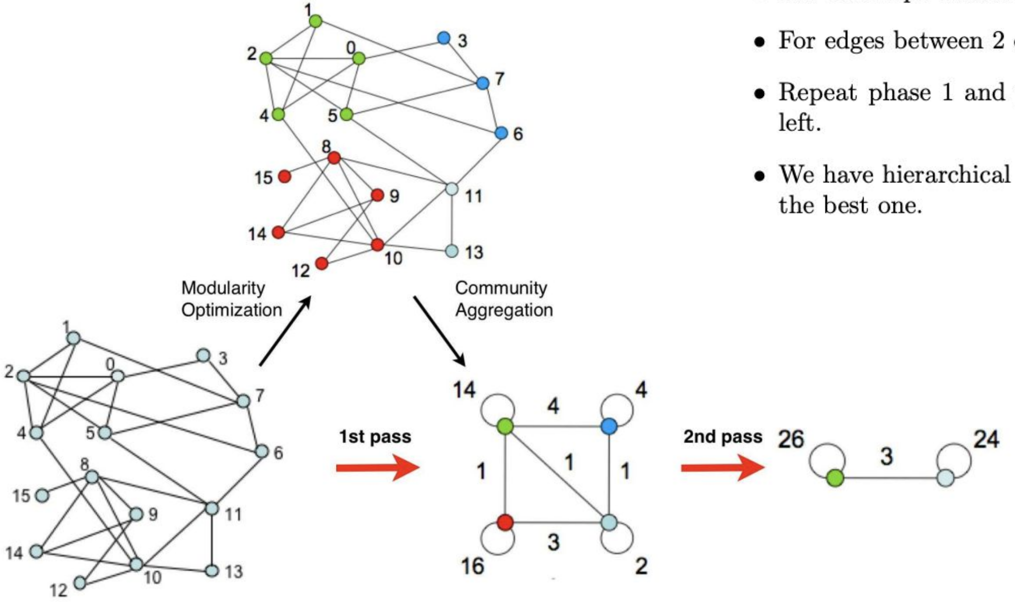
- $\sigma_{i \cup Y} = \sigma_Y + k_i$
- $Q_{after} = \frac{\sigma_{in}(Y) + k_{i,Y}}{2m} - (\frac{\sigma_Y + k_i}{2m})^2$

Total degree of nodes in community {i U Y}

$$\boxed{\Delta Q(i- > Y) = Q_{after} - Q_{before}}$$

Change in modularity for demerging also computed similarly.

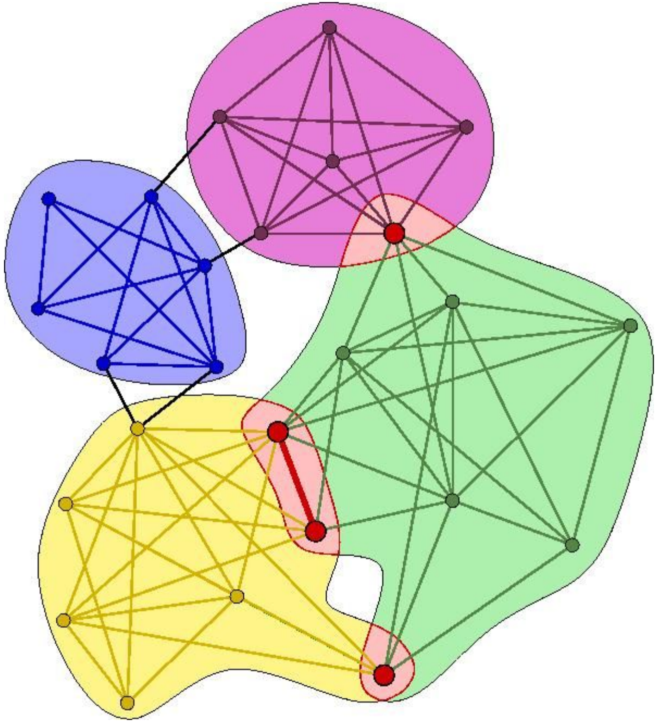
Louvain Algorithm



- For self loops within community super node C, weight is: $\sum_{i,j \in C} A_{i,j}$
- For edges between 2 community super nodes, weight is $\sum_{i \in X, j \in Y} A_{i,j}$
- Repeat phase 1 and phase 2 on super nodes until only two super nodes left.
- We have hierarchical partitions of nodes into different communities. Pick the best one.

V. Blondel et. al. 2008

Overlapping Communities

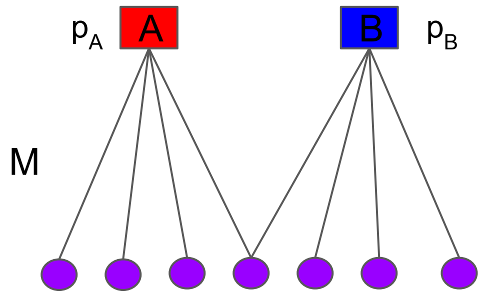


How do we find such overlapping communities?
Especially in large scale graphs?

Agglomerative Generative Models

- Step 1: Define a probabilistic model that can generate overlapping communities
- Assumption: All community can be modeled by this probabilistic model
- Step 2: Look at the actual data in the graph, essentially the adjacency matrix and ask the question: “what parameters should the probabilistic model have to generate this adjacency matrix?”
- If we can come up with an efficient algorithm to answer this question, we can actually determine overlapping communities

A Candidate Probabilistic Model for Community Generation



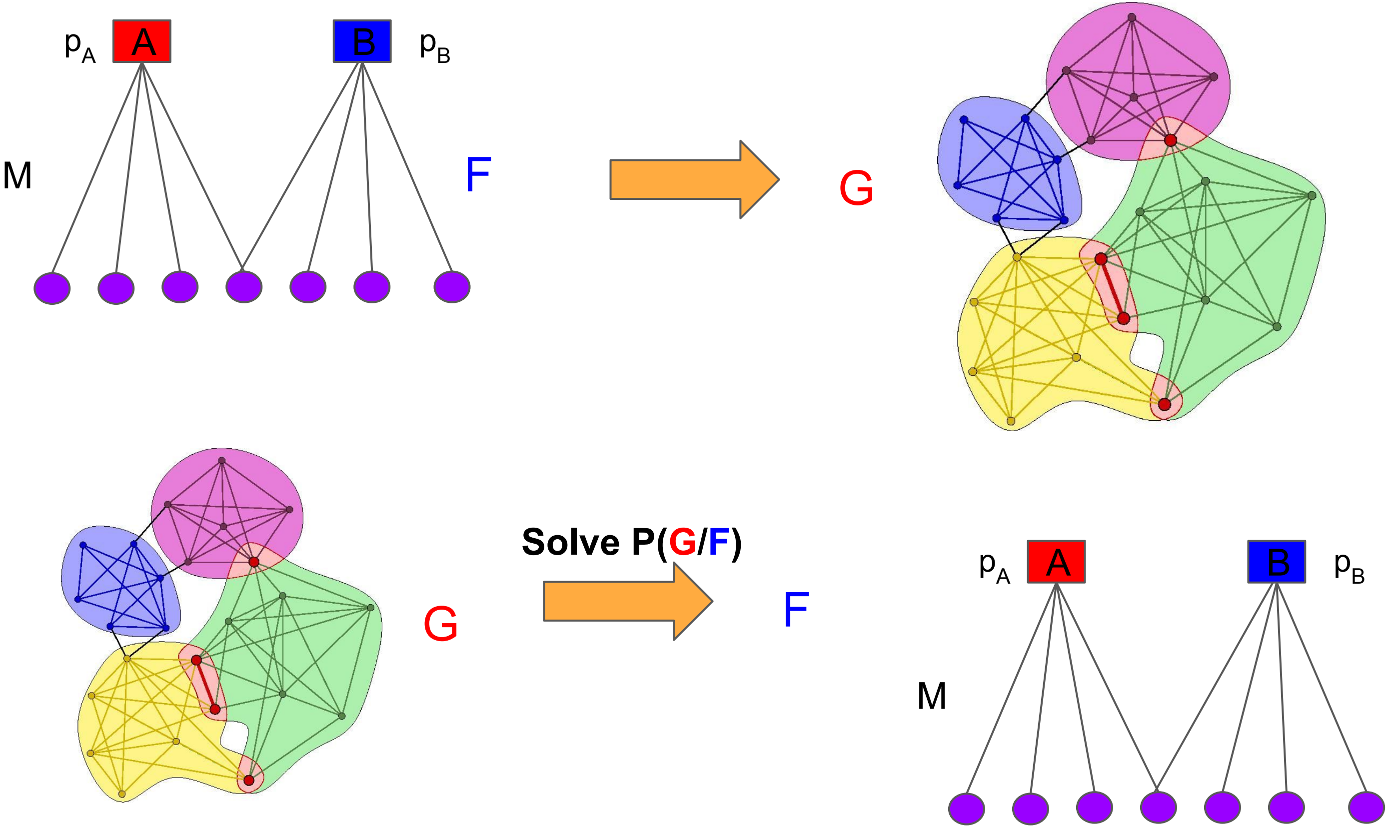
- Assume you know number of communities *C* that are there
- Assume you know membership *M* of nodes to each community.
- Let p_c be the probability that node u and v in community c have an edge between them
- This is our generative model F = (V, M, $\{p_c\}$)

$p(u, v)$ is the probability that two nodes have an edge between them.

$$p(u.v) = 1 - \prod_{c \in M_u \cap M_v} (1 - p_c)$$

To allow for edges to form between nodes that never belong to the same community, we assume all nodes belong to a background community with a small edge probability ϵ

AGM -> Graph, Graph -> AGM?



What do we know about G and F

- For G, the adjacency matrix

$$\begin{array}{cc} \begin{array}{ccccc} A & B & C & D & E \end{array} & \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} & \begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} \end{array}$$

- For a model F, we know edge probability

$$\begin{array}{cc} \begin{array}{ccccc} A & B & C & D & E \end{array} & \\ \begin{pmatrix} 0 & 0.3 & 0.4 & 0.52 & 0.42 \\ 0.3 & 0 & 0.45 & 0.45 & 0.12 \\ 0.4 & 0.45 & 0 & 0.23 & 0.45 \\ 0.52 & 0.45 & 0.23 & 0 & 0.38 \\ 0.42 & 0.12 & 0.45 & 0.38 & 0 \end{pmatrix} & \begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} \end{array}$$

Find model parameters which maximize P(G/F)

$$P(G/F) = \prod_{(u,v)\in G} P(u,v) \prod_{(u,v\notin G} (1 - P(u,v))$$

Probability that there every edge present in G
is generated by the model F

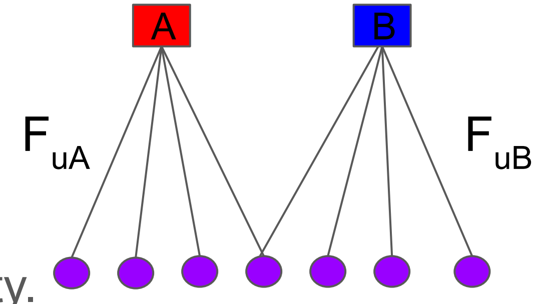
Probability that there every edge absent in G
is not generated by the model F

Relaxed AGM

- Relax the AGM constraint
- Instead of associating each community

  with a generative probability, assign an affinity for

  each node to a community. Need not be a probability.



Can we Generate P(u,v) from F_{uA} ?

- When two nodes have high affinity for same community, want high probability of edge being formed, else very small

- Set $P_C(u,v) = 1 - exp(-F_{uC}F_{vC})$
- $P_C(u,v)$ is a probability, $0 \leq P_C(u,v) \leq 1, because F_{uC}, F_{vC} \geq 0$
- $P_C(u,v) = 0$ if at least one of F_{uC} or F_{vC} is 0.
- $P_C(u,V) \approx 1$ iff $F_{uC}.F_{vC}$ is large. So if 2 nodes have high affinity for same community, they are likely to be connected.

- Recall: $P(u,v) = 1 - \prod_{C \in \mathbb{C}} (1 - P_C(u,v))$
- $\quad = 1 - \prod_{C \in \mathbb{C}} (1 - (1 - exp(-F_{uC} F_{vC}))$
- $\quad = 1 - \prod_{C \in \mathbb{C}} exp(-F_{uC} F_{vC})$
- $\quad = 1 - exp(-\sum_{C \in \mathbb{C}} F_{uC} F_{vC})$
- $\quad = 1 - exp(-F_u^T F_v)$

- Hence for a graph G, we maximize P(G/F)

- $P(G/F) = \prod_{(u,v)\in E} P(u,v) \prod_{(u,v) \not\in G)} (1 - P(u,v)$

- $= \prod_{(u,v)\in E} 1 - exp(-F_u^T F_v)$

Probability edge is in graph

$\times$

$\prod_{(u,v)\notin G} exp(-F_u^T F_v)$

Probability edge not in graph

Log likelihood

- Instead of maximizing $P(G/F)$, maximize $log(P(G/F))$, allows for numerical stability and easier math.
- $log(P(G/F)) = log(\prod_{(u,v)\in E}(1 - exp(-F_u^T F_v)) \prod_{(u,v)\notin G} exp(-F_u^T F_v))$
- $= \sum_{(u,v)\in E} log(1 - exp(-F_u^T F_v)) - \sum_{(u,v)\notin E} F_u^T F_v$

- Start with random F vector
- Do gradient ascent iteratively
- for every $u \in V$, differentiate w.r.t. u
- update F_u while keeping other F_i constant
- keep iterating until convergence
- gradient $= \sum_{v \in N(u)} \frac{exp(-F_u^T F_v)}{1-exp(-F_u^T F_v)} - \sum_{v \notin N(u)} F_v$