IT5429E-1-24 (24.1A01)(Fall 2024): Graph Analytics for Big Data

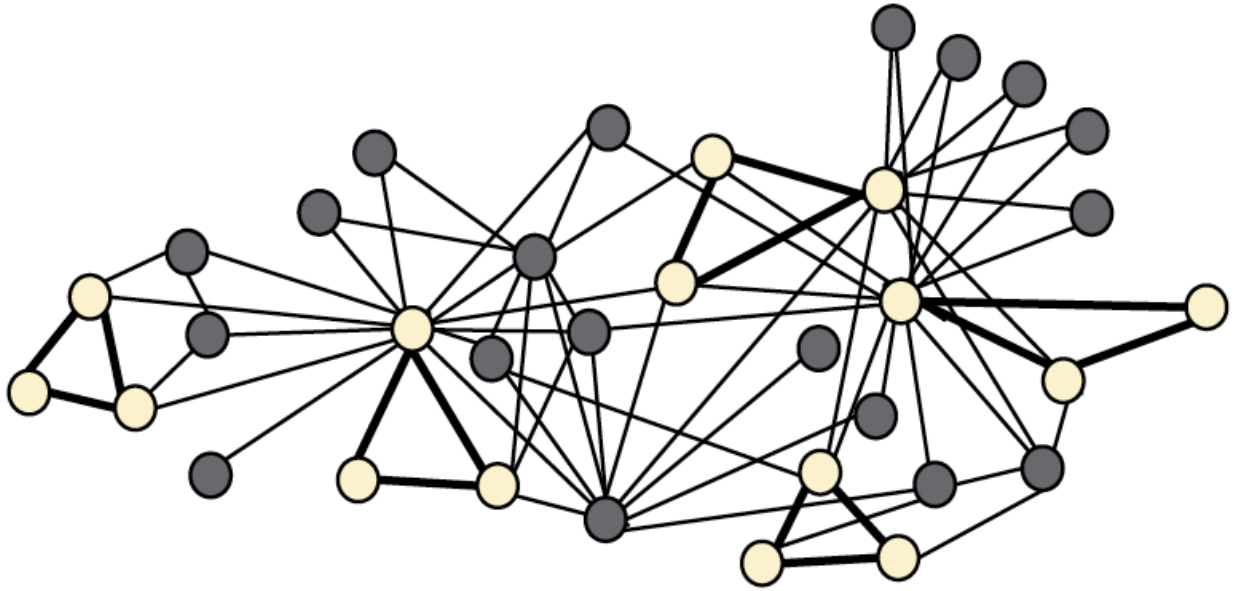
Week 8: Motifs and Community Detection

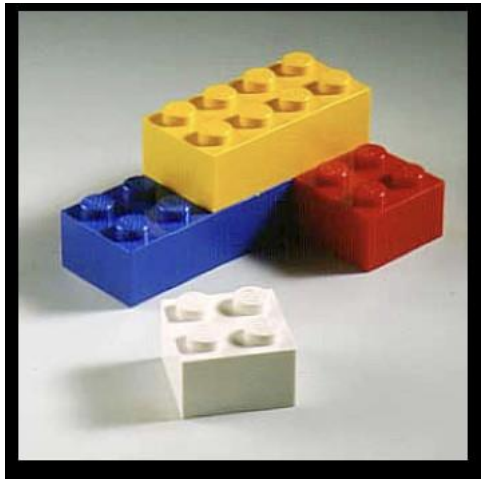
Instructor: Thanh H. Nguyen

Many slides are adapted from https://web.stanford.edu/class/cs224w/

Subgraphs

- Subgraphs are the building blocks of networks:





- They have the power to characterize and discriminate networks

Outline

1) Subgraphs and motifs
   - Defining Subgraphs and Motifs
   - Determining Motif Significance

2) Neural Subgraph Representations

3) Mining Frequent Motifs

Subgraphs and Motifs

Definition: Subgraph (1)

- Two ways to formalize "network building blocks"
- Given graph $G = (V, E)$:

Def 1. Node-induced subgraph: Take subset of the nodes and all edges induced by the nodes:

- $G' = (V', E')$ is a node-induced subgraph iff
  - $V' \in V$
  - $E' = \{(u, v) \in E \mid u, v \in V'\}$
  - G' is subgraph of G induced by V'

- Alternate terminology: "induced subgraph"

Definition: Subgraph (2)

- Two ways to formalize "network building blocks"
- Given graph $G = (V, E)$:

Def 1. Edge-induced subgraph: Take subset of the edges and all corresponding nodes:

- $G' = (V', E')$ is a edge-induced subgraph iff
  - $E' \in E$
  - $V' = \{v \in V \mid (v, u) \in E' for\ some\ u\}$
  - G' is subgraph of G induced by V'

- Alternate terminology: "induced subgraph"

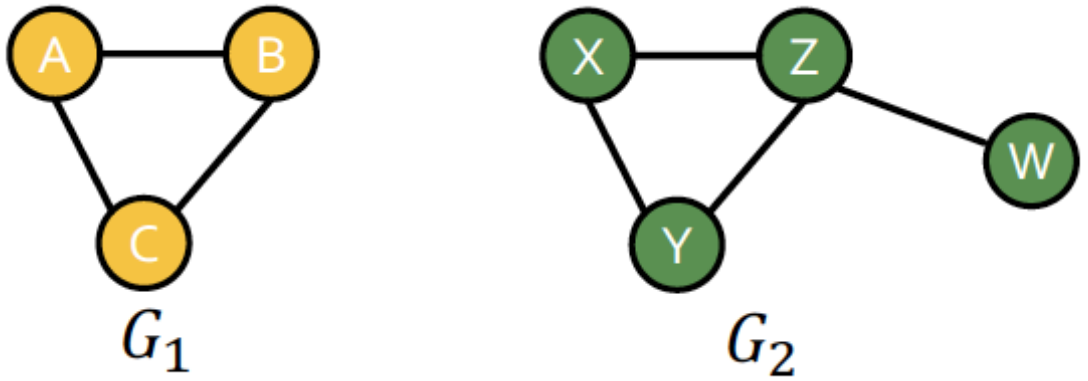
Definition: Subgraph (3)

- Two ways to formalize "network building blocks"

- The best definition depends on the domain!
- Examples:
  - Chemistry: Node-induced (functional groups)
  - Knowledge graphs: Often edge-induced (focus is on edges representing logical relations)

Definition: Subgraph (4)

- The preceding definitions define subgraphs when $V' \subseteq V$ and $E' \subseteq E$, i.e. nodes and edges are taken from the original graph G.
- What if $\boldsymbol{V'}$ and $\boldsymbol{E'}$ come from a totally different graph? Example:



G_1 G_2

- We would like to say that ***G1*** is "contained in" ***G2***

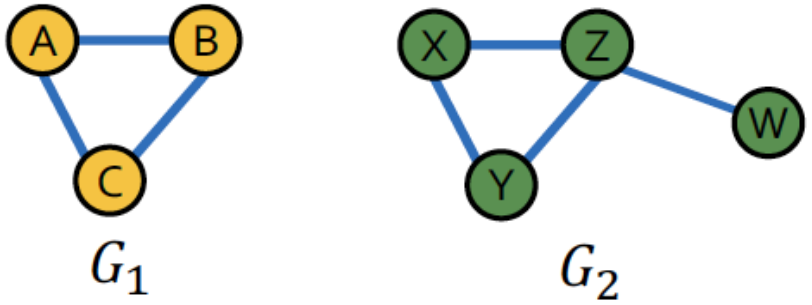
Graph Isomorphism

- Graph isomorphism problem: Check whether two graphs are identical:
  - $G_1 = (V_1, E_1)$ and $G_2 = (\mathrm{V}_2, \mathrm{E}_2)$ are isomorphic if there exists a bijection $f: V_1 \to V_2$ such that $(u, v) \in E_1$ iff $(f(u), f(v)) \in E_2$
  - f is called the isomorphism:

- We do not know if graph isomorphism is NP-hard, nor is any polynomial algorithm found for solving graph isomorphism.

Subgraph Isomorphism

- G_2 is subgraph-isomorphic to G_1 if some subgraph of G_2 is isomorphic to G_1
  - We also commonly say G_1 is a subgraph of G_2
  - We can use either the node-induced or edge-induced definition of subgraph
  - This problem is NP-hard



G_1 G_2

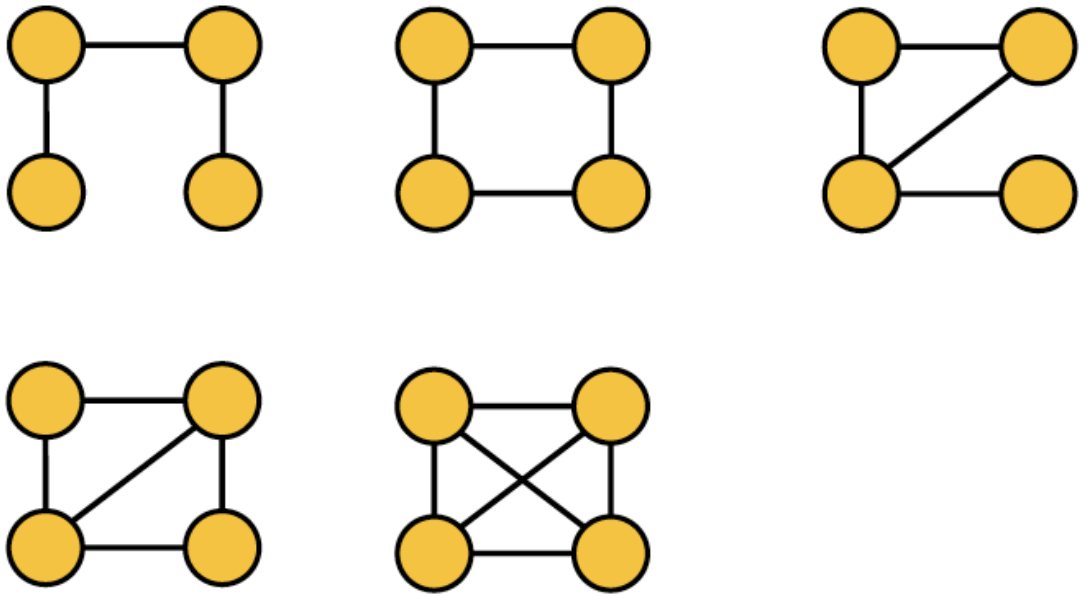
f:

V_1	V_2
A	X
B	Y
C	Z

A-B-C matches with X-Y-Z: There is a subgraph isomorphism between G1 and G2.

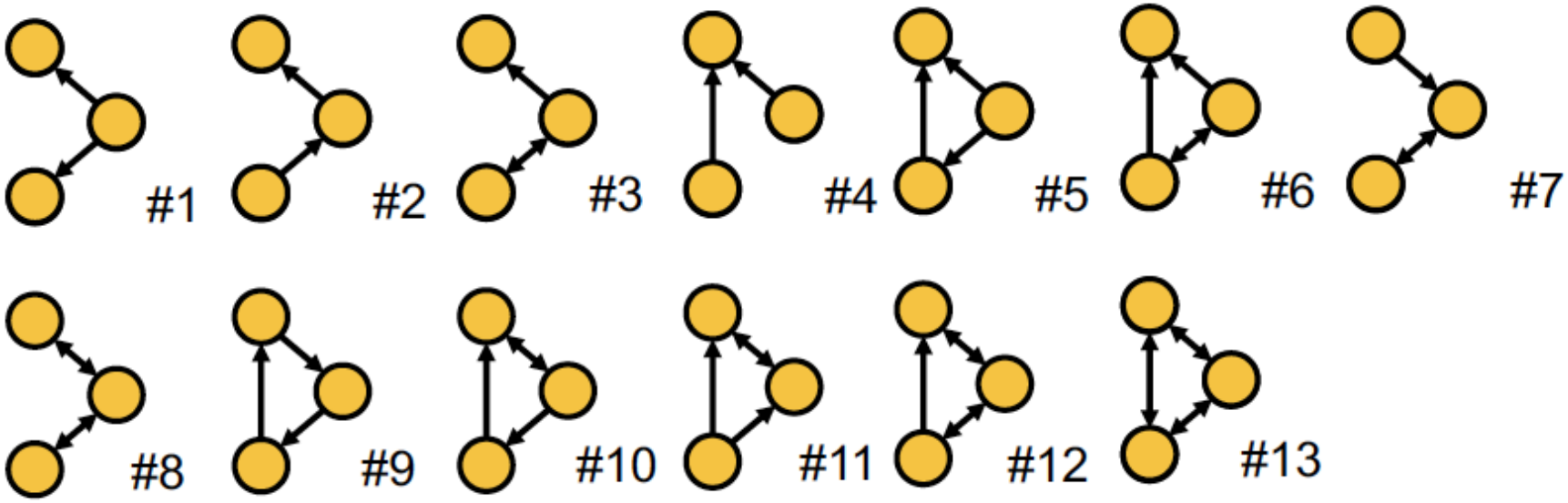
Example of Subgraphs (1)

All non-isomorphic, connected, undirected graphs of size 4



Example of Subgraphs (2)

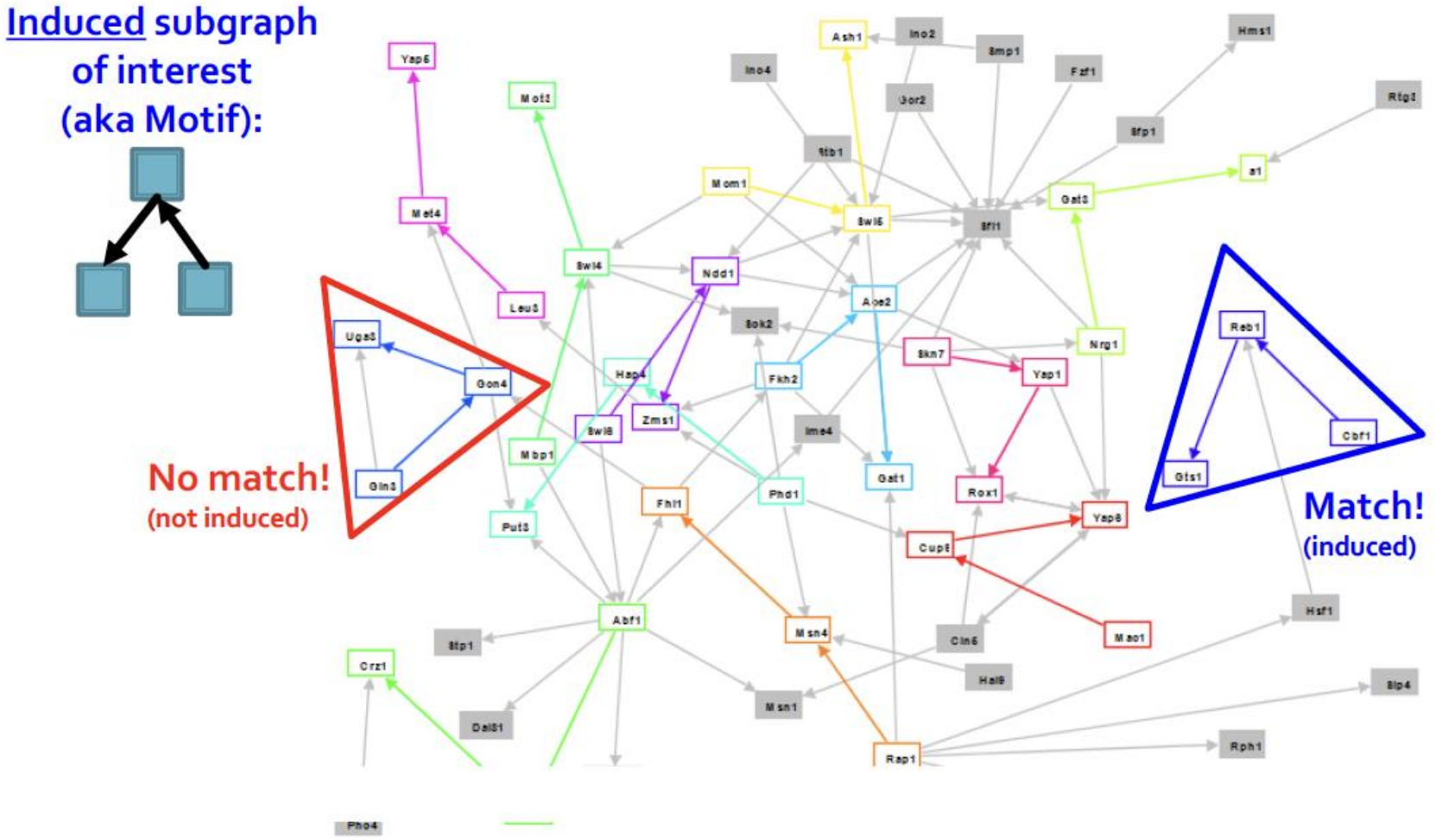
All non-isomorphic, connected, directed graphs of size 3



Network Motifs

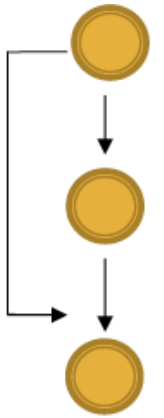
- Network motifs: "recurring, significant patterns of interconnections"
- How to define a network motif:
  - Pattern: Small (node-induced) subgraph
  - Recurring: Found many times, i.e., with high frequency
    - How to define frequency?
  - Significant: More frequent than expected, i.e., in randomly generated graphs?
    - How to define random graphs?

Motifs: Induced Subgraphs

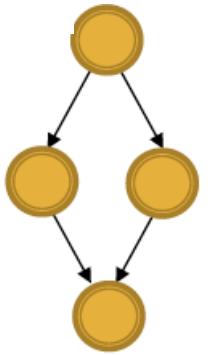


Why do We Need Motifs?

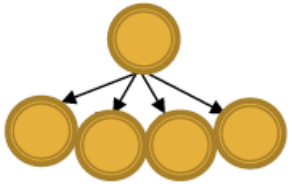
- Motifs:
  - Help us understand how graphs work
  - Help us make predictions based on presence or lack of presence in a graph dataset
- Examples:
  - Feed-forward loops: Found in networks of neurons, where they neutralize "biological noise"
  - Parallel loops: Found in food webs
  - Single-input modules: Found in gene control networks



Feed-forward loop



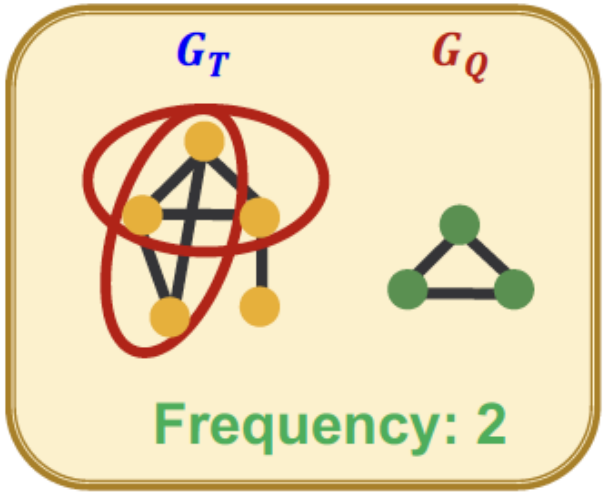
Parallel loop

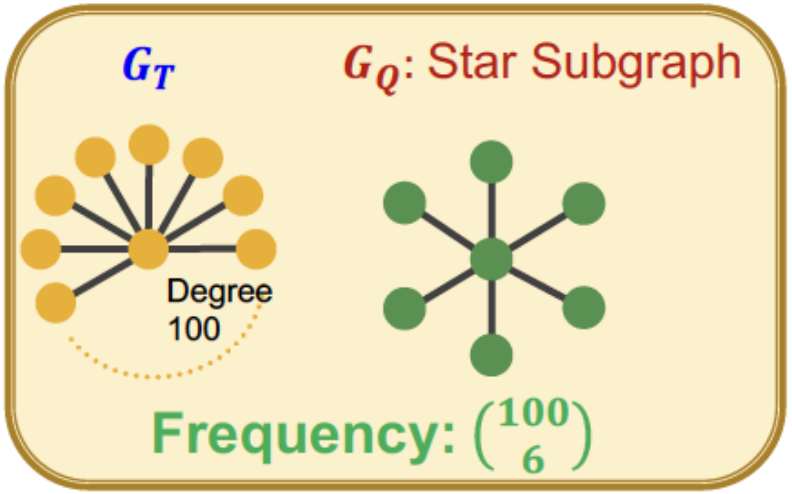


Single-input module

Subgraph Frequency (1)

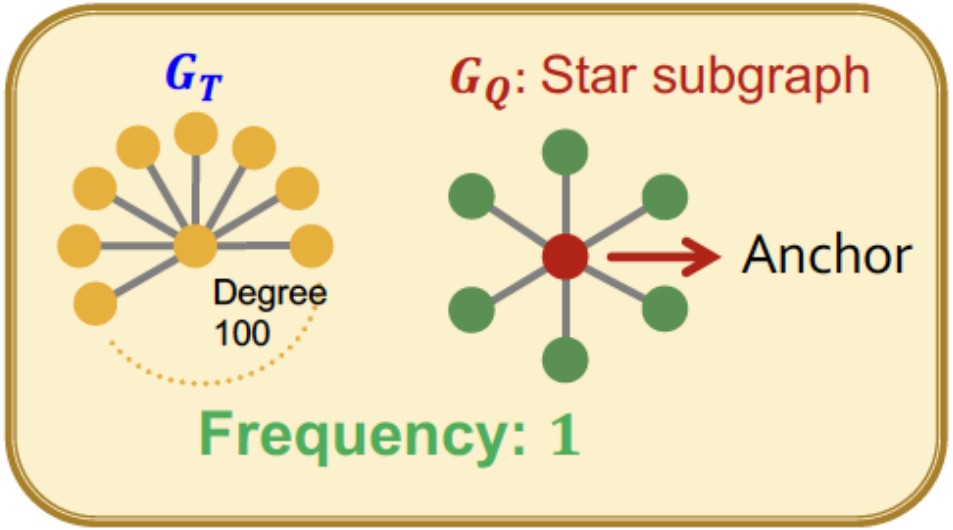
- Let G_Q be a small graph and G_T be a target graph dataset.
- Graph-level Subgraph Frequency Definition
  - Frequency of G_Q in G_T : number of unique subsets of nodes V_T of G_T for which the subgraph of G_T induced by the nodes V_T is isomorphic to G_Q





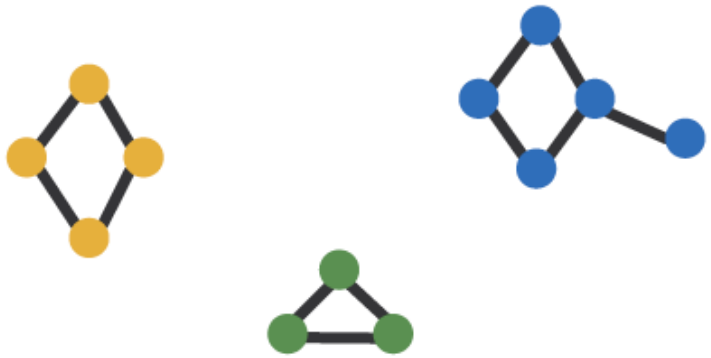
Subgraph Frequency (2)

- Let G_Q be a small graph, v be a node in G_Q (the "anchor") and G_T be a target graph dataset.
- Node-level Subgraph Frequency Definition:
  - The number of nodes u in G_T for which some subgraph of G_T is isomorphic to G_Q and the isomorphism maps node u to v
  - Let (G_Q, v) be called a node-anchored subgraph
  - Robust to outliers



Subgraph Frequency (3)

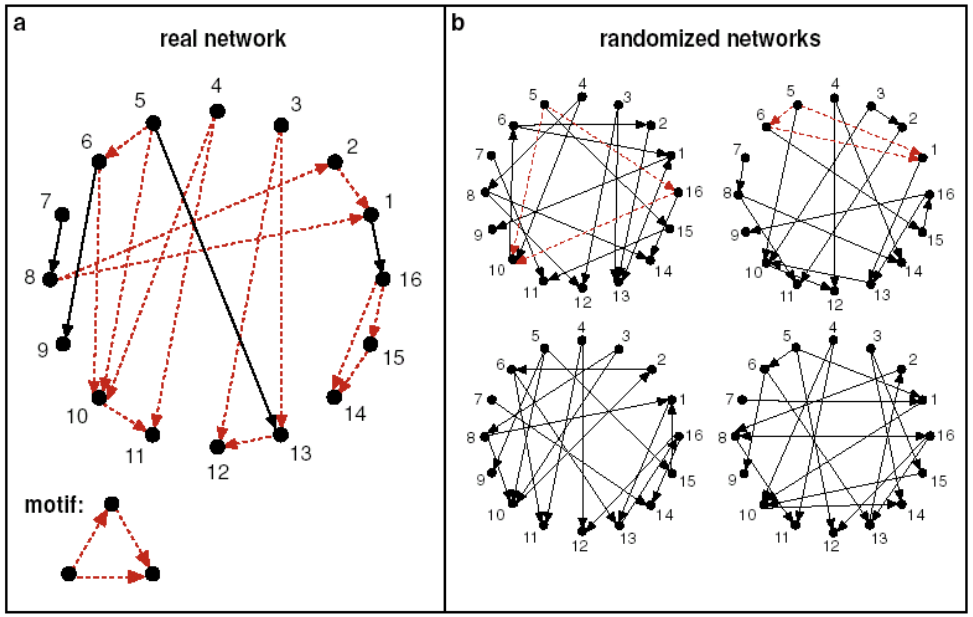
- What if the dataset contains multiple graphs, and we want to compute frequency of subgraphs in the dataset?
- Solution: Treat the dataset as a giant graph G_T with disconnected components corresponding to individual graphs.



Defining Motif Significance

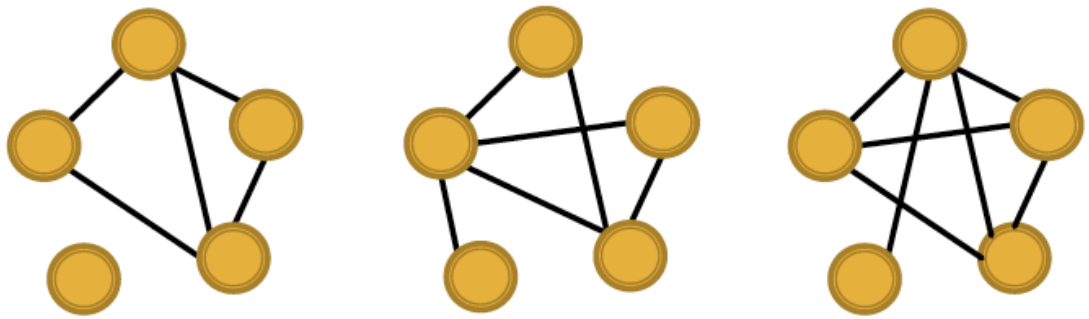
- To define significance, we need to have a null-model (i.e., point of comparison).
- Key idea: Subgraphs that occur in a real network much more often than in a random network have functional significance.

Fig. 2



Defining Random Graphs

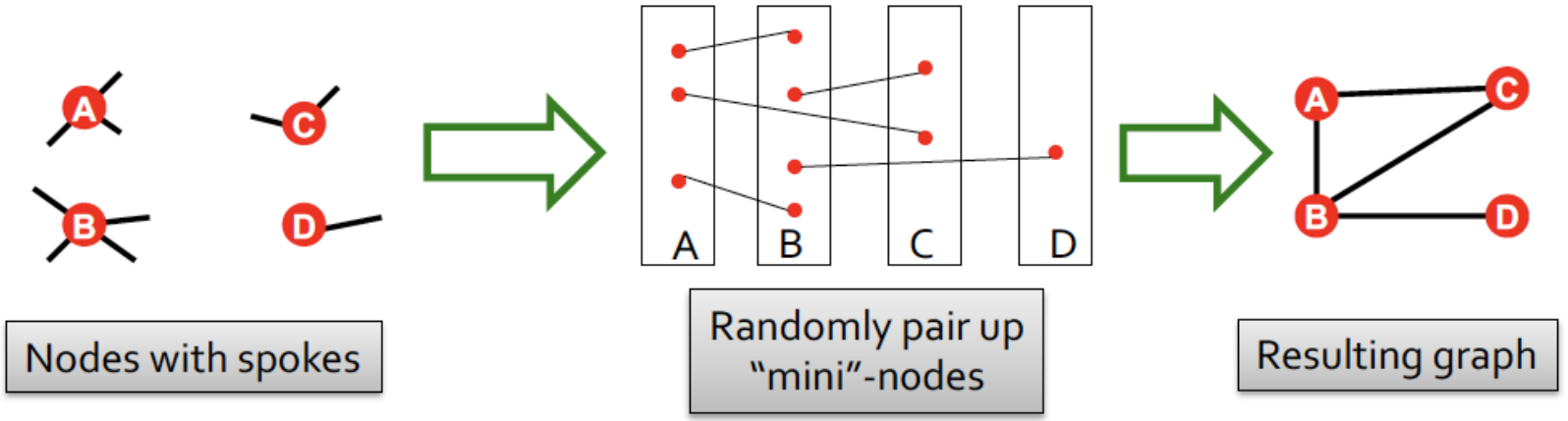
- Erdős–Rényi (ER) random graphs:
- $G_{n,p}$: undirected graph on n nodes where each edge (u, v) appears i.i.d. with probability p
  - How to generate the graph: Create n nodes, for each pair of nodes u, v flip a biased coin with bias p
- Generated graph is a result of a random process:



Three random graphs drawn from $G_{5,0.6}$

New Model: Configuration Model

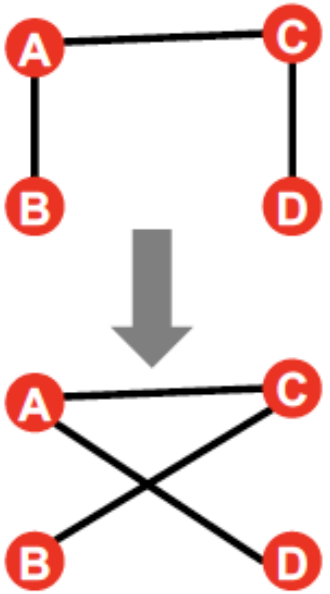
- Goal: Generate a random graph with a given degree sequence $k_1, k_2, \ldots k_N$
- Useful as a "null" model of networks:
  - We can compare the real network G^{real} and a "random" G^{rand} which has the same degree sequence as G^{real}
- Configuration model:



We ignore double edges and self-loops when creating the final graph

Alternative to Spokes: Switching

- Start from a given graph G
- Repeat the switching step $Q \cdot |E|$ times:
  - Select a pair of edges A→B, C → D at random
  - Exchange the endpoints to give A → D, C → B
  - Exchange edges only if no multiple edges or self-edges are generated
- Result: A randomly rewired graph:
  - Same node degrees, randomly rewired edges
- Q is chosen large enough (e.g., $Q = 100$) for the process to converge



Motif Significance Overview

- Intuition: Motifs are overrepresented in a network when compared to random graphs:
- Step 1: Count motifs in the given graph (G^{real})
- Step 2: Generate random graphs with similar statistics (e.g. number of nodes, edges, degree sequence), and count motifs in the random graphs
- Step 3: Use statistical measures to evaluate how significant is each motif
  - Use Z-score

Z-score for Statistical Significance

- Z_i captures statistical significance of motif ***i***:

$$Z_i = \frac{N_i^{real} - \bar{N}_i^{rand}}{std\left(N_i^{rand}\right)}$$

  - N_i^{real} is #(motif i) in graph G^{real}
  - $\bar{N}_i^{rand}$ is average #(motifs i) in random graph instances

- Network significance profile (SP):

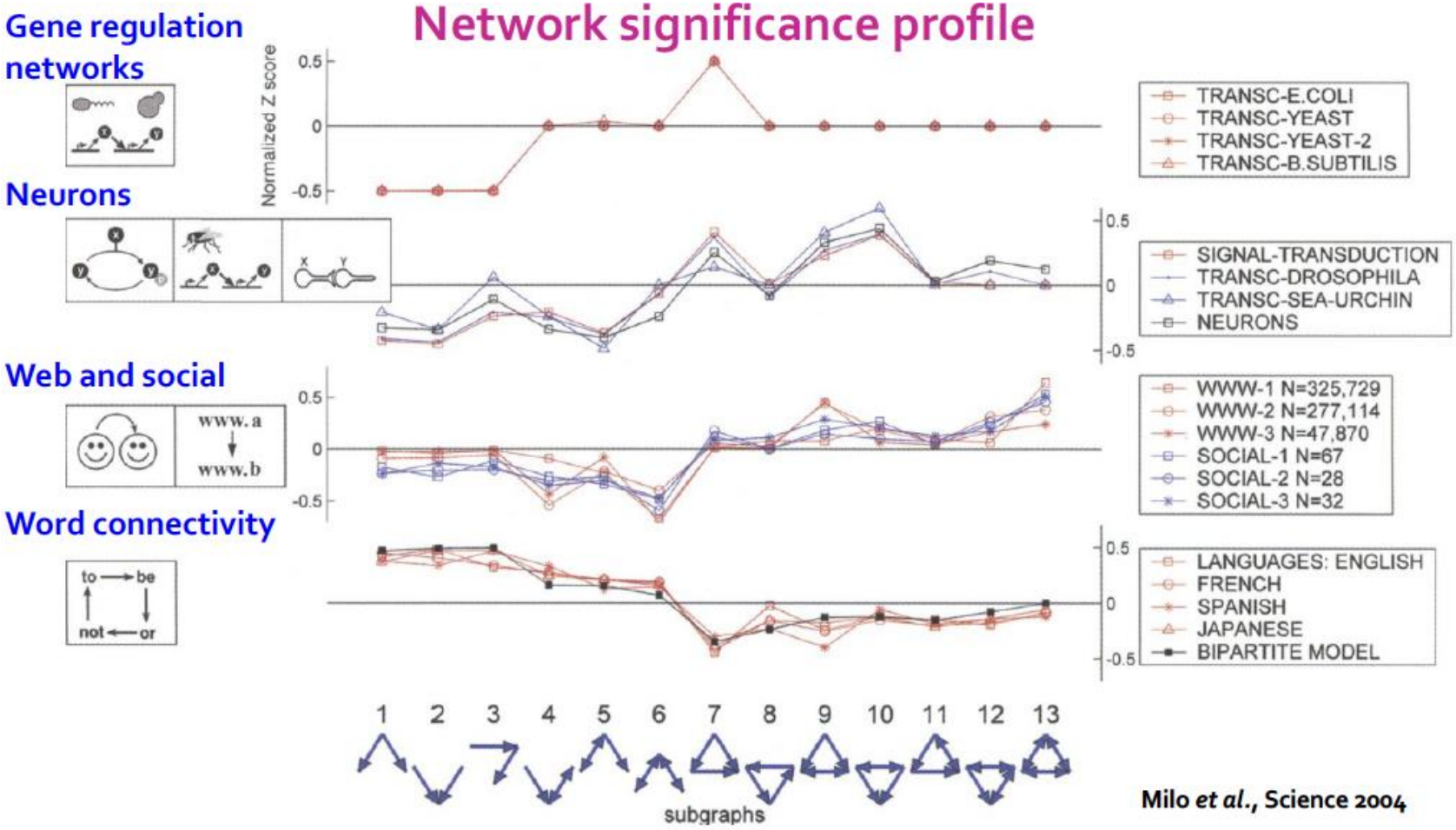
$$SP_i = \frac{Z_i}{\sqrt{\sum_j Z_j^2}}$$

  - SP is a vector of normalized Z-scores
  - The dimension depends on number of motifs considered
  - SP emphasizes relative significance of subgraphs:
    - Important for comparison of networks of different sizes
    - Generally, larger graphs display higher Z-scores

Significance Profile

- For each subgraph:
  - z-score metric is capable of classifying the subgraph "significance":
  - Negative values indicate under-representation
  - Positive values indicate over-representation
- We create a network significance profile:
  - A feature vector with values for all subgraph types
- Next: Compare profiles of different graphs with random graphs:
  - Regulatory network (gene regulation)
  - Neuronal network (synaptic connections)
  - World Wide Web (hyperlinks between pages)
  - Social network (friendships)
  - Language networks (word adjacency)

Example: Significance Profile



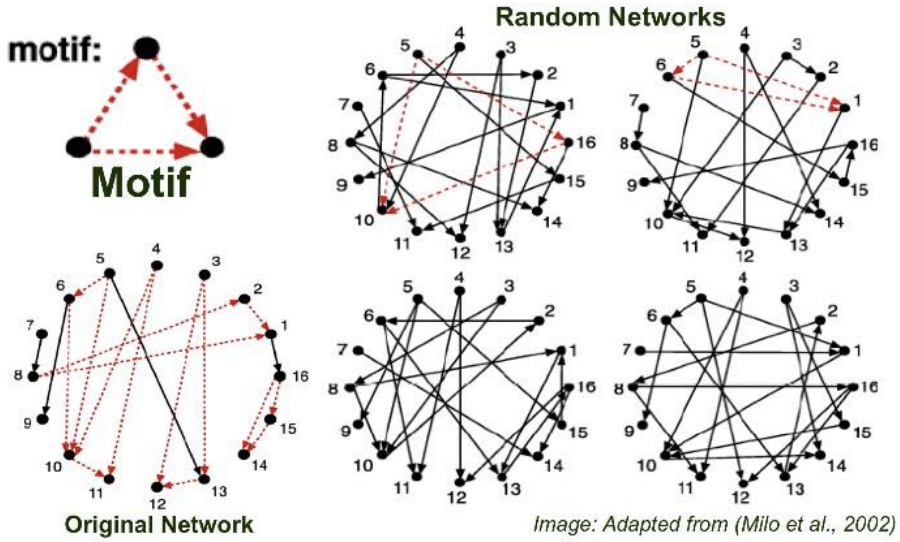
Networks from the same domain have similar significance profiles

Summary: Detecting Motifs

- Count subgraphs i in G^{real}
- Count subgraphs i in random graphs G^{rand}
  - Null model: Each G^{rand} has the same #(nodes), #(edges) and degree distribution as G^{real}
- Assign Z-score to motif i:

$$Z_i = \frac{N_i^{real} - \overline{N}_i^{rand}}{std\left(N_i^{rand}\right)}$$

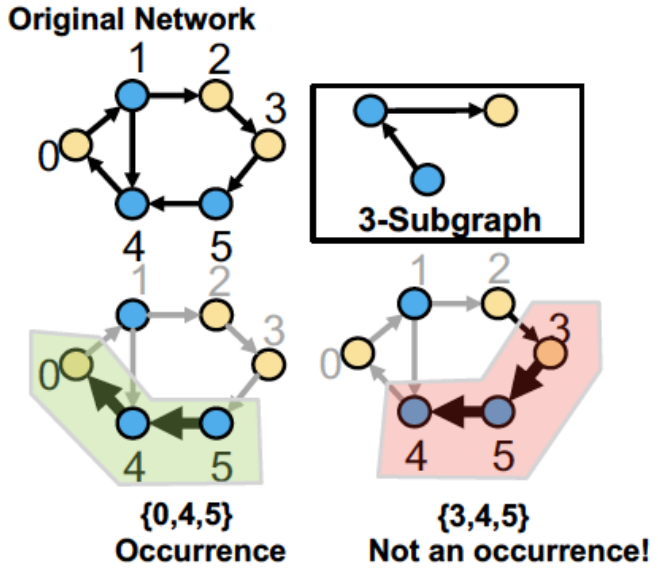
  - High Z-score: Subgraph i is a network motif of $\boldsymbol{G}$

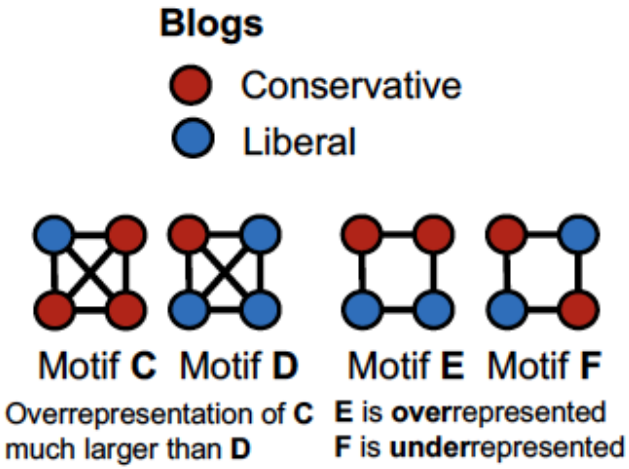


Variations on the Motif Concept

- Extensions:
  - Directed and undirected
  - Colored and uncolored
  - Temporal and static motifs

- Variations on the concept:
  - Different frequency concepts
  - Different significance metrics
  - Under-Representation (anti-motifs)
  - Different null models





Summary: Motifs

- Subgraphs and motifs are the building blocks of graphs
  - Subgraph isomorphism and counting are NP-hard
- Understanding which motifs are frequent or significant in a dataset gives insight into the unique characteristics of that domain
- Use random graphs as null model to evaluate the significance of motif via Z-score

Neural Subgraph Matching

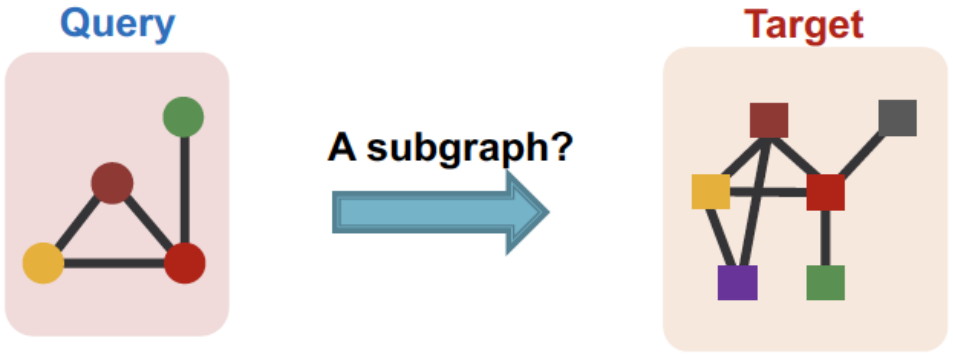
Subgraph Matching

Given:

- Large target graph (can be disconnected)
- Query graph (connected)

Decide:

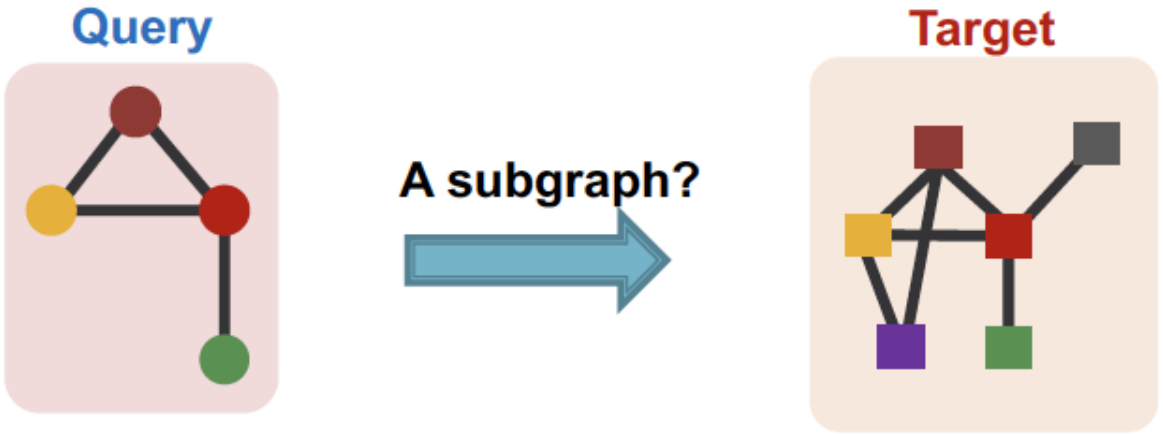
- Is a query graph a subgraph in the target graph?



  - Node colors indicate the correct mapping of the nodes

Isomorphism as an ML Task

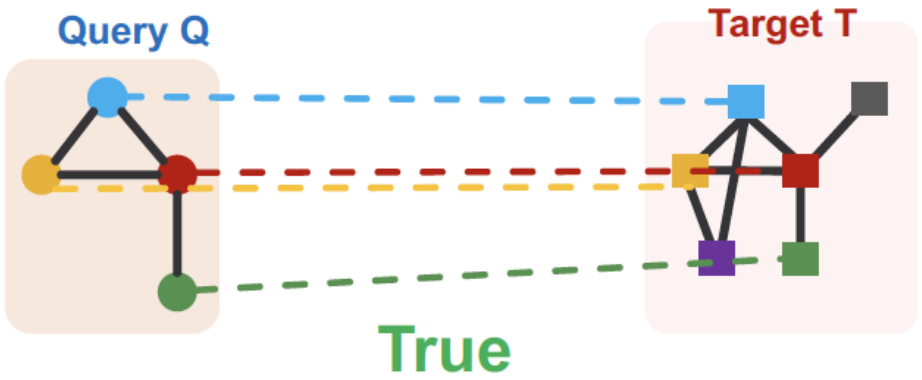
- Large target graph (can be disconnected)
- Query graph (has to be connected)
- Use GNN to predict subgraph isomorphism:



- Intuition: Exploit the geometric shape of embedding space to capture the properties of subgraph isomorphism

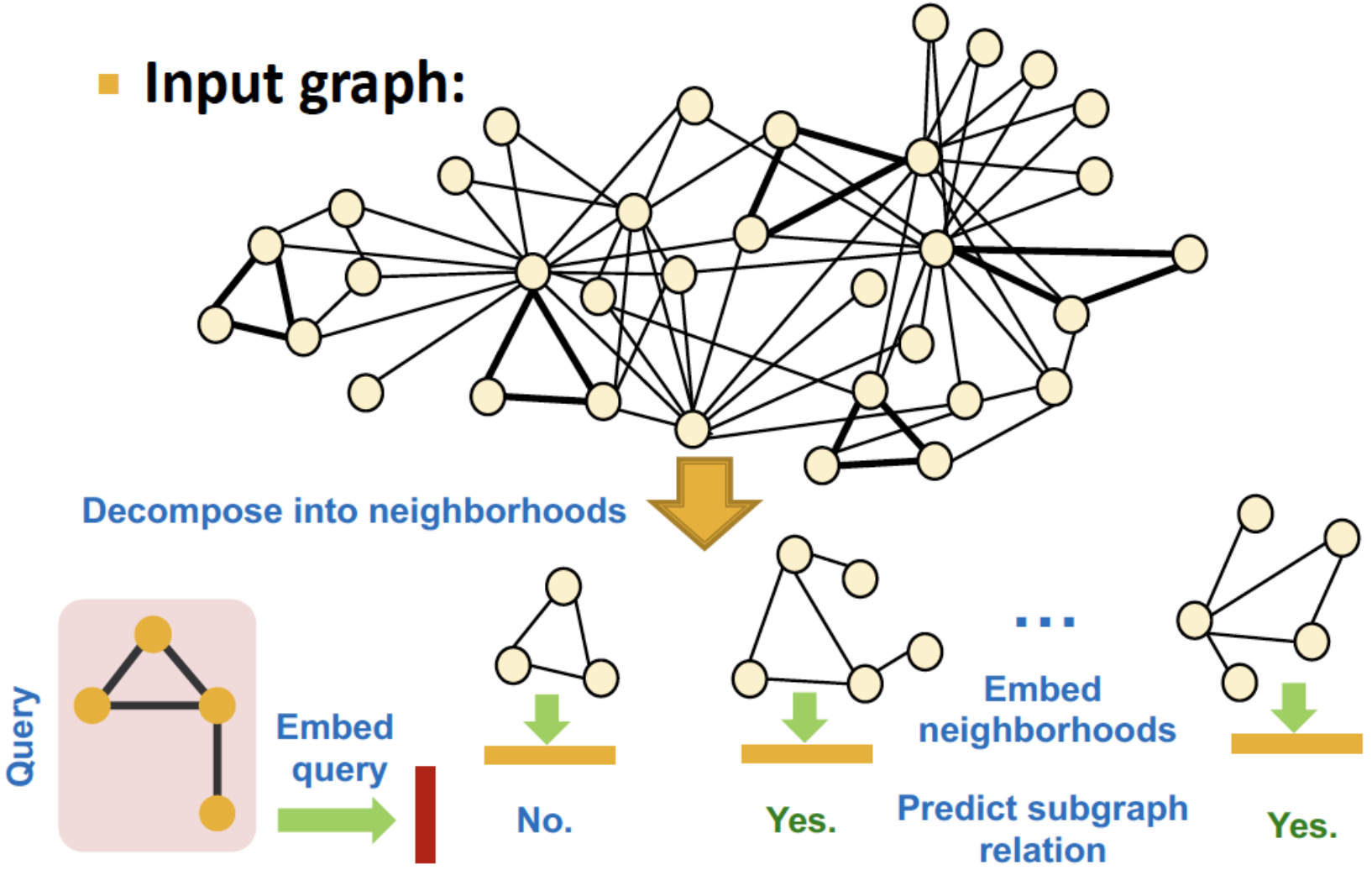
Task Setup

- Consider a binary prediction: Return True if query is isomorphic to a subgraph of the target graph, else return False



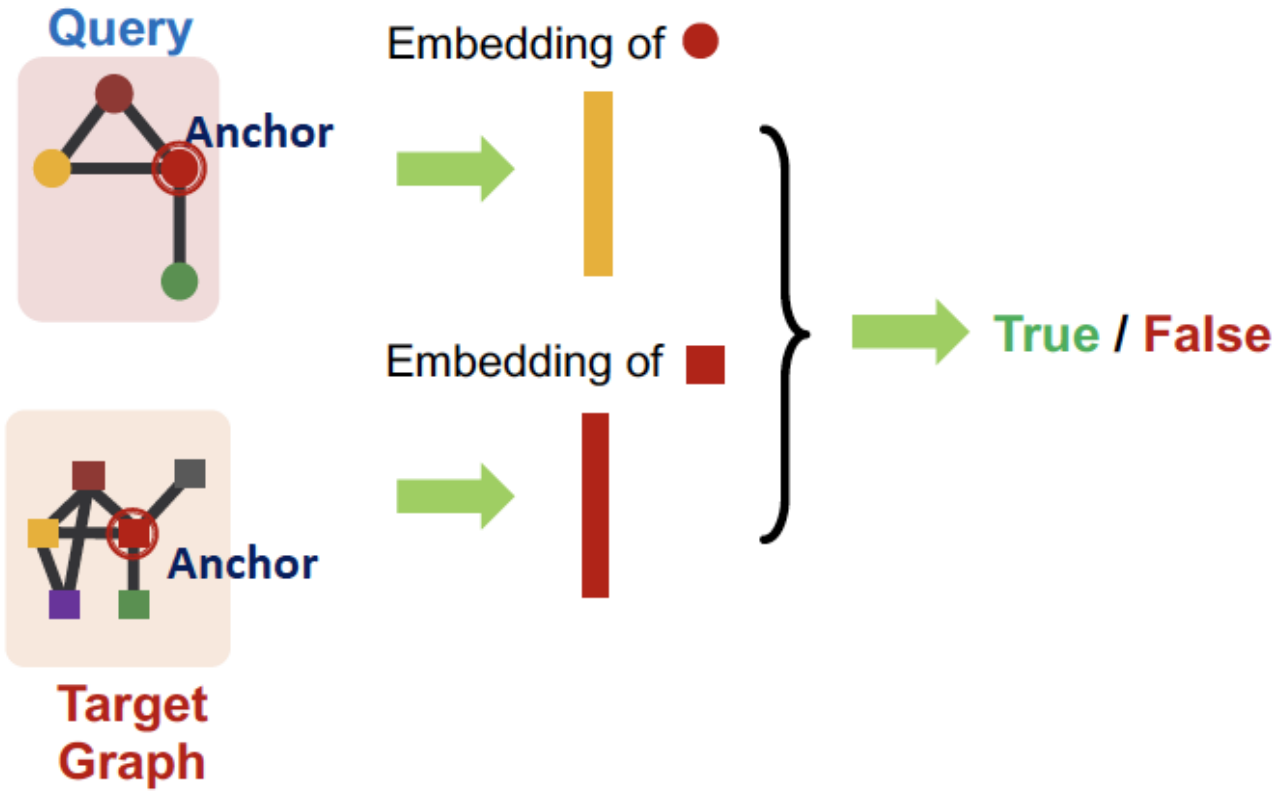
Finding node correspondences between Q and T is another challenging problem, which will not be covered in this lecture.

Overview of the Approach



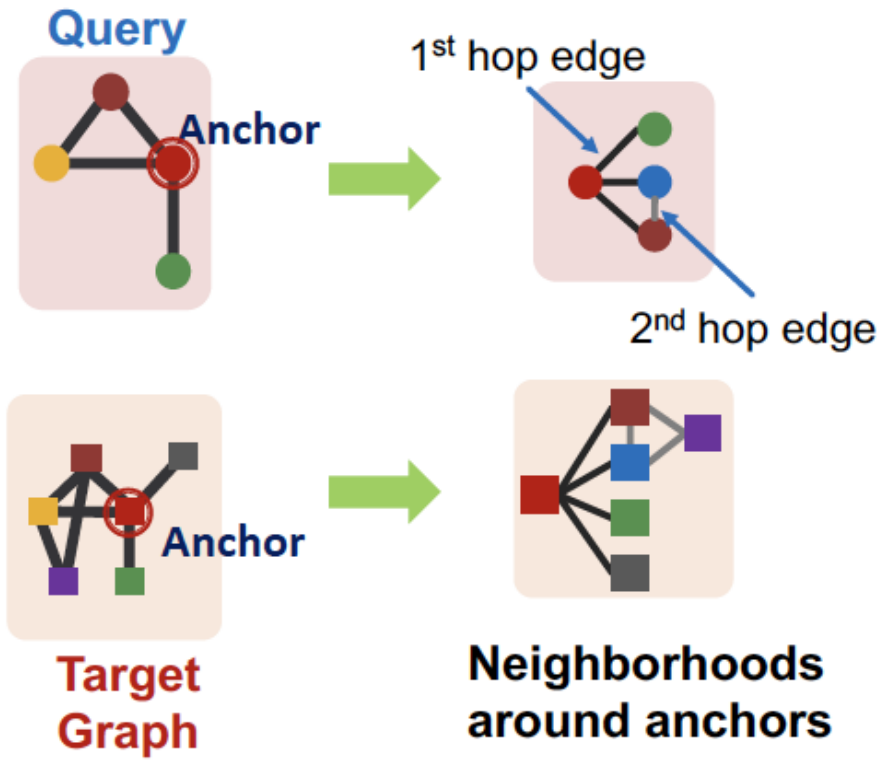
Neural Architecture for Subgraphs (1)

- (1) We are going to work with node-anchored definitions:



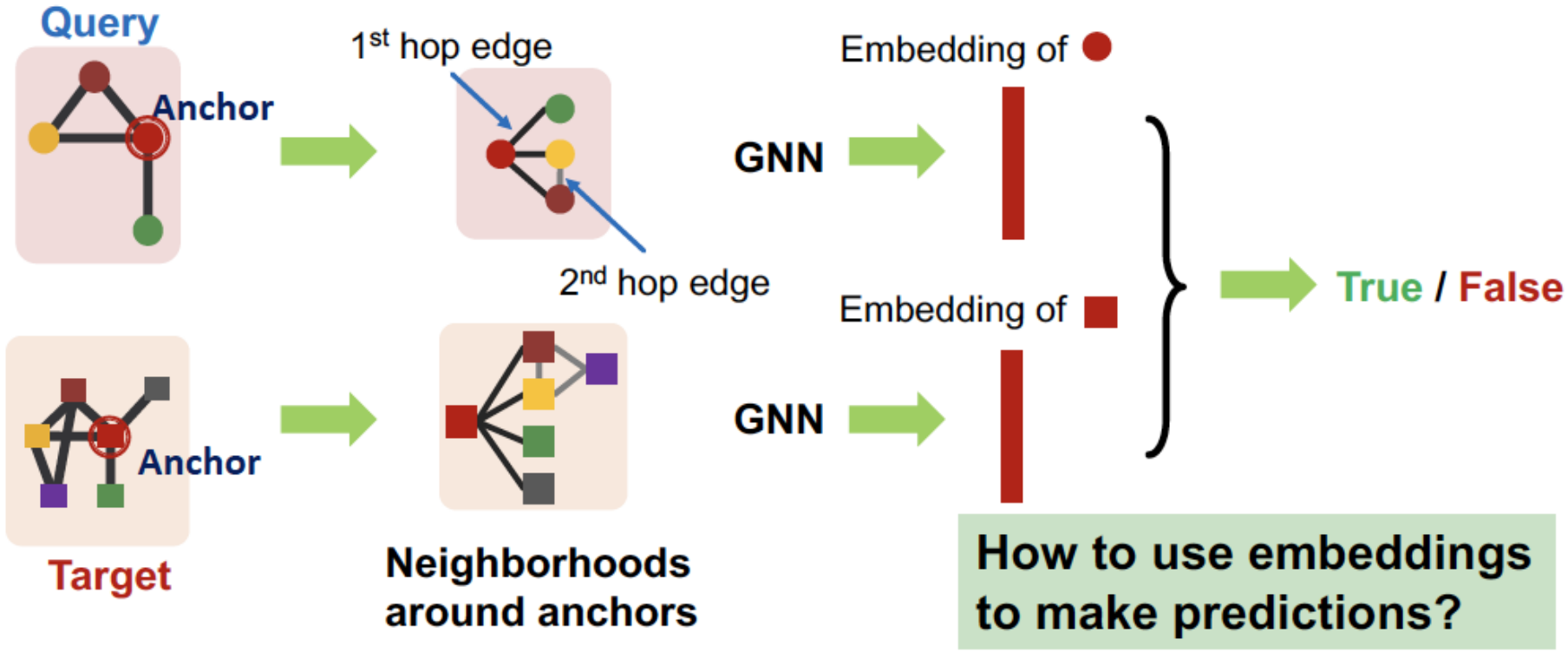
Neural Architecture for Subgraphs (2)

- (2) We are going to work with node-anchored neighborhoods:



Neural Architecture for Subgraphs (3)

- Use GNN to obtain representations of u and v
- Predict if node $\boldsymbol{u}$'s neighborhood is isomorphic to node $\boldsymbol{v}$'s neighborhood:



Why Anchor?

- Recall node-level frequency definition:
- The number of nodes u in G_T for which some subgraph of G_T is isomorphic to G_Q and the isomorphism maps u to v
- We can compute embeddings for u and v using GNN
- Use embeddings to decide if neighborhood of u is isomorphic to subgraph of neighborhood of v
- We not only predict if there exists a mapping, but also a identify corresponding nodes (u and v)!

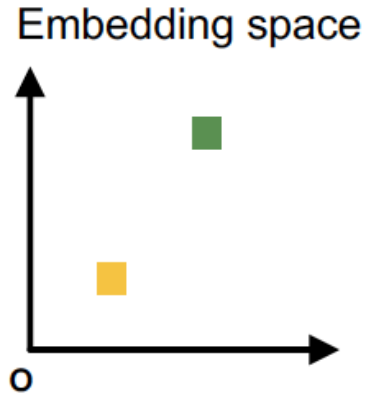
Decomposing G_T into Neighborhoods

- For each node in $\boldsymbol{G_T}$:
  - Obtain a k-hop neighborhood around the anchor
  - Can be performed using breadth-first search (BFS)
  - The depth k is a hyper-parameter (e.g. 3)
  - Larger depth results in more expensive model
- Same procedure applies to G_Q, to obtain the neighborhoods
- We embed the neighborhoods using a GNN
  - By computing the embeddings for the anchor nodes in their respective neighborhoods

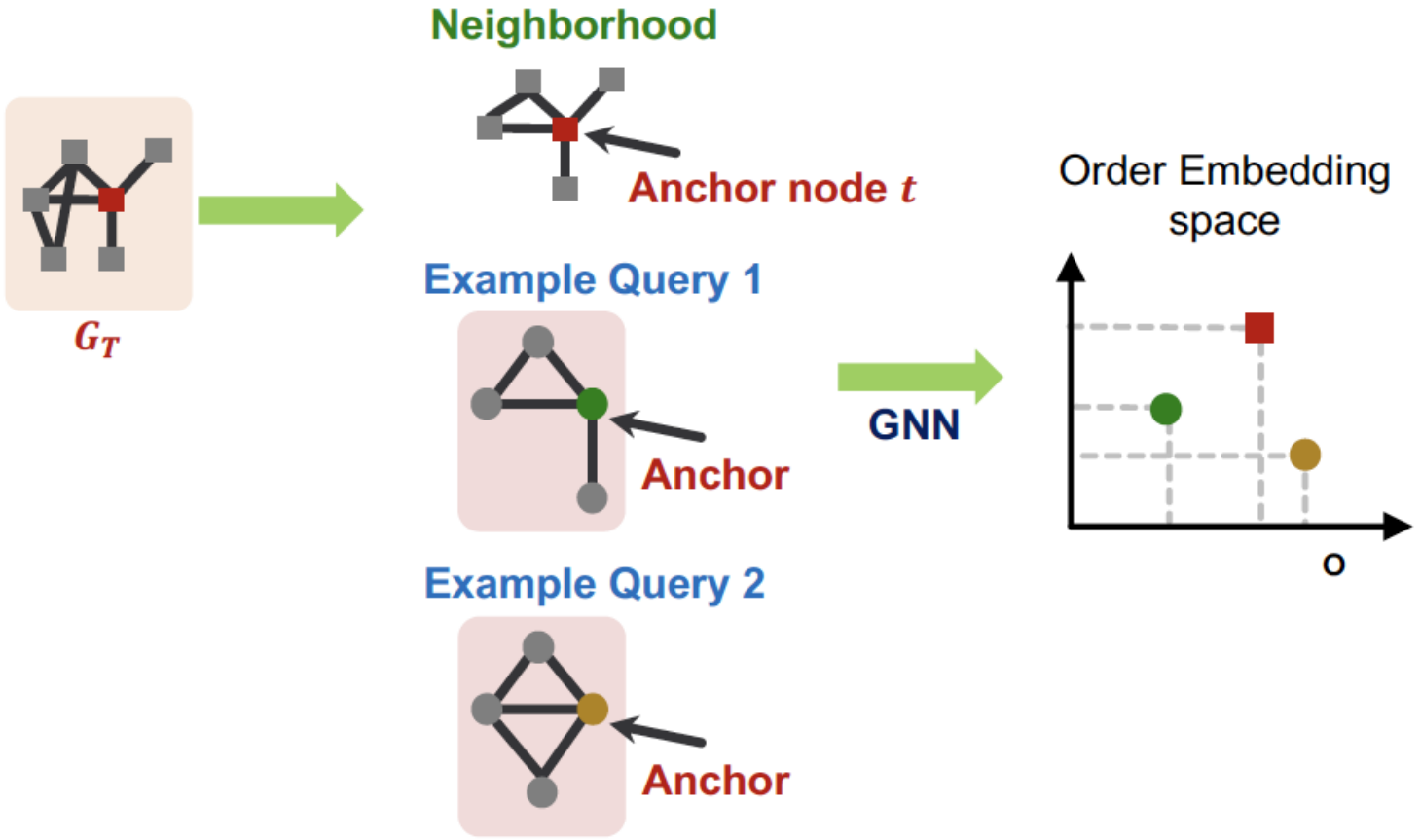
Idea: Order Embedding Space

- Map graph A to a point z_A into a high-dimensional (e.g. 64-dim) embedding space, such that z_A is non-negative in all dimensions
- Capture partial ordering (transitivity):
  - We use $z_A \preceq z_B$ to denote that the embedding of A is less than or equal to B in all of its coordinates
- If $z_A \preceq z_B$, $z_B \preceq z_C$ then $z_A \preceq z_C$

Intuition: subgraph is to the lower-left of its supergraph (in 2D)



Subgraph Order Embedding Space



By comparing the embedding, we find that ● ≺ ■ but ● ⊀ ■ ,
Indicating that only query 1 is a subgraph of the neighborhood of t

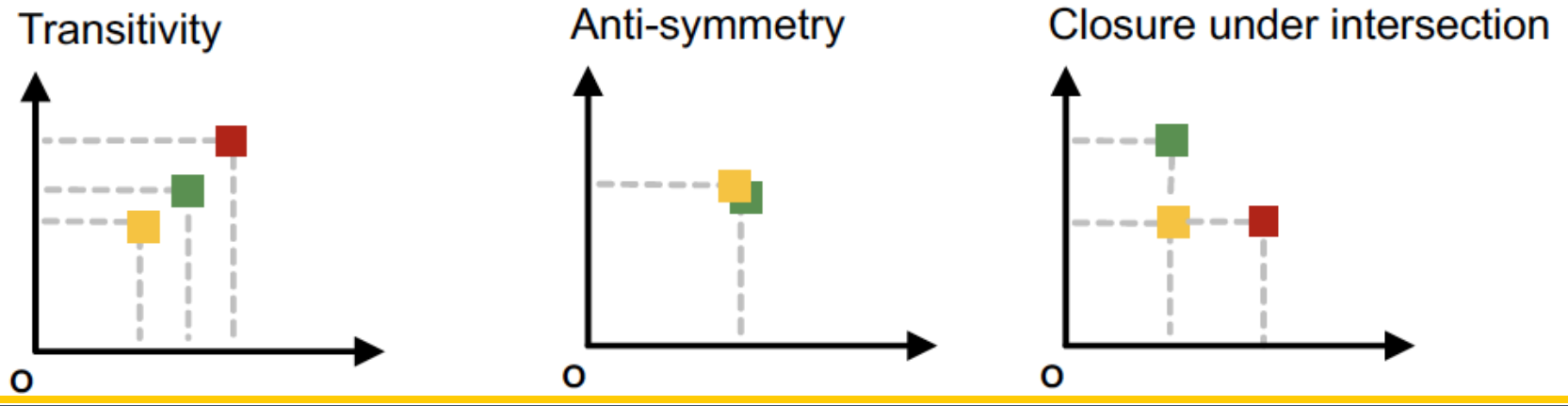
Why Order Embedding Space?

- Subgraph isomorphism relationship can be nicely encoded in order embedding space
  - Transitivity: If G_1 is a subgraph of G_2, G_2 is a subgraph of G_3, then G_1 is a subgraph of G_3
  - Anti-symmetry: If G_1 is a subgraph of G_2, and G_2 is a subgraph of G_1, then G_1 is isomorphic to G_2
  - Closure under intersection: The trivial graph of 1 node is a subgraph of any graph
  - All properties have their counter-parts in the order embedding space

Why Order Embedding Space?

- **Subgraph isomorphism relationship** can be nicely encoded in order embedding space
  - **Transitivity**: If $u \preceq v$, $v \preceq w$ then $u \preceq w$
  - **Anti-symmetry**: If $u \preceq v$ and $v \preceq u$, then $u = v$
  - **Closure under intersection**: The 0 embedding satisfies $0 \preceq u$ for any order embedding u since all dimensions of order embedding are non-negative
    - Corollary: If $u \preceq v$ and $u \preceq w$ then u has a valid embedding

0 embedding: Trivial graph with one node



Order Constraint (1)

- We use a GNN to learn to embed neighborhoods and preserve the order embedding structure

- What loss function should we use, so that the learned order embedding reflects the subgraph relationship?

- We design loss functions based on the order constraint:
  - Order constraint specifies the ideal order embedding property that reflects subgraph relationships

Order Constraint (2)

- We specify the order constraint to ensure that the subgraph properties are preserved in the order embedding space

Embedding dimension

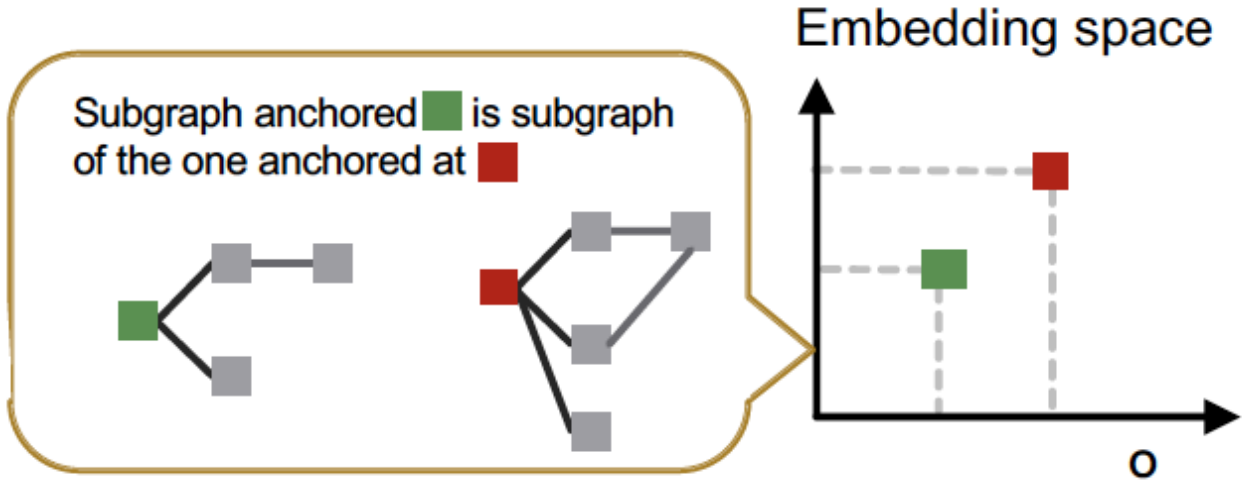
$$\forall_{i=1}^{D} z_q[i] \leq z_t[i] \quad \text{iff} \quad G_Q \subseteq G_T$$

trained with max-margin loss

Query embedding

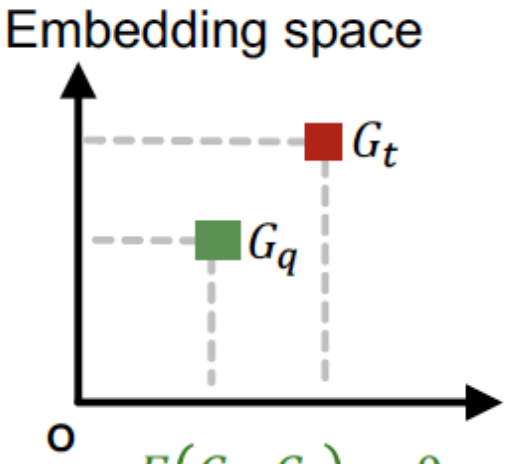
Target embedding

Subgraph Relation



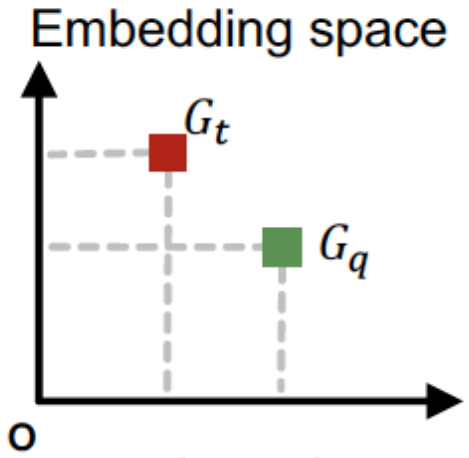
Loss Function: Order Constraint

- GNN Embeddings are learned by minimizing a max margin loss
- Define: $E(G_q, G_t) = \sum_{i=1}^{D} \left(\max(0, z_q[i] - z_t[i])\right)^2$ as the "margin" between graphs G_q and G_t



$E(G_q, G_t) = 0$

According to the order embedding, G_q is a subgraph of G_t!



$E(G_q, G_t) > 0$

According to the order embedding, G_q is **not** a subgraph of G_t!

Loss Function

- Embeddings are learned by minimizing a max margin loss
- Let $E(G_q, G_t) = \sum_{i=1}^{D}\left(\max(0, z_q[i] - z_t[i])\right)^2$ be the "margin" between graphs G_q and G_t

- To learn the correct order embeddings, we want to learn $\boldsymbol{z_q}$, $\boldsymbol{z_t}$ such that
  - $E(G_q, G_t) = 0$ when G_q is a subgraph of G_t
  - $E(G_q, G_t) > 0$ when G_q is not a subgraph of G_t

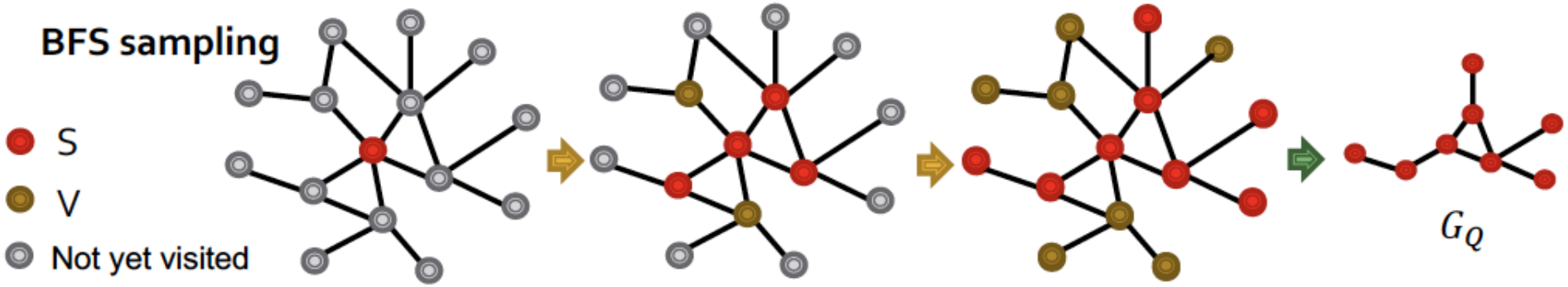
Training Neural Subgraph Matching

- To learn such embeddings, construct training examples (G_q, G_t) where half the time, G_q is a subgraph of G_t, and the other half, it is not

- Train on these examples by minimizing the following max-margin loss:
  - For positive examples: Minimize $E(G_q, G_t)$ when G_q is a subgraph of G_t
  - For negative examples: Minimize $\max(0, \alpha - E(G_q, G_t))$
    - Max-margin loss prevents the model from learning the degenerate strategy of moving embeddings further and further apart forever

Training Example Construction

- Need to generate training queries G_Q and targets G_T from the dataset G
- Get G_T by choosing a random anchor v and taking all nodes in G within distance K from v to be in G_T
- Positive examples: Sample induced subgraph G_Q of G_T. Use BFS sampling:
  - Initialize $S = \{v\}$, $V = \emptyset$
  - Let $N(S)$ be all neighbors of nodes in S. At every step, sample 10% of the nodes in $N(S) \setminus V$, put them in S. Put the remaining nodes of $N(S)$ in V.
  - After K steps, take the subgraph of G induced by S anchored at q
- Negative examples (G_Q not subgraph of G_T): "corrupt" G_Q by adding/removing nodes/edges so it's no longer a subgraph.



Training Details

- How many training examples to sample?
  - At every iteration, we sample new training pairs
  - Benefit: Every iteration, the model sees different subgraph examples
  - Improves performance and avoids overfitting – since there are exponential number of possible subgraphs to sample from

- How deep is the BFS sampling?
  - A hyper-parameter that trades off runtime and performance
  - Usually use 3-5, depending on size of the dataset

Subgraph Predictions on New Graphs

- **Given**: query graph G_q anchored at node q, target graph G_t anchored at node t
- **Goal**: output whether the query is a node-anchored subgraph of the target
- **Procedure**:
  - If $E(G_q, G_t) < \epsilon$, predict "True"; else "False"
  - ϵ is a hyper-parameter
- To check if G_Q, is isomorphic to a subgraph of G_T, repeat this procedure for all $q \in G_Q$, $t \in G_T$. Here G_q is the neighborhood around node $q \in G_Q$.

Summary: Neural Subgraph Matching

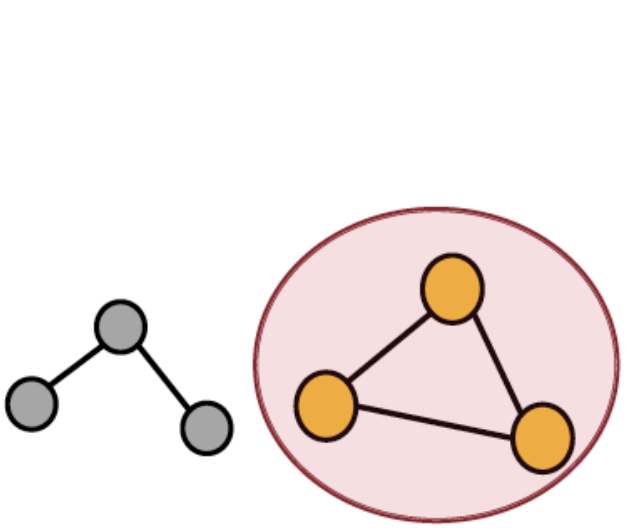
- Neural subgraph matching uses a machine learning-based approach to learn the NP-hard problem of subgraph isomorphism
  - Given query and target graph, it embeds both graphs into an order embedding space
  - Using these embeddings, it then computes $E(G_q, G_t)$ to determine whether query is a subgraph of the target

- Embedding graphs within an order embedding space allows subgraph isomorphism to be efficiently represented and tested by the relative positions of graph embeddings

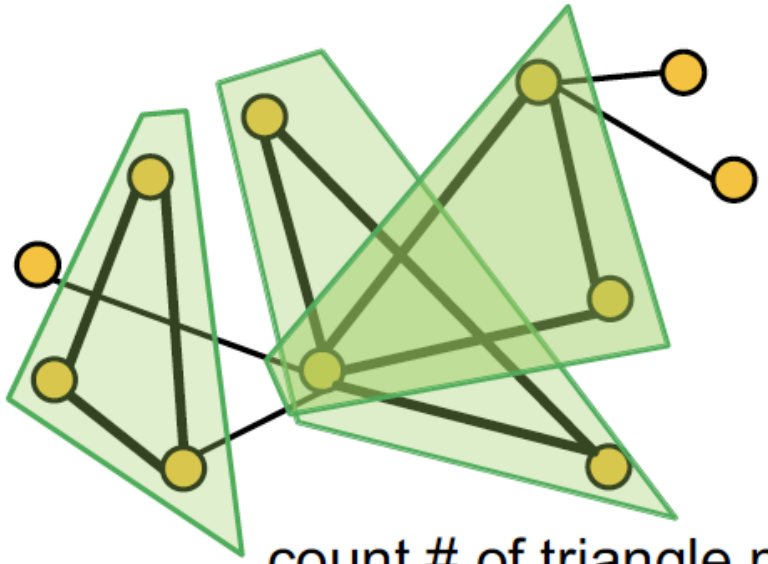
Finding Frequent Subgraphs

Intro: Finding Frequent Subgraphs

- Generally, finding the most frequent size-k motifs requires solving two challenges:
  - 1) Enumerating all size-k connected subgraphs
  - 2) Counting #(occurrences of each subgraph type)



Possible size-3 motifs



count # of triangle motifs

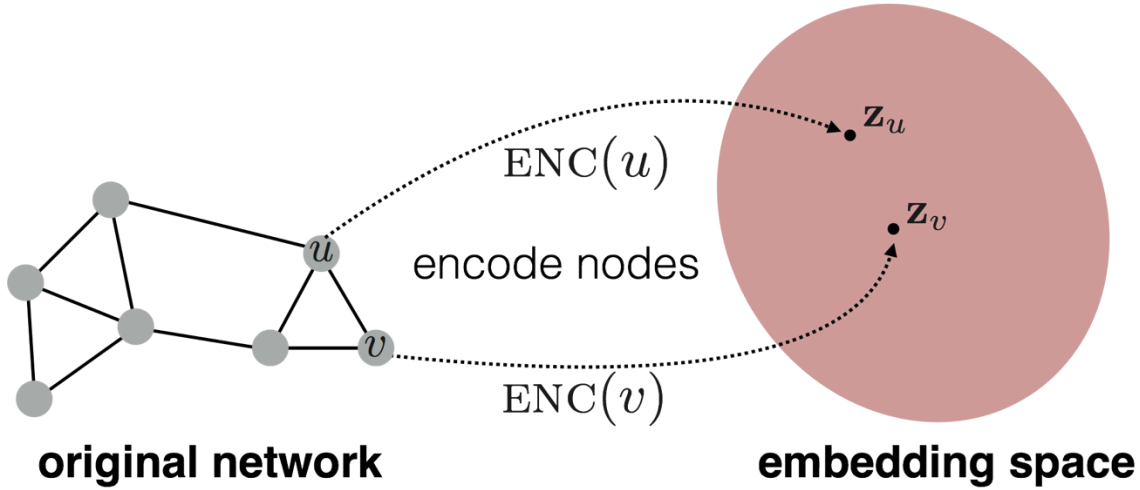
Why is It Hard?

- Just knowing if a certain subgraph exists in a graph, is a hard computational problem!
  - Subgraph isomorphism is NP-complete

- Computation time grows exponentially as the size of the subgraphs increases
  - Feasible motif size for traditional methods is relatively small (3 to 7)

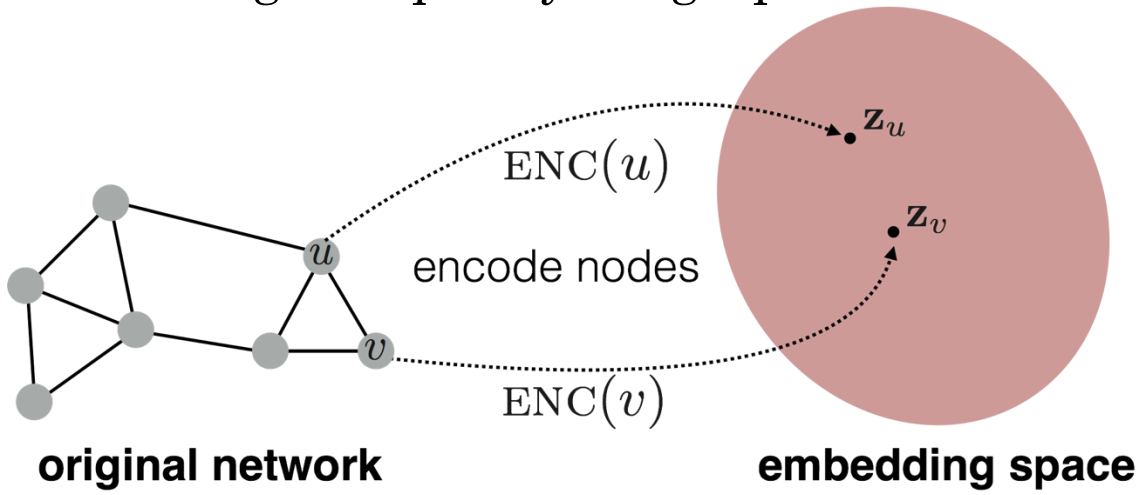
Solution with Representation Learning

- Finding frequent subgraph patterns is computationally hard
  - Combinatorial explosion of number of possible patterns
  - Counting subgraph frequency is NP-hard
- Representation learning can tackle these challenges:
  - Combinatorial explosion → organize the search space
  - Subgraph isomorphism → prediction using GNN



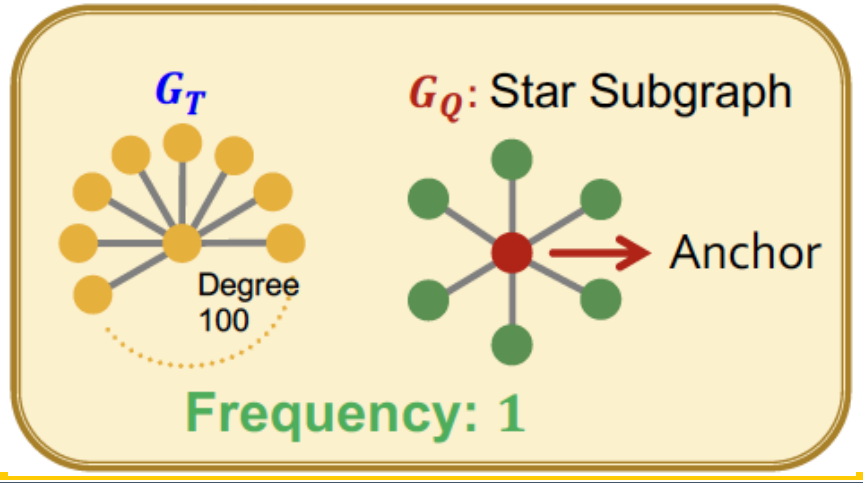
Solution with Representation Learning

- Representation learning can tackle these challenges:
  - 1) Counting #(occurrences of each subgraph type)
    - Solution: Use GNN to "predict" the frequency of the subgraph.
  - 2) Enumerating all size-k connected subgraphs
    - Solution: Don't enumerate subgraphs but construct a size-k subgraph incrementally
      - Note: We are only interested in high frequency subgraphs



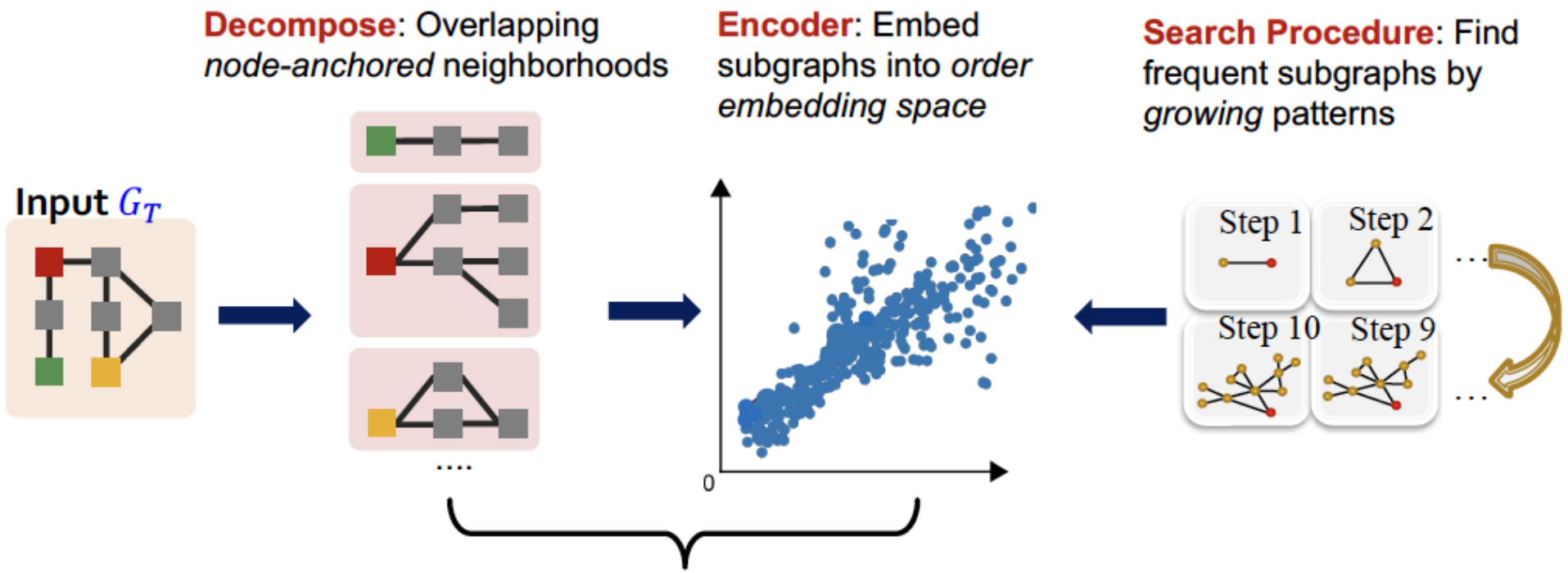
Problem Setup: Frequent Motif Finding

- Target graph (dataset) G_T, size parameter k
- Desired number of results r
- Goal: Identify, among all possible graphs of k nodes, the r graphs with the highest frequency in G_T.
- We use the node-level definition: The number of nodes u in G_T for which some subgraph of G_T is isomorphic to G_Q and the isomorphism maps u to v.



SPMiner: Overview

SPMiner: A neural model to identify frequent motifs

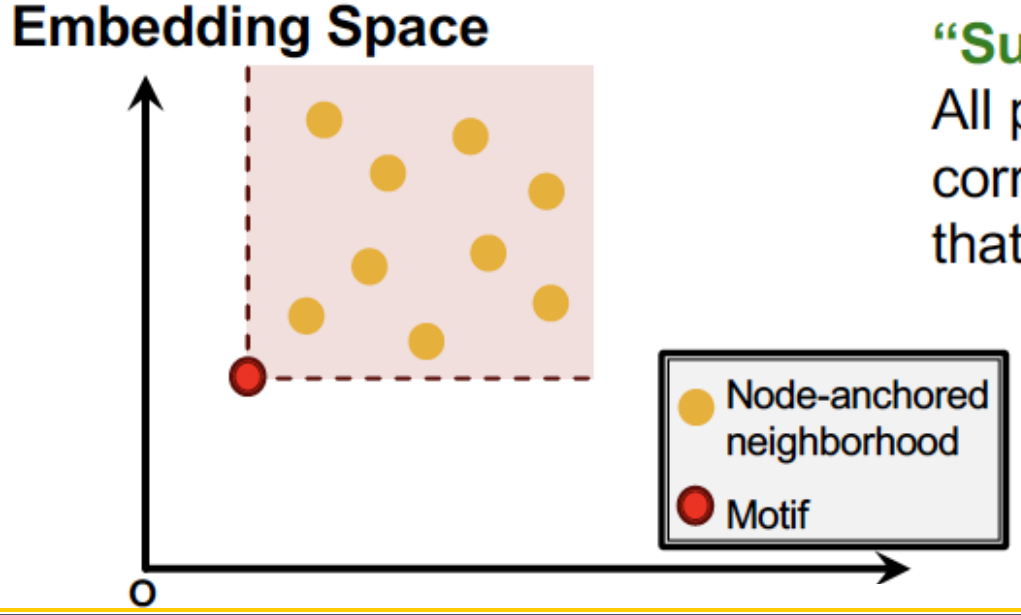


SPMiner: Key Idea

- Decompose input graph G_T into neighborhoods
- Embed neighborhoods into an order embedding space
- Key benefit of order embedding: We can quickly "predict" the frequency of a given subgraph G_Q

Motif Frequency Estimation

- Given: Set of subgraphs ("node-anchored neighborhoods") G_{N_i} of G_T (sampled randomly)
- Key idea: Estimate frequency of G_Q by counting the number of G_{N_i} such that their embeddings z_{N_i} satisfy $z_Q \leq z_{N_i}$
  - This is a consequence of the order embedding space property

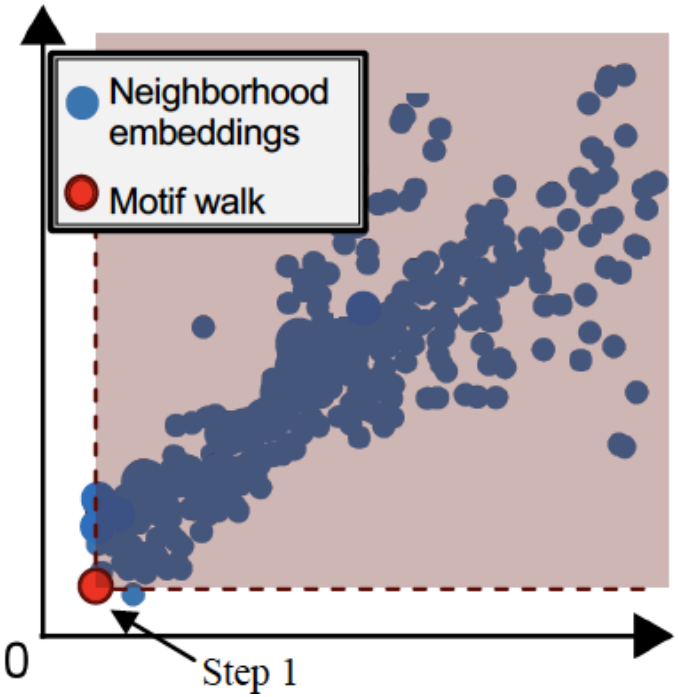


"Super-graph" region:
All points in the red shaded region correspond to neighborhoods in G_T that contain G_Q

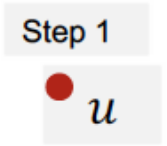
Benefit: Super-fast subgraph frequency counting!

SPMiner Search Procedure (1)

- Initial step: Start by randomly picking a starting node u in the target graph G_T. Set $S = \{u\}$.



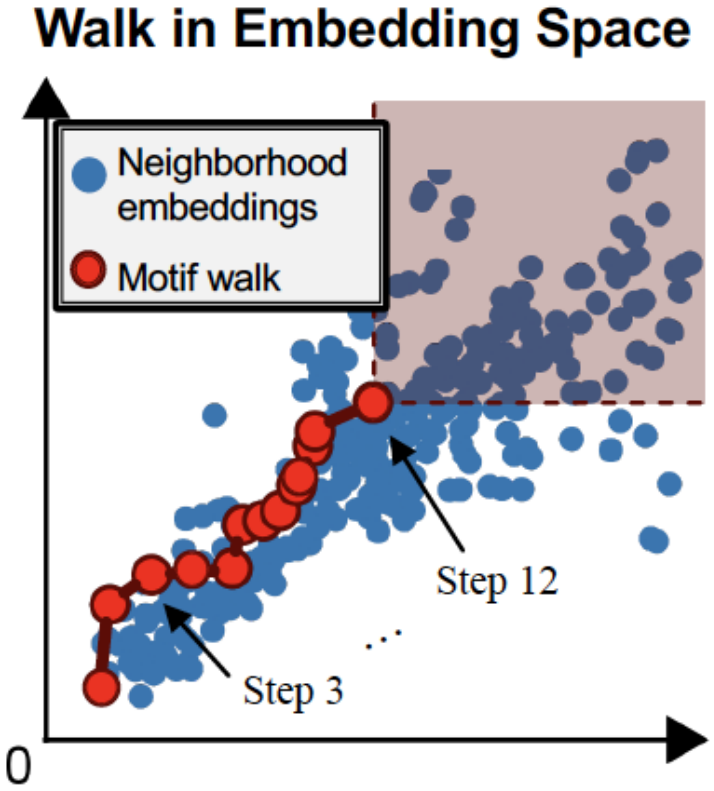
Each point in the shaded region represents a neighborhood in target graph that contains the motif pattern



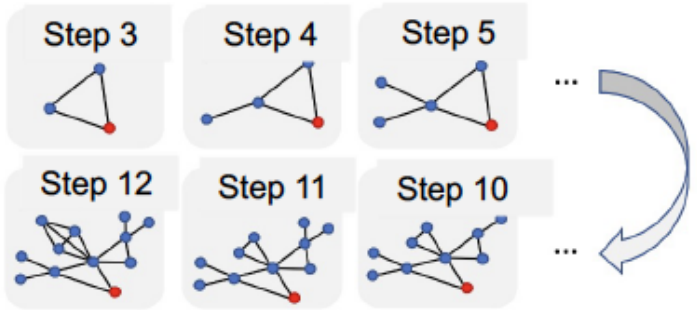
Initially, all neighborhoods contain the trivial subgraph

SPMiner Search Procedure (2)

- Iteratively: Grow a motif by iteratively choosing a neighbor in G_T of a node in S and add that node to S. We grow the motif S to find larger frequent motifs!



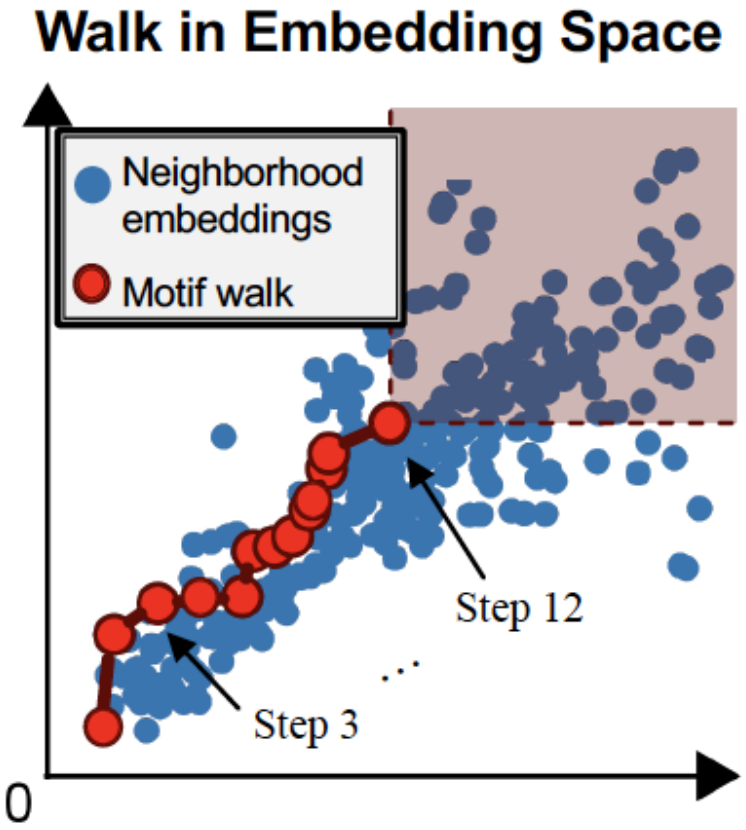
- Small motifs grow by adding neighbors
- Their embeddings correspond to red points on the left

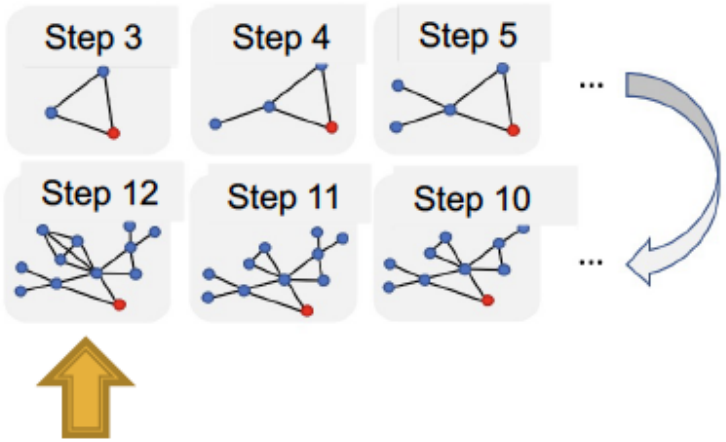


Goal: maximize number of neighborhoods in red shaded area after k step!

SPMiner Search Procedure (3)

- Termination: Upon reaching a desired motif size, take the subgraph of the target graph induced by S.

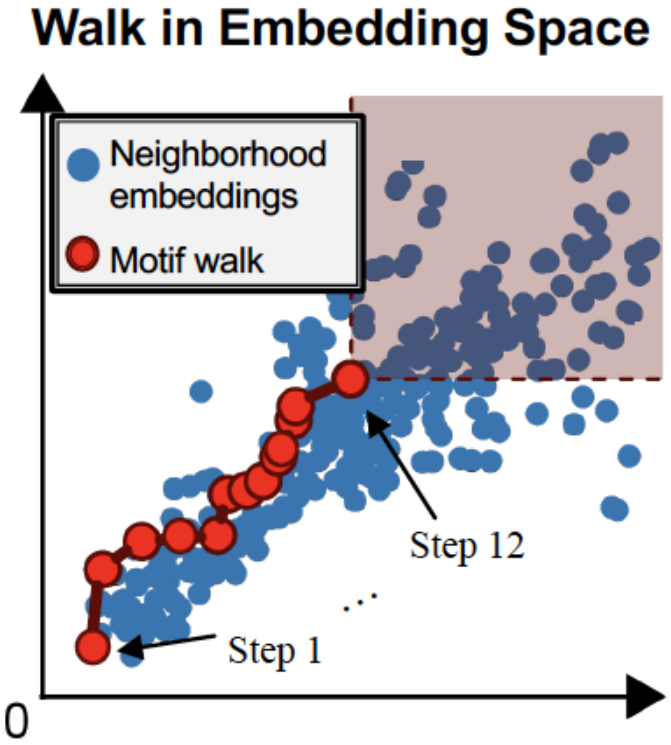




Identified frequent motif of size 12:
It has the largest number of blue points in super-graph region, among all embeddings of possible subgraphs of size 12

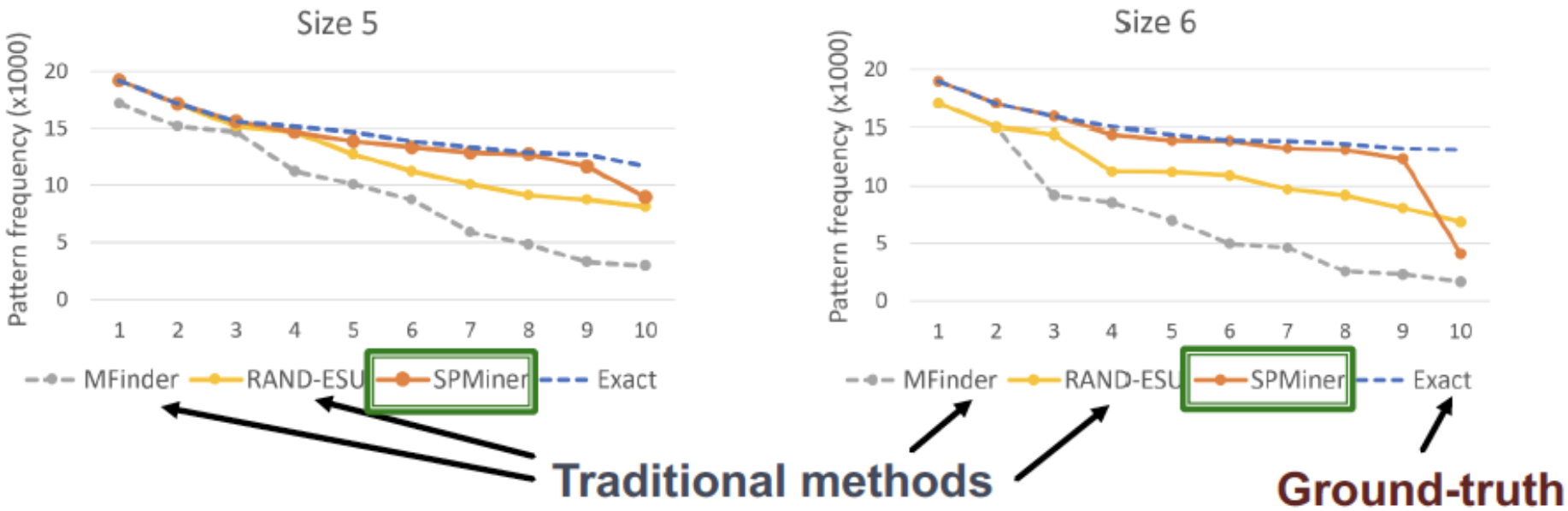
SPMiner Search Procedure (4)

- How to pick which node to add at each step?
- Def: Total violation of a subgraph G: the number of neighborhoods that do not contain G.
  - The number of neighborhoods G_{N_i} that do not satisfy $z_Q \preccurlyeq z_{N_i}$
  - Minimizing total violation = maximizing frequency
- Greedy strategy (heuristic): At every step, add the node that results in the smallest total violation



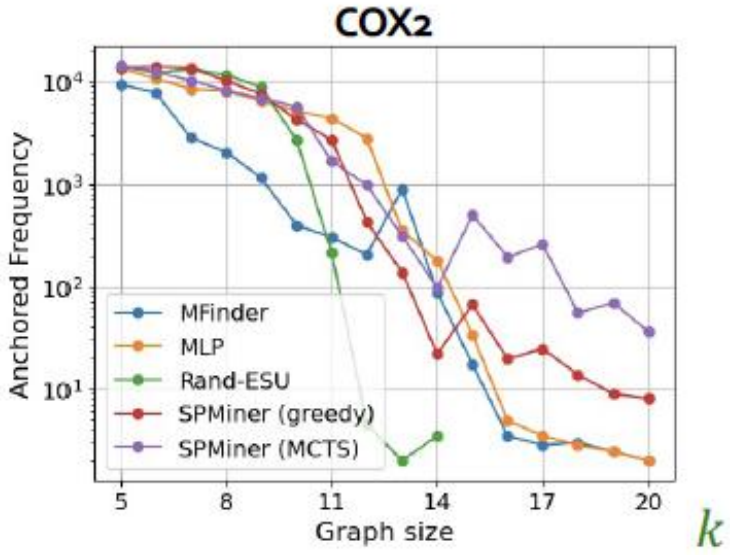
Results: Small Motifs

- Ground-truth: Find most frequent 10 motifs in dataset by brute-force exact enumeration (expensive)
- Question: Can the model identify frequent motifs?
- Result: The model identifies 9 and 8 of the top 10 motifs, respectively

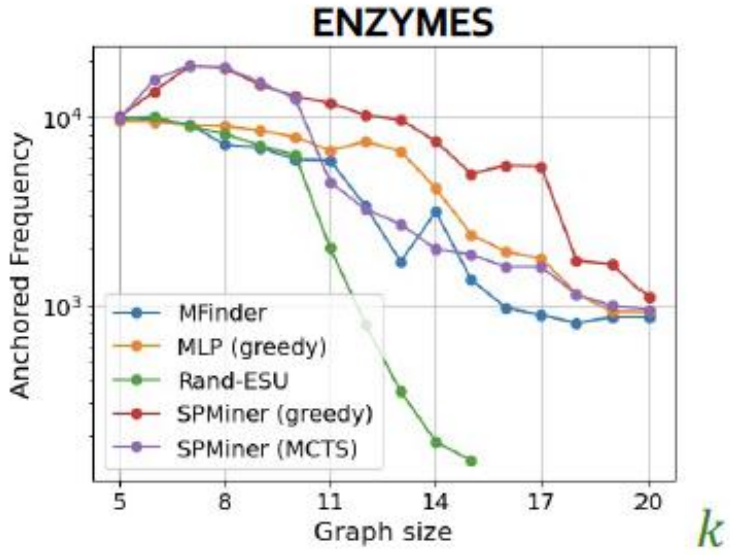


Experiments: Large Motifs

- Question: How do the frequencies of the identified motif compare?
- Result: SPMiner identifies motifs that appear 10-100x more frequently than the baselines



Molecule dataset



Protein dataset

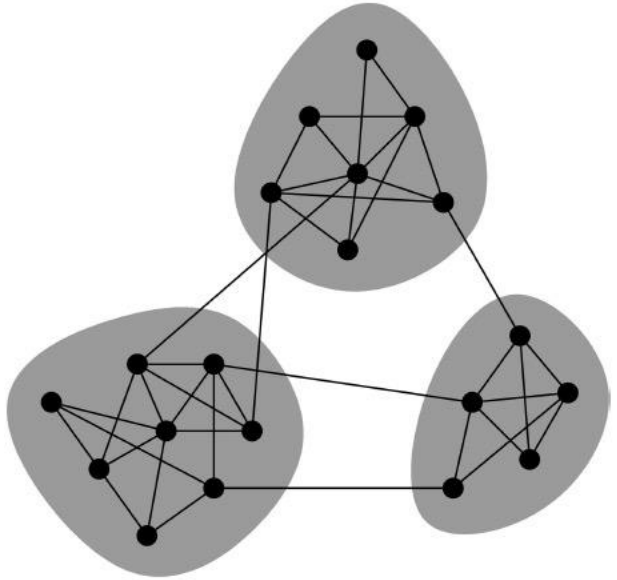
Summary

- Subgraphs and motifs are important concepts that provide insights into the structure of graphs. Their frequency can be used as features for nodes/graphs.
- We covered neural approaches to prediction subgraph isomorphism relationship.
- Order embeddings have desirable properties and can be used to encode subgraph relations
- Neural embedding-guided search in order embedding space can enable ML model to identify motifs much more frequent than existing methods

Community Detection

Networks and Communities

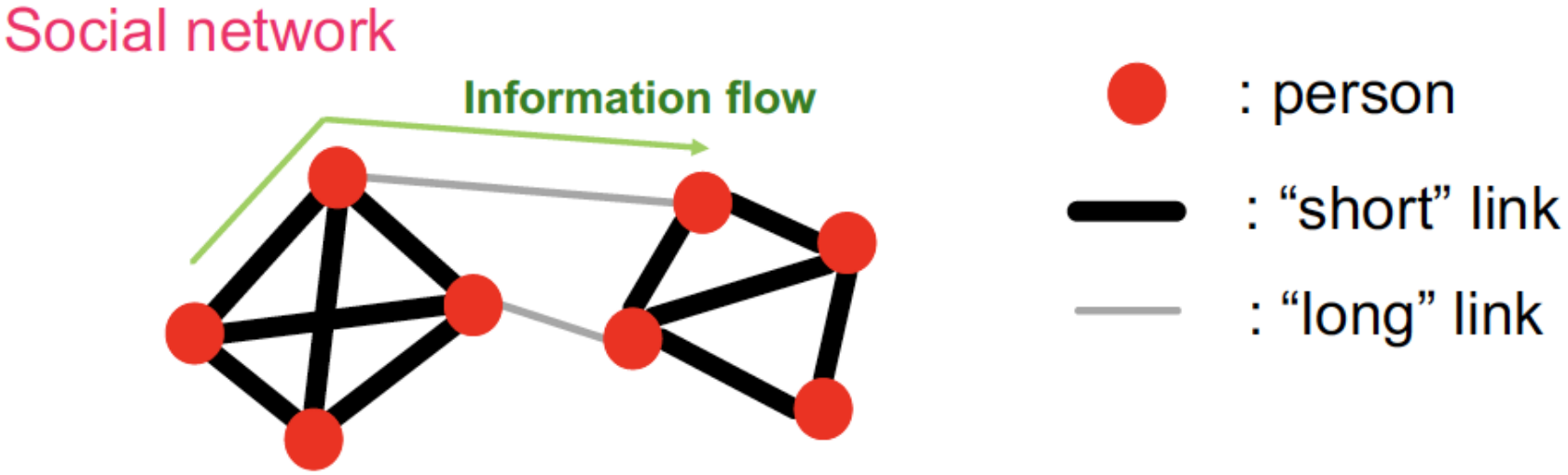
- We often think of networks "looking" like this:



- What led to such a conceptual picture?

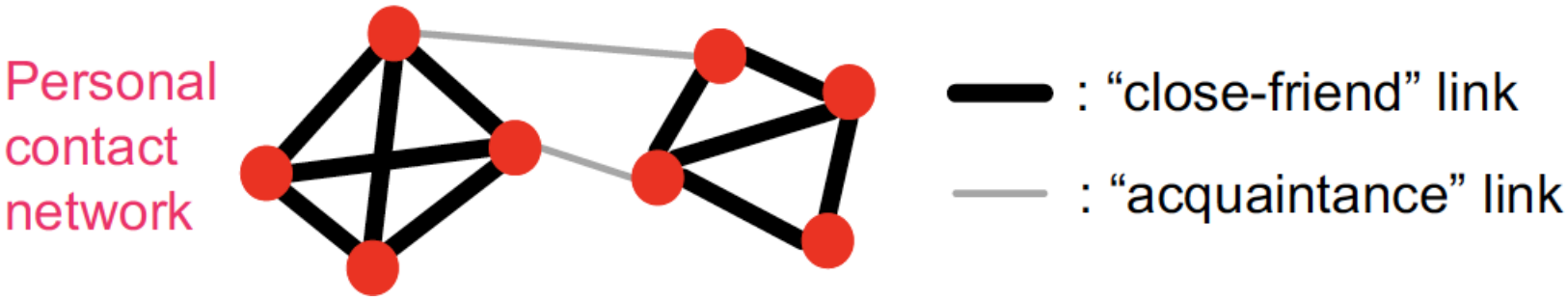
Networks: Flow of Information

- How does information flow through the network?
  - People are "embedded" in a social network.
  - There are different links ("short" vs. "long") in the network, through which information flows.



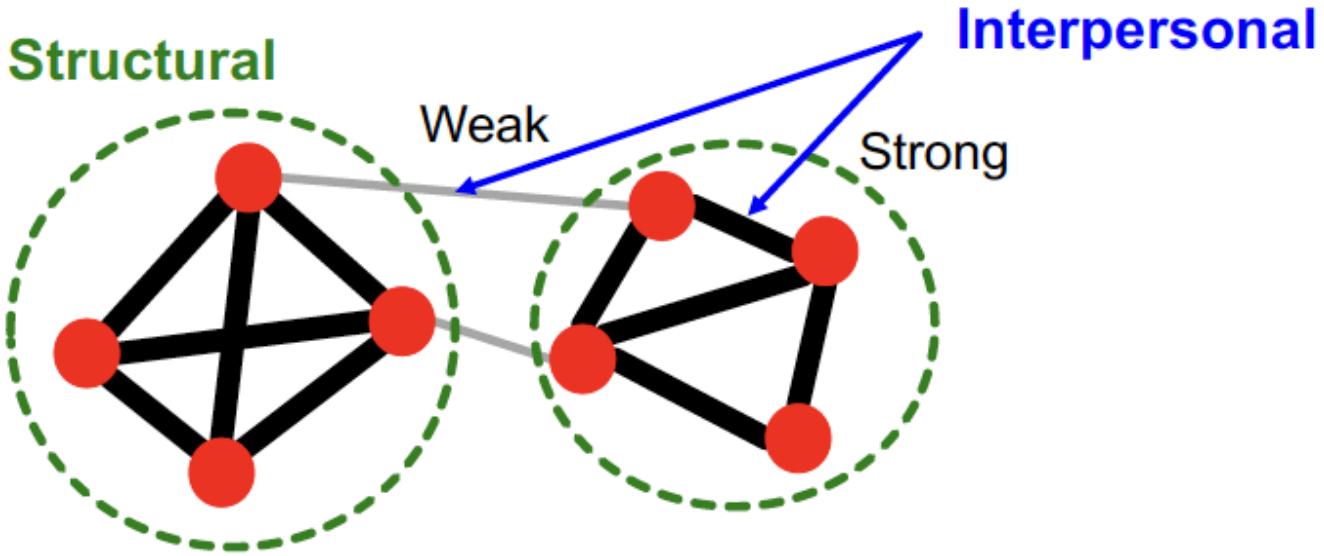
Flow of Job Information

- How do people find out about new jobs?
  - Mark Granovetter, part of his PhD in 1960s
  - People find the information through personal contacts
- But: Contacts were often acquaintances rather than close friends
  - This is surprising: One would expect your friends to help you out more than casual acquaintances
- Why is it that acquaintances are most helpful?



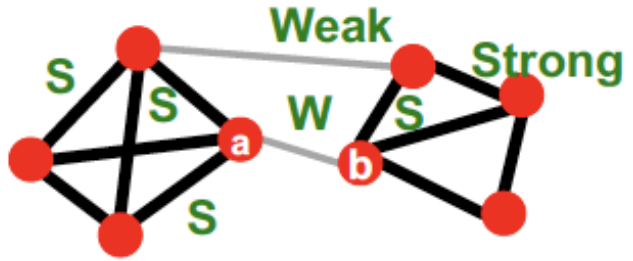
Granovetter's Answer

- Two perspectives on friendships:
  - Structural: Friendships span different parts of the network
  - Interpersonal: Friendship between two people is either strong or weak



Granovetter's Explanation

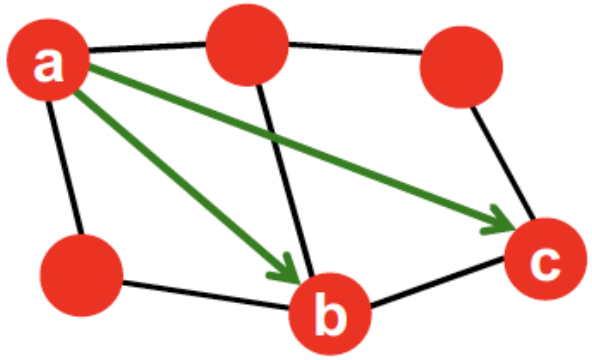
- Granovetter makes a connection between the social and structural role of an edge
- First point: Structure
  - Structurally embedded (tightly-connected) edges are also socially strong
  - Long-range edges spanning different parts of the network are socially weak
- Second point: Information
  - Long-range edges allow you to gather information from different parts of the network and get a job
  - Structurally embedded edges are heavily redundant in terms of information access



Triadic Closure

- How community (tightly-connected cluster of nodes) forms?

Which edge is more likely, a-b or a-c?



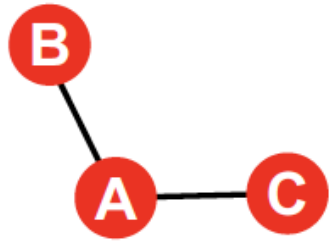
If two people in a network have a friend in common, then there is an increased likelihood they will become friends themselves.

Reasons for Triadic Closure

- Triadic closure = High clustering coefficient

Reasons for triadic closure:

- If B and C have a friend A in common, then:
  - B is more likely to meet C
    - (since they both spend time with A)
  - B and C trust each other
    - (since they have a friend in common)
  - A has incentive to bring B and C together
    - (since it is hard for A to maintain two disjoint relationships)



- Empirical study by Bearman and Moody:
  - Teenage girls with low clustering coefficient are more likely to contemplate suicide

Edge Strength in Real Data

- For many years Granovetter's theory was not tested

- But today we have large who-talks-to-whom graphs:
  - Email, Messenger, Cell phones, Facebook

- Onnela et al. 2007:
  - Cell-phone network of 20% of EU country's population
  - Edge weight: # phone calls

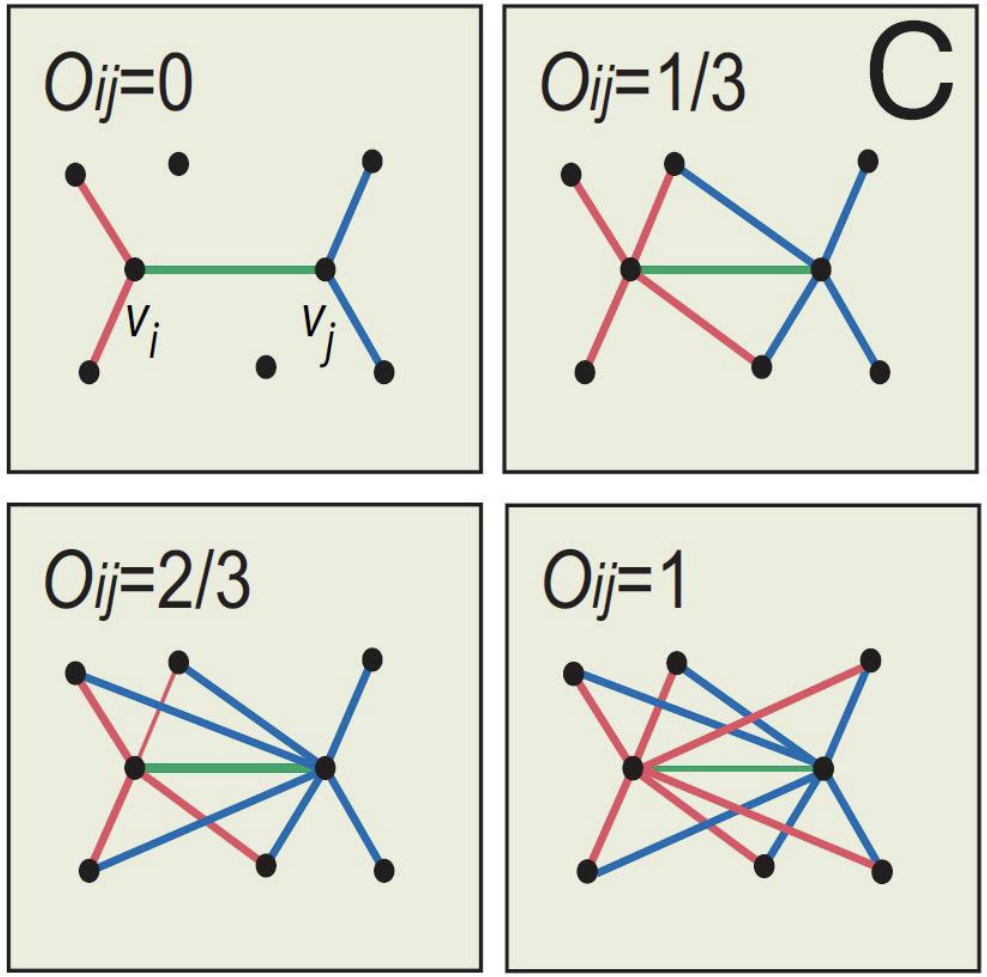
Edge Overlap

- Edge overlap:

$$O_{ij} = \frac{\left|\left(N(i) \cap N(j)\right) - \{i, j\}\right|}{\left|\left(N(i) \cup N(j)\right) - \{i, j\}\right|}$$

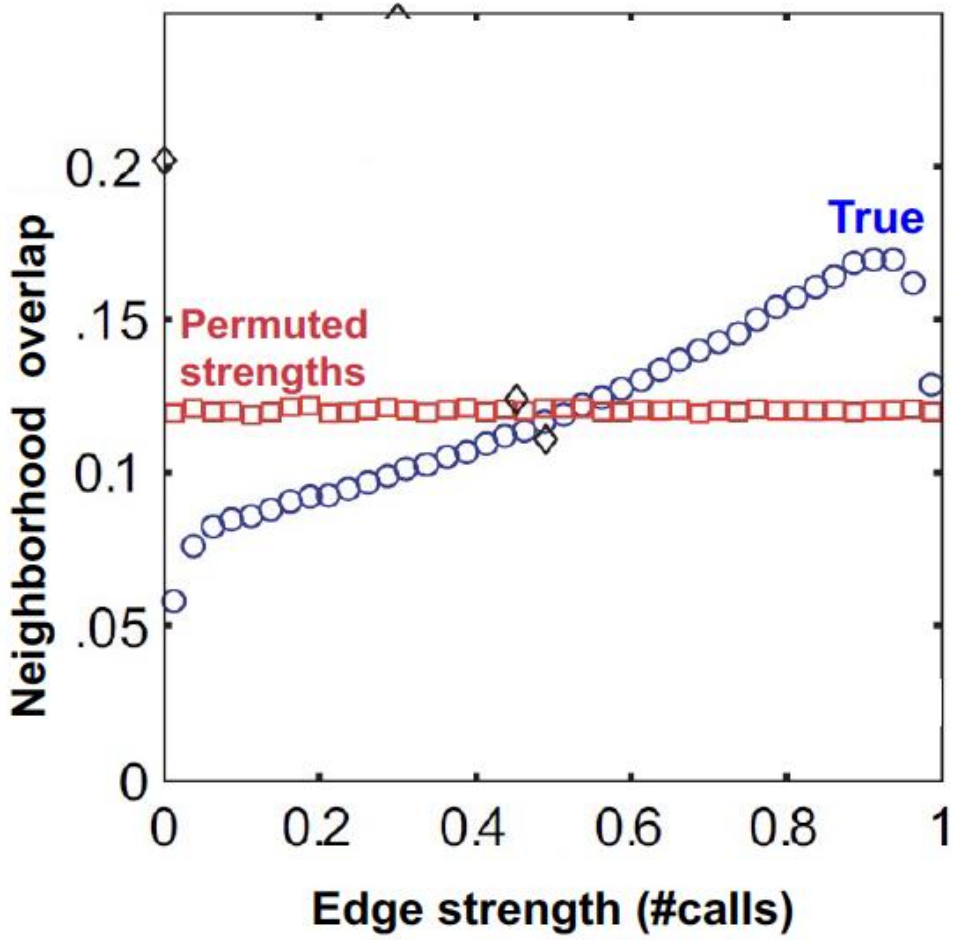
  - $N(i)$ … the set of neighbors of node i

- Note: Overlap = **0** when an edge is a "local bridge"

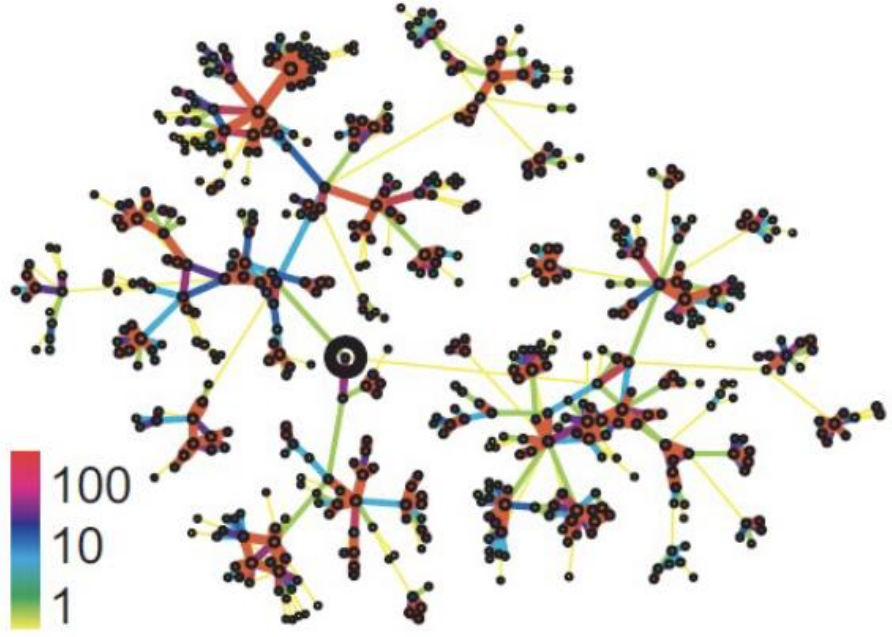


Phones: Edge Overlap vs Strength

- Cell-phone network
- Observation:
  - Highly used links have high overlap!
- Legend:
  - True: The data
  - Permuted strengths: Keep the network structure but randomly reassign edge strengths

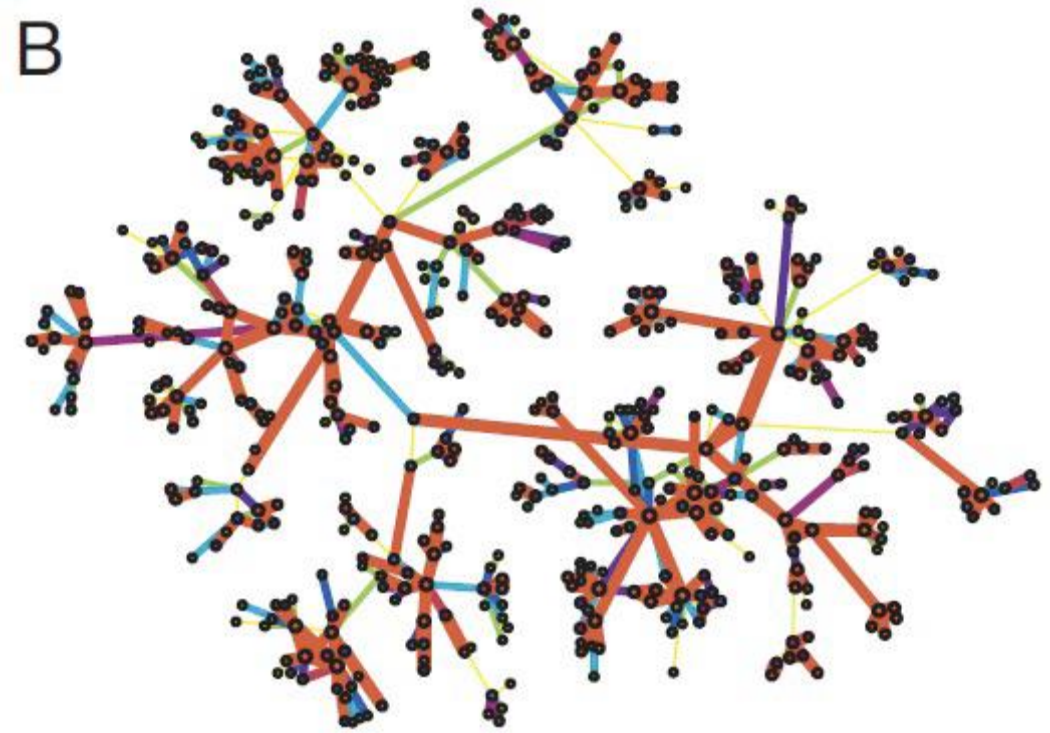


Real Networks, Real Edge Strengths



- Real edge strengths in mobile call graph
  - Strong ties are more embedded (have higher overlap)

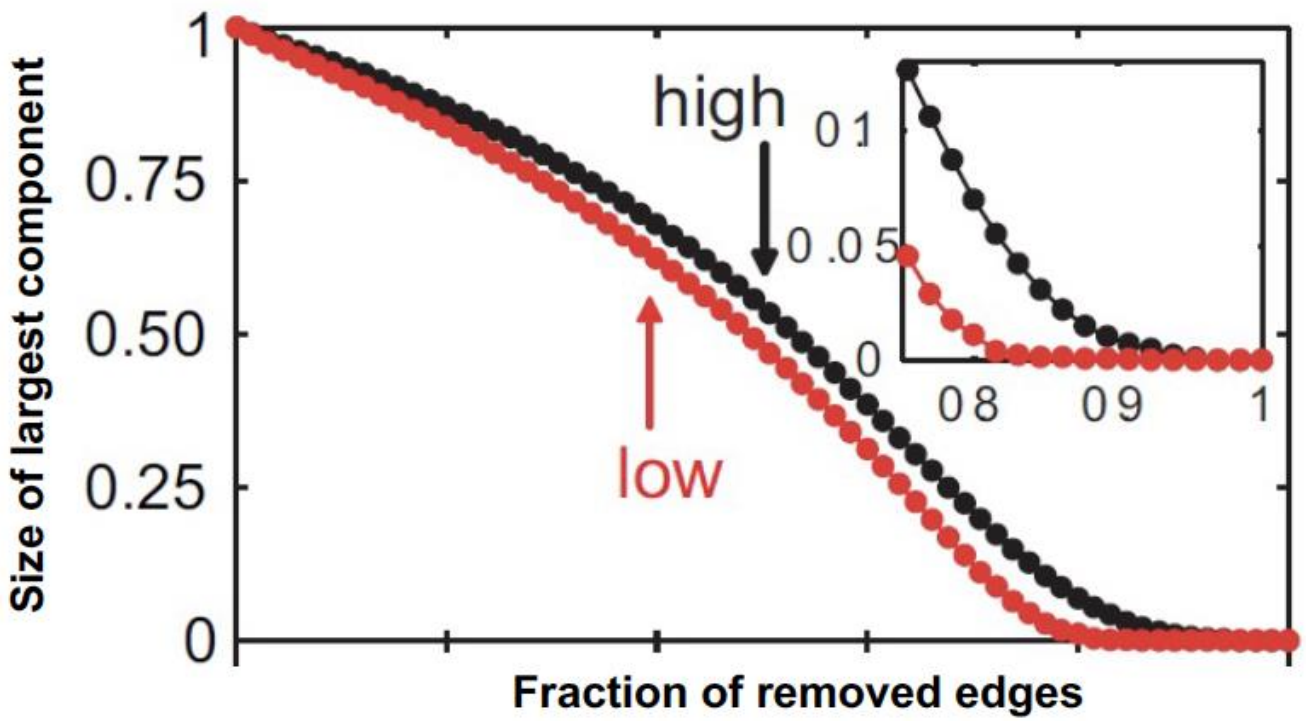
Real Net, Permuted Tie Strengths



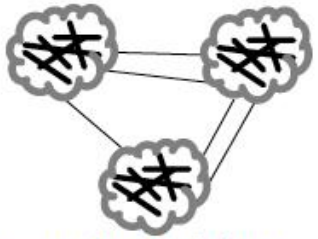
- Same network, same set of edge strengths but now strengths are randomly shuffled

Edge Removal by Strength

- Removing edges based on strength (#calls)
  - Low to high
  - High to low



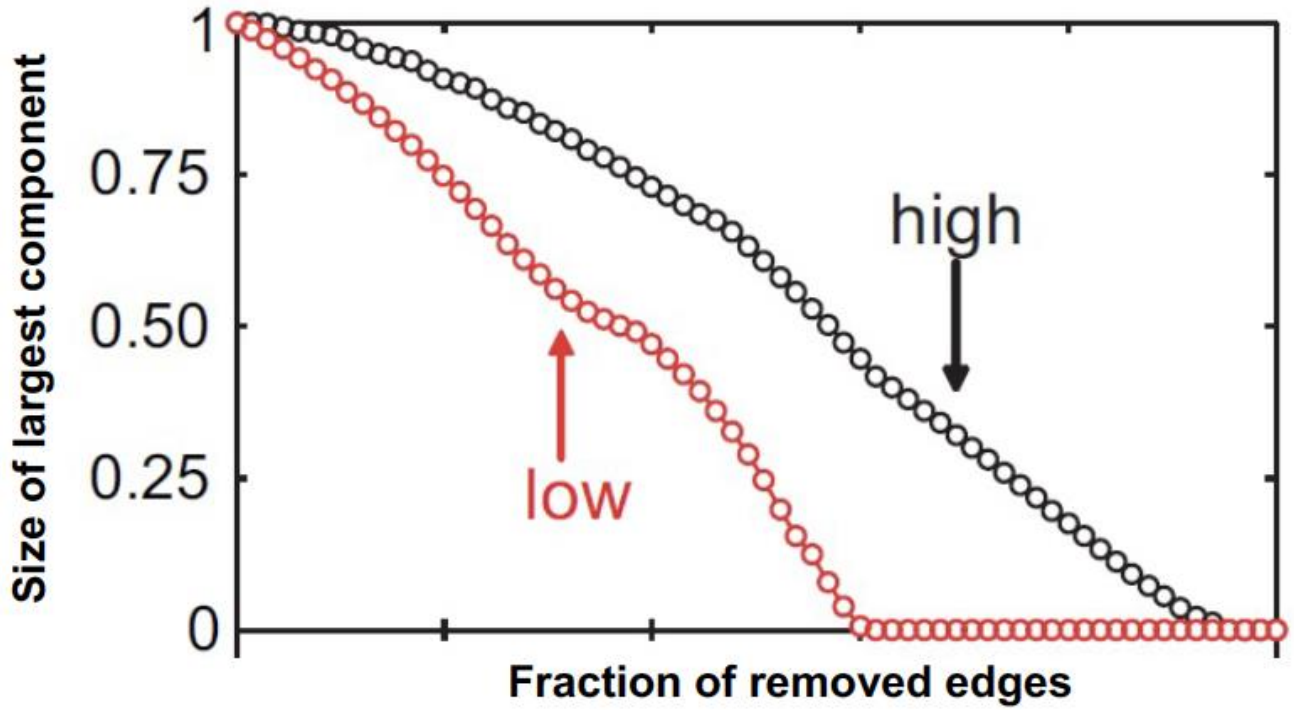
Low disconnects the network sooner



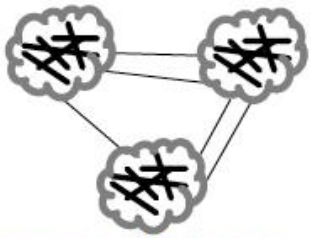
Conceptual picture of network structure

Edge Removal by Overlap

- Removing edges based on edge overlap
  - Low to high
  - High to low



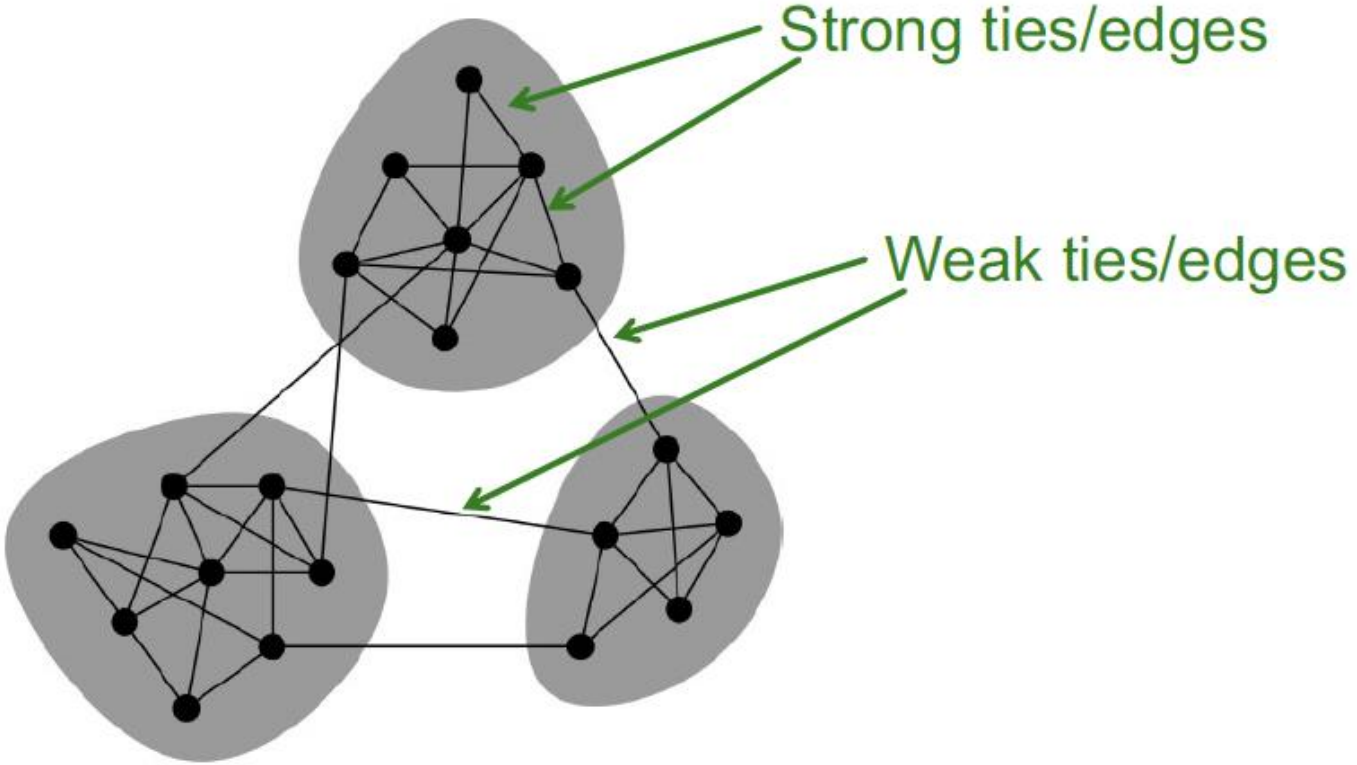
Low disconnects the network sooner



Conceptual picture of network structure

Conceptual Picture of Networks

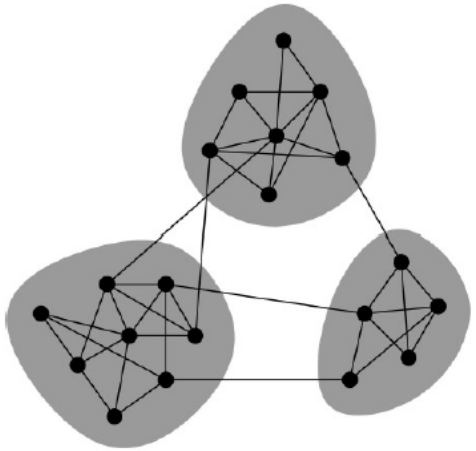
- Granovetter's theory leads to the following conceptual picture of networks



Network Communities

Network Communities

- Granovetter's theory suggests that networks are composed of tightly connected sets of nodes

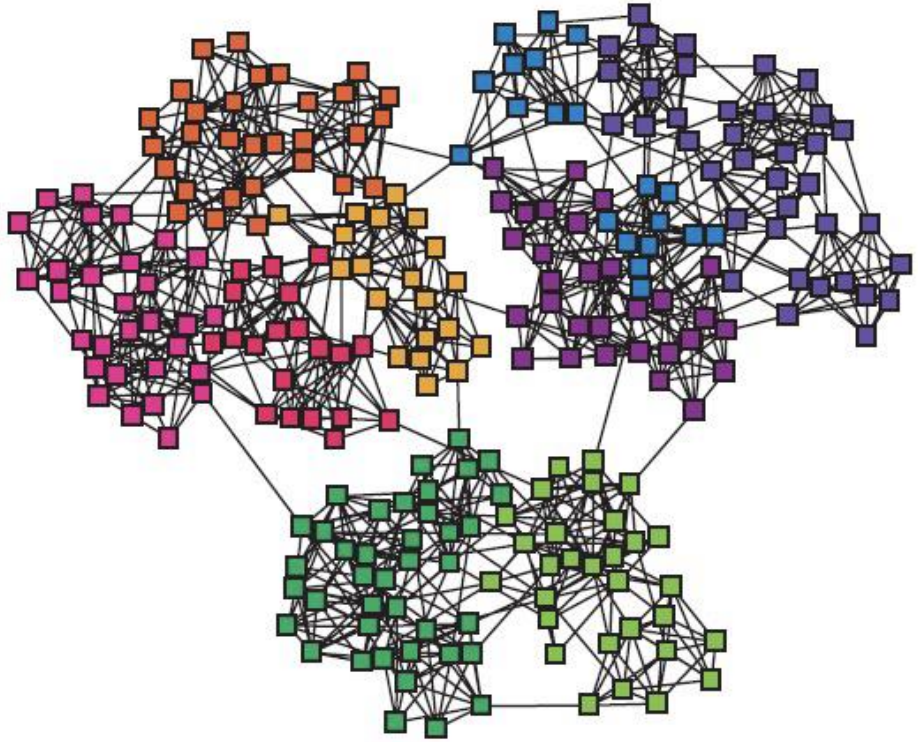


Communities, clusters, groups, modules

- Network communities:
  - Sets of nodes with lots of internal connections and few external ones (to the rest of the network).

Finding Network Communities

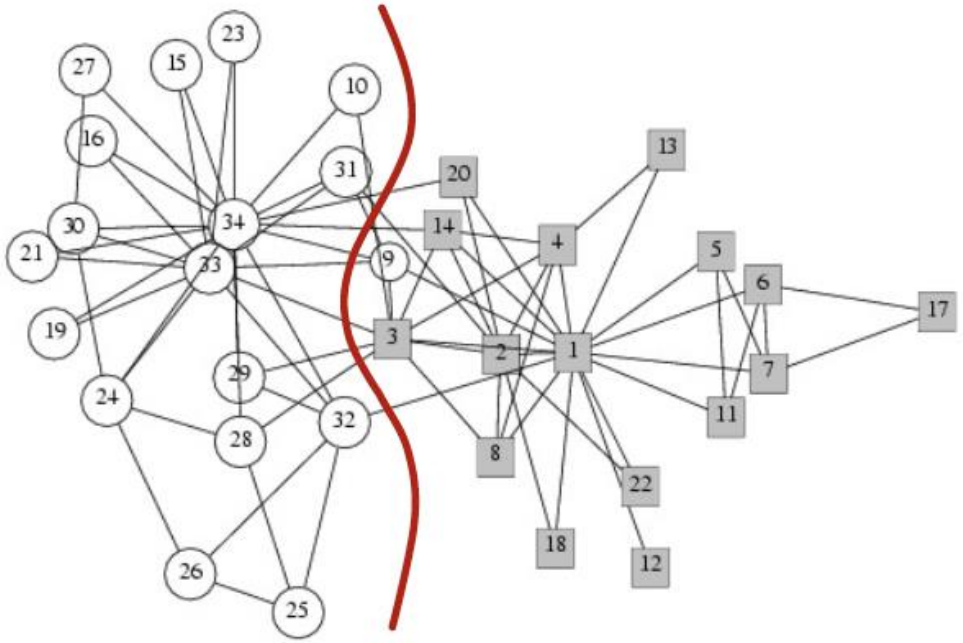
- How do we automatically find such densely connected groups of nodes?
- Ideally such automatically detected clusters would then correspond to real groups



- For example:

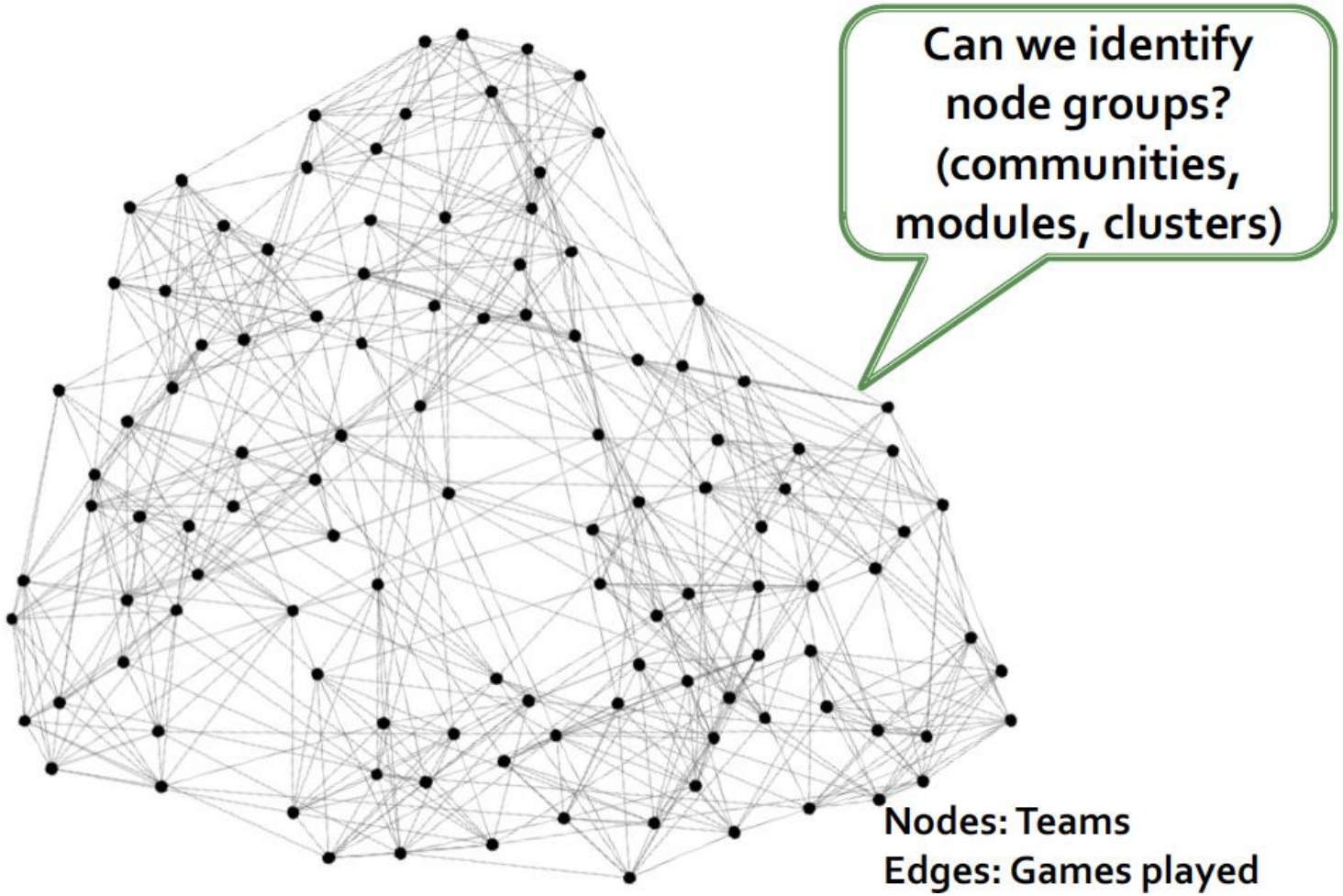
Communities, clusters, groups, modules

Social Network Data

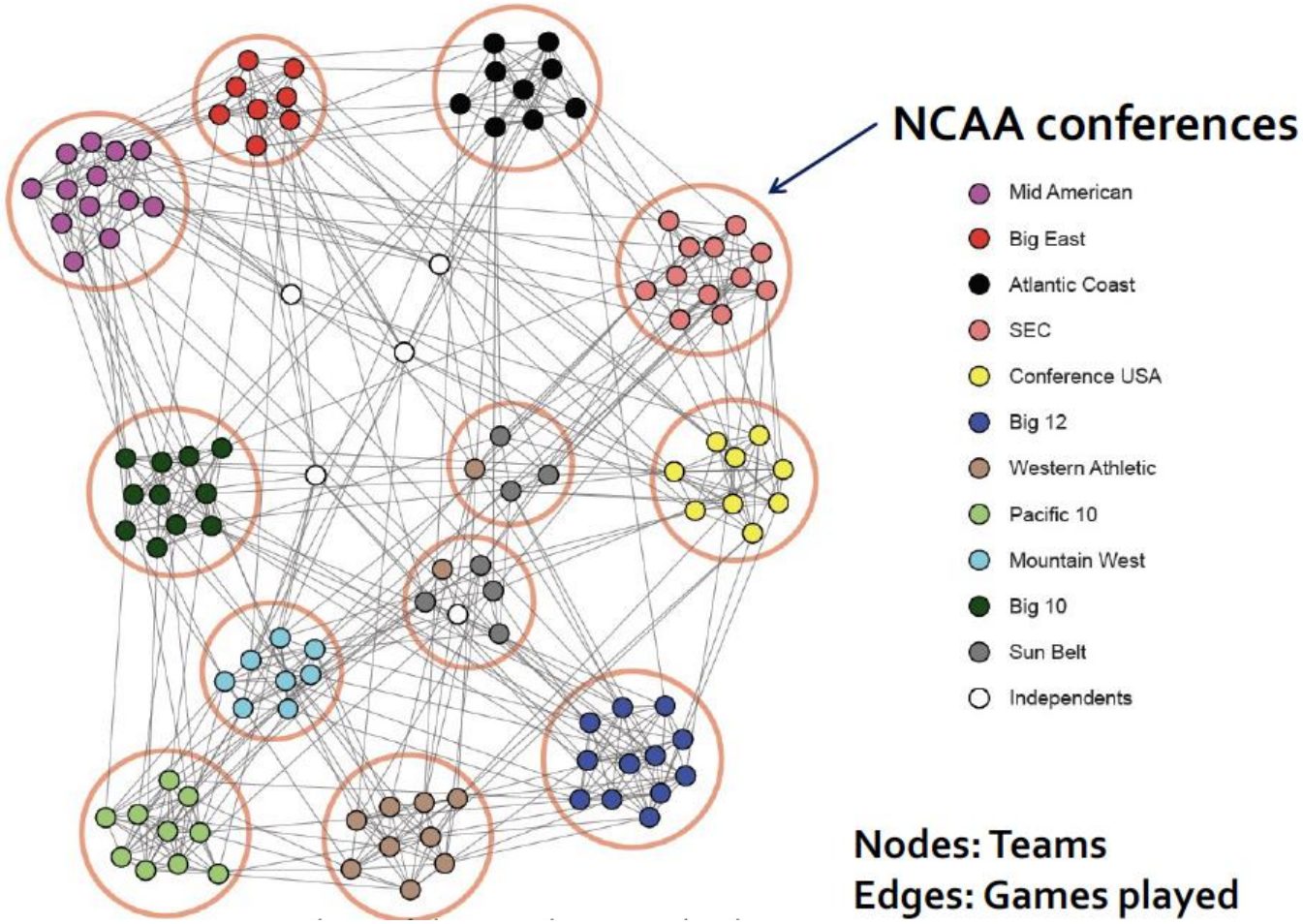


- Zachary's Karate club network:
  - Observed social ties & rivalries in a university karate club
  - During the study, conflicts led the group to split
  - Split could be explained by a minimum cut in the network

NCAA Football Network

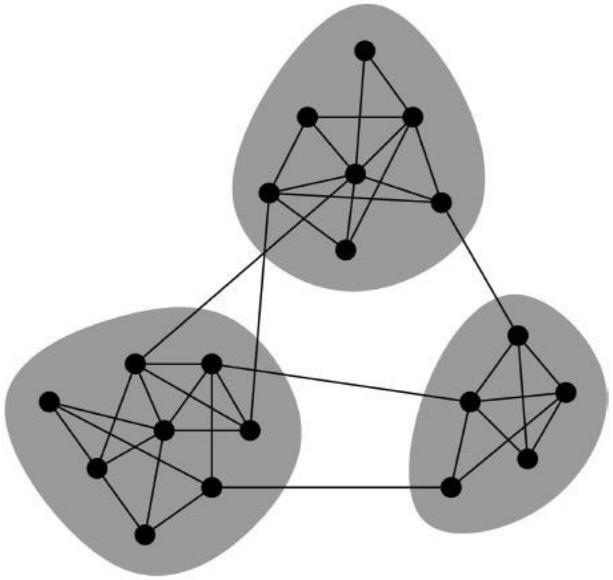


NCAA Football Network



Network Communities

- Communities: Sets of tightly connected nodes
- Define: Modularity Q
  - A measure of how well a network is partitioned into communities
  - Given a partitioning of the network into groups disjoint $s \in S$:

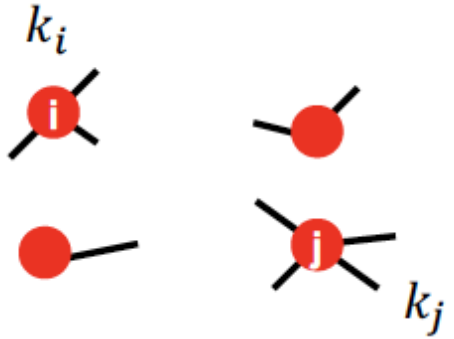


$$Q \propto \sum_{s \in S} [\ (\#\ \text{edges within group } s) - (\text{expected } \#\ \text{edges within group } s)\ ]$$

Need a null model

Null Model: Configuration Model

- Given real $\boldsymbol{G}$ on $\boldsymbol{n}$ nodes and $\boldsymbol{m}$ edges, construct rewired network $\boldsymbol{G}$'
  - Same degree distribution but uniformly random connections
  - Consider $\boldsymbol{G}$' as a multigraph (multiple edges exist between nodes)
  - The expected number of edges between nodes $\boldsymbol{i}$ and $\boldsymbol{j}$ of degrees $\boldsymbol{k_i}$ and $\boldsymbol{k_j}$ equals: $k_i \cdot \frac{k_j}{2m} = \frac{k_i k_j}{2m}$



  - There are $2m$ directed edges (counting $i \to j$ and $j \to i$) in total.
  - For each of k_i out-going edges from node i, the chance of it landing to node j is $\frac{k_j}{2m}$, hence $\frac{k_i k_j}{2m}$.

Null Model: Configuration Model

- The expected number of edges between nodes i and j of degrees k_i and k_j equals: $k_i \cdot \frac{k_j}{2m} = \frac{k_i k_j}{2m}$
  - The expected number of edges in (multigraph) G':
    - $= \frac{1}{2}\sum_{i \in N}\sum_{j \in N}\frac{k_i k_j}{2m} = \frac{1}{2}\cdot\frac{1}{2m}\sum_{i \in N} k_i \sum_{j \in N} k_j$
    - $= \frac{1}{4m} 2m \cdot 2m = m$
- Under null model, both the degree distribution and the total number of edges are preserved.

Notice: This model applies to both weighted and unweighted networks. For weighted networks we use the weighted degree (sum of the edge weights).

Modularity

- Modularity of partitioning S of graph G:
  - **Q ∝ $\sum_{s \in S}$ [(# edges within group *s*) – (expected # edges within group *s*)]**
  - $Q(G,S) = \underbrace{\frac{1}{2m}} \sum_{s \in S} \sum_{i \in s} \sum_{j \in s} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$

Normalizing const.: $-1 \leq Q \leq 1$

A_{ij} = 1 if i→j,
0 otherwise
(if G is weighted then A_{ij} is the edge weight)

- Modularity values take range [−1,1]
  - It is positive if the number of edges within groups exceeds the expected number
  - Q greater than 0.3-0.7 means significant community structure
  - Notice Modularity applies to weighted and unweighted networks.

Recap: Modularity

For each group s

$$Q(G,S) = \frac{1}{2m}\sum_{s\in S}\sum_{i\in s}\sum_{j\in s}\left(A_{ij} - \frac{k_i k_j}{2m}\right)$$

Equivalently modularity can be written as:

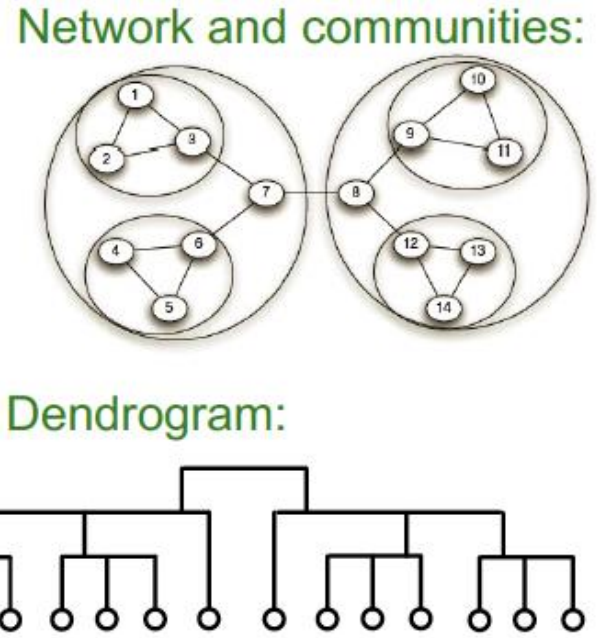
$$Q = \frac{1}{2m}\sum_{ij}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j)$$

- A_{ij} represents the edge weight between nodes i and j;
- k_i and k_j are the sum of the weights of the edges attached to nodes i and j, respectively;
- $2m$ is the sum of all of the edge weights in the graph;
- c_i and c_j are the communities of the nodes; and
- δ is an indicator function $\delta(c_i, c_j) = 1 \; if \; c_i = c_j \; else \; 0$

Idea: We can identify communities by maximizing modularity

Louvain Algorithm

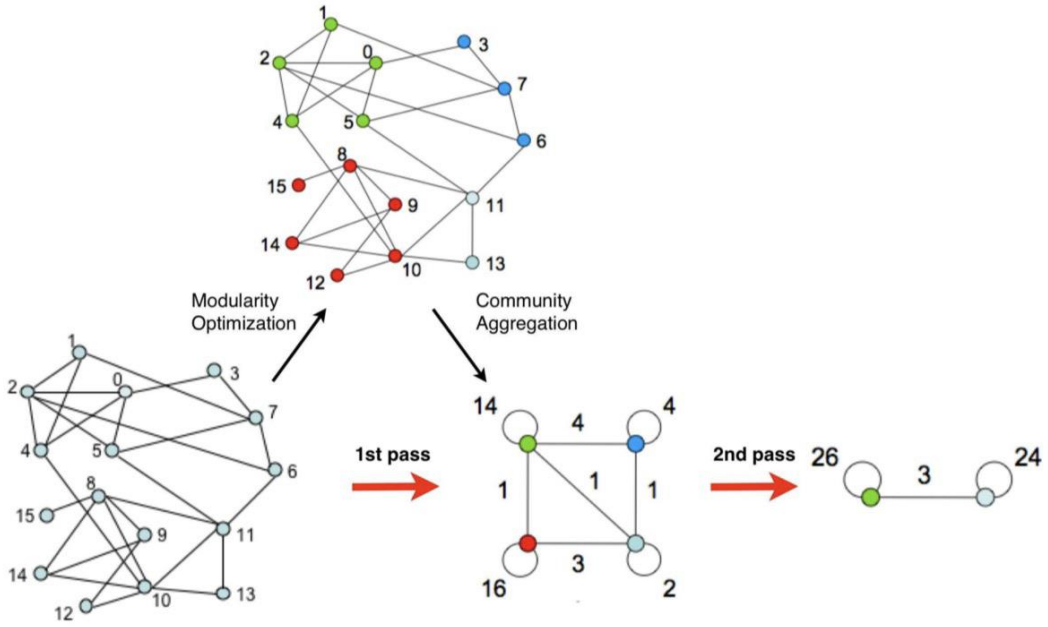
- Greedy algorithm for community detection
  - $O(n \log n)$ run time
- Supports weighted graphs
- Provides hierarchical communities
- Widely utilized to study large networks because:
  - Fast
  - Rapid convergence
  - High modularity output (i.e., "better communities")



Louvain Algorithm: At High Level

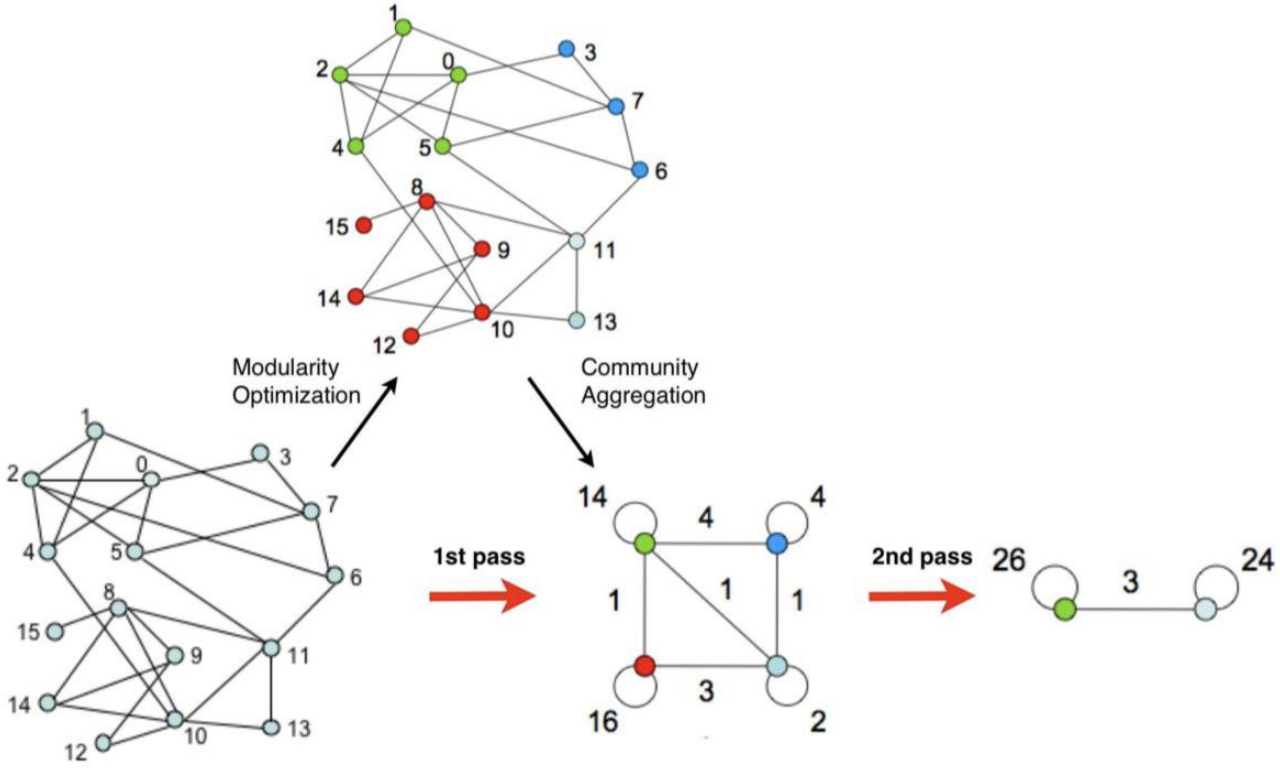
- Louvain algorithm greedily maximizes modularity
- Each pass is made of 2 phases:
  - Phase 1: Modularity is optimized by allowing only local changes to node-communities memberships
  - Phase 2: The identified communities are aggregated into super-nodes to build a new network
  - Go to Phase 1

The passes are repeated **iteratively** until no increase of modularity is possible.



Louvain Algorithm: At High Level

- Louvain algorithm considers graphs as weighted
  - The original graph can be unweighted (i.e., edge weights are all 1)
  - As the communities get identified and aggregated into super-nodes, weighted graphs are created (weights count the number of edges in the original graph)
  - Weighted version of Modularity is applied



Louvain: 1st phase (Partitioning)

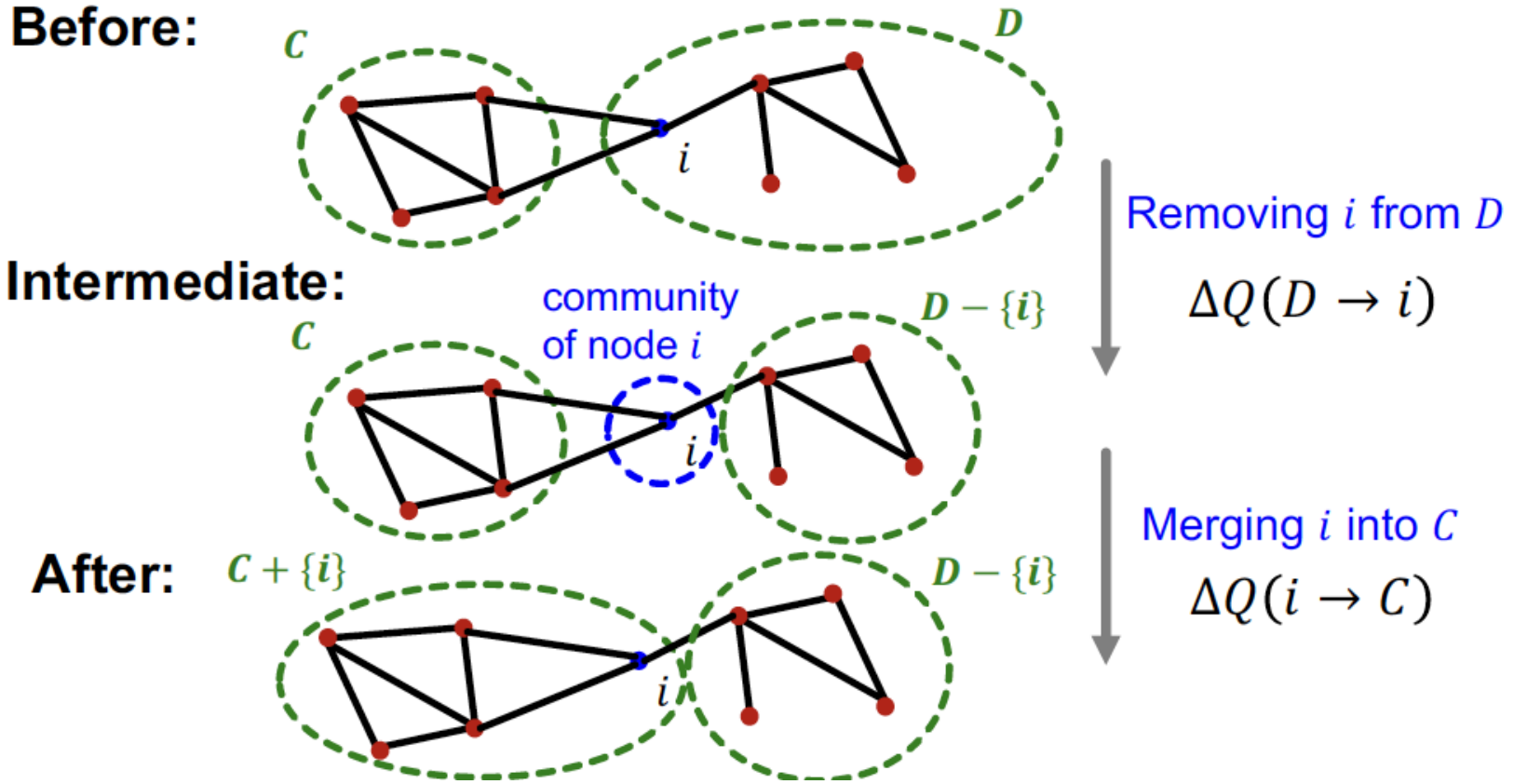
- Put each node in a graph into a distinct community (one node per community)
- For each node i, the algorithm performs two calculations:
  - Compute the modularity delta (ΔQ) when putting node i into the community of some neighbor j
  - Move i to a community of node j that yields the largest gain in ΔQ
- Phase 1 runs until no movement yields a gain

This first phase stops when a local maxima of the modularity is attained, i.e., when no individual node move can improve the modularity. Note that the output of the algorithm depends on the order in which the nodes are considered. Research indicates that the ordering of the nodes does not have a significant influence on the overall modularity that is obtained.

Louvain: Modularity Gain

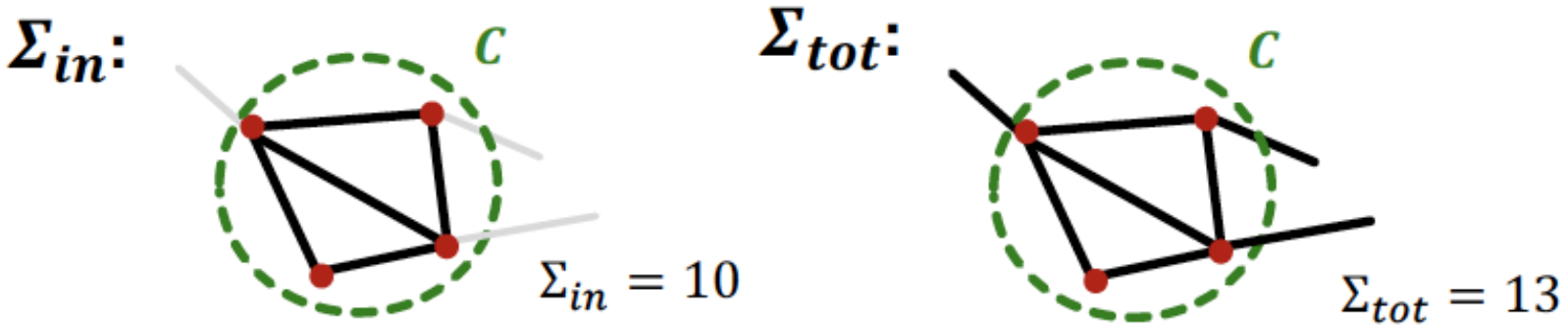
What is ΔQ if node i moves from community D to C?

$$\Delta Q(D \to i \to C) = \Delta Q(D \to i) + \Delta Q(i \to C)$$



Deriving $\Delta Q(i \to C)$

- Let's derive $\boldsymbol{\Delta Q(i \to C)}$
- First, we derive modularity within $\boldsymbol{C}$, i.e., $Q(C)$.
- Define:
  - $\sum_{in} \equiv \sum_{i,j \in C} A_{ij}$ ... sum of link weights between nodes in C
  - $\sum_{tot} \equiv \sum_{i \in C} k_i$ .... sum of all link weights of nodes in C



Deriving $\Delta Q(i \to C)$

- Let's derive $\Delta Q(i \to C)$
- First, we derive modularity within $\boldsymbol{C}$, i.e., $Q(C)$.
- Define:
  - $\sum_{in} \equiv \sum_{i,j \in C} A_{ij}$ ... sum of link weights between nodes in C
  - $\sum_{tot} \equiv \sum_{i \in C} k_i$ .... sum of all link weights of nodes in C
- Then, we have:

$$Q(C) \equiv \frac{1}{2m} \sum_{i,j \in C} \left[A_{ij} - \frac{k_i k_j}{2m} \right] = \frac{\sum_{i,j \in C} A_{ij}}{2m} - \frac{\left(\sum_{i \in C} k_i\right)\left(\sum_{j \in C} k_j\right)}{(2m)^2}$$

$$= \frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2$$

Links within the community

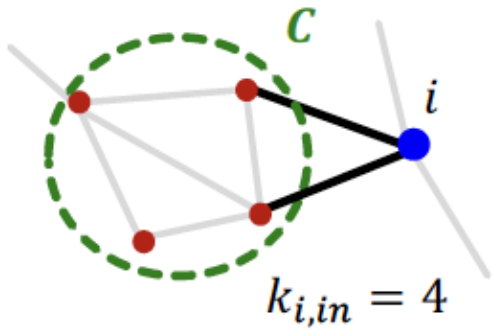
Total links

$\boldsymbol{Q(C)}$ is large when most of the total links are within community links

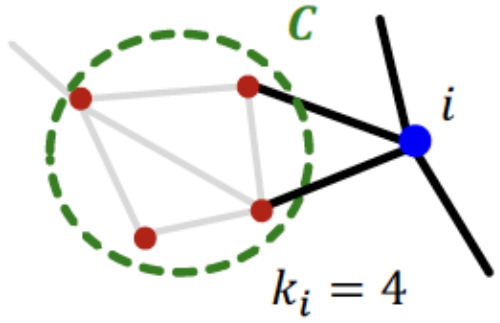
Deriving $\Delta Q(i \to C)$

- Further define:
  - $k_{i,in} \equiv \sum_{j \in C} A_{ij} + \sum_{j \in C} A_{ji}$ … sum of link weights connecting node i and C
    - (note that each edge gets counted twice, see formula)
  - $\boldsymbol{k_i}$… sum of all link weights (i.e., degree) of node i

$\boldsymbol{k_{i,in}}$:

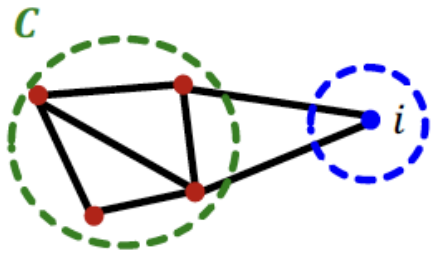


$\boldsymbol{k_i}$:



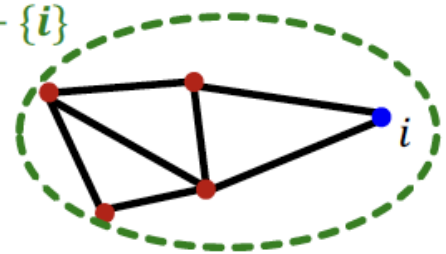
Deriving $\Delta Q(i \to C)$

Before merging



Isolated community of node i

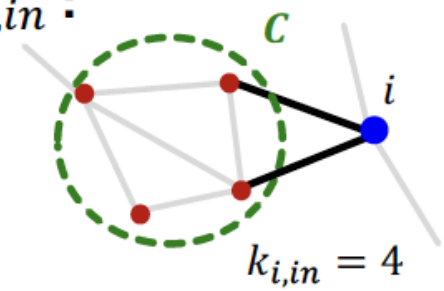
After merging



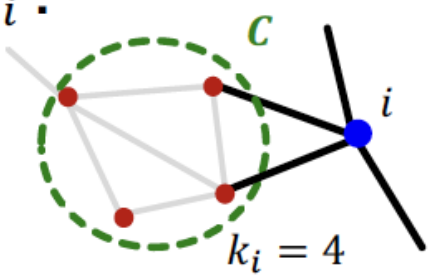
$$Q_{\text{before}} = Q(C) + Q(\{i\}) = \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2\right] + \left[0 - \left(\frac{k_i}{2m}\right)^2\right]$$

$$Q_{\text{after}} = Q(C + \{i\}) = \frac{\text{"}\Sigma_{in}\text{" of } C+\{i\}}{\Sigma_{in} + k_{i,in}} - \left(\frac{\text{"}\Sigma_{tot}\text{" of } C+\{i\}}{\Sigma_{tot} + k_i}\right)^2$$

Recall: $k_{i,in}$:



k_i :



Louvain: Modularity Gain

- $\Delta Q(i \to C) = Q_{\text{after}} - Q_{\text{before}}$

$$= \left[\frac{\Sigma_{in}+k_{i,in}}{2m} - \left(\frac{\Sigma_{tot}+k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

- $\Delta Q(D \to i)$ can be derived similarly.

- **In summary, we can compute:**

$$\Delta Q(D \to i \to C) = \Delta Q(D \to i) + \Delta Q(i \to C)$$

Louvain 1st Phase: Summary

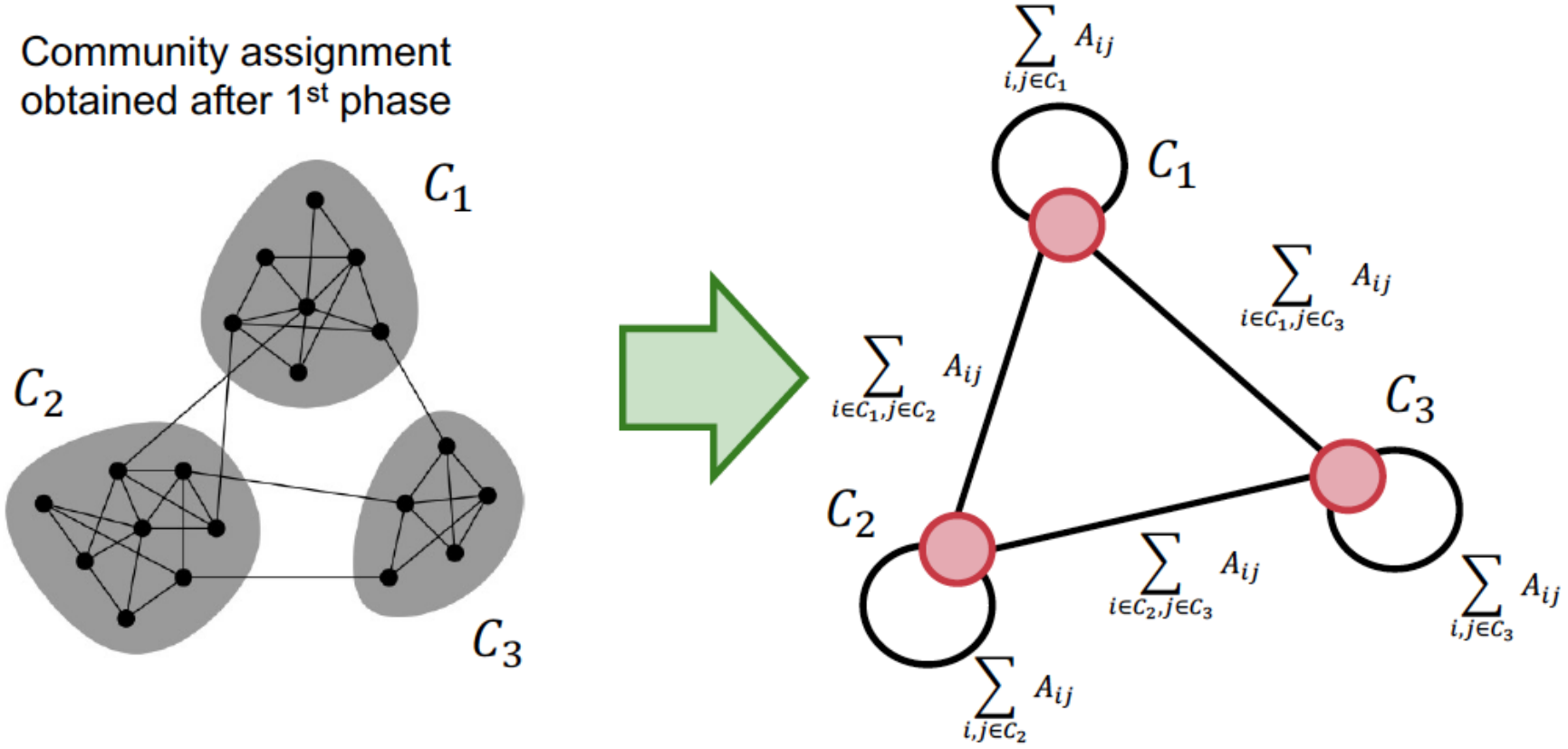
- Iterate until no node moves to a new community:
  - For each node $i \in V$ currently in community C, compute the best community C':
  - $C' = argmax_{C'} \Delta Q(C \rightarrow i \rightarrow C')$
  - If $\Delta Q(C \rightarrow i \rightarrow C') > 0$, then update the community:
    - $C \leftarrow C - \{i\}$
    - $C' \rightarrow C' + \{i\}$

Louvain: 2nd Phase (Restructuring)

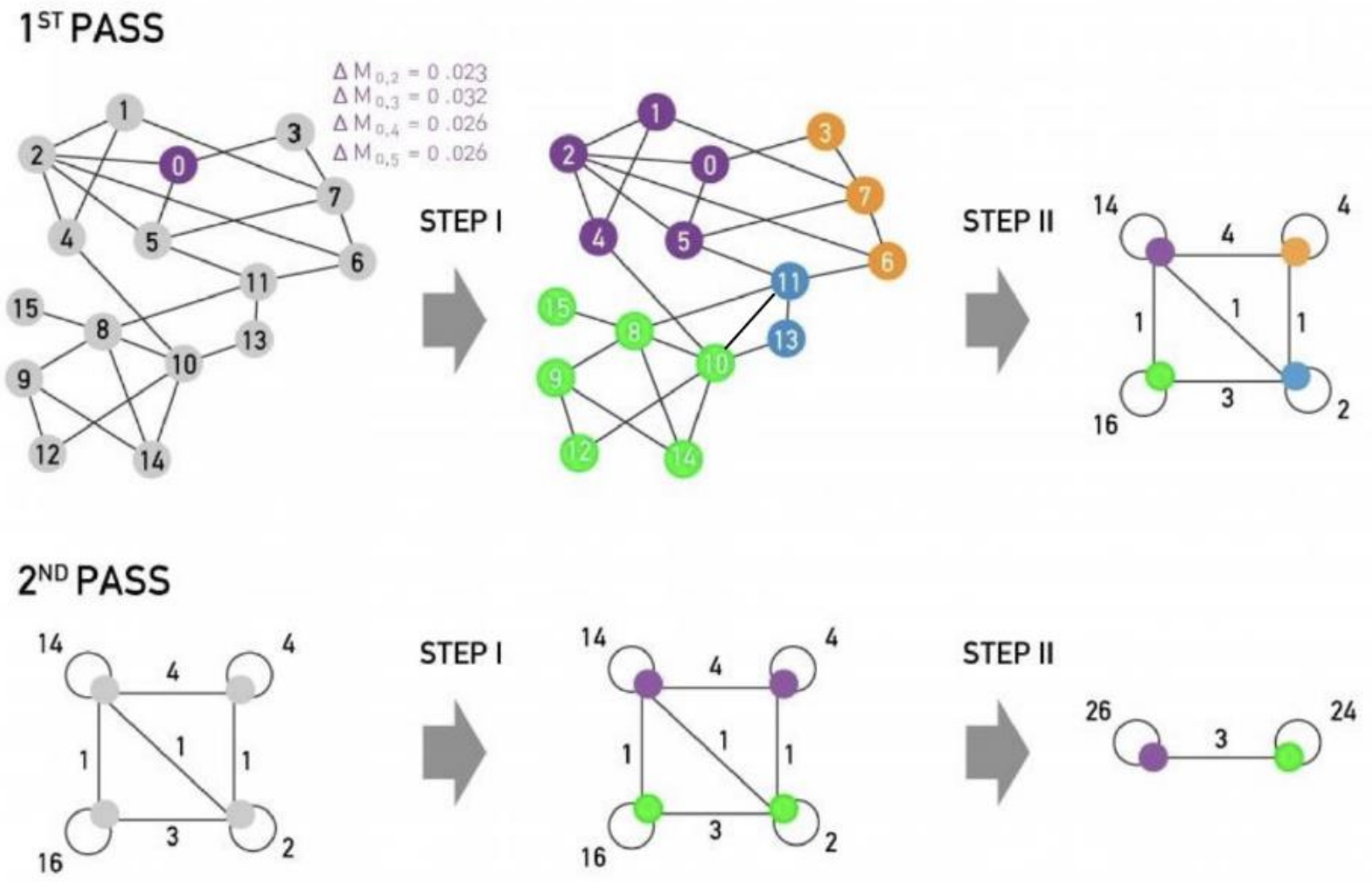
- The communities obtained in the first phase are contracted into super-nodes, and the network is created accordingly:
  - Super-nodes are connected if there is at least one edge between the nodes of the corresponding communities
  - The weight of the edge between the two super-nodes is the sum of the weights from all edges between their corresponding communities
  - Phase 1 is then run on the super-node network

Louvain 2nd Phase: Summary

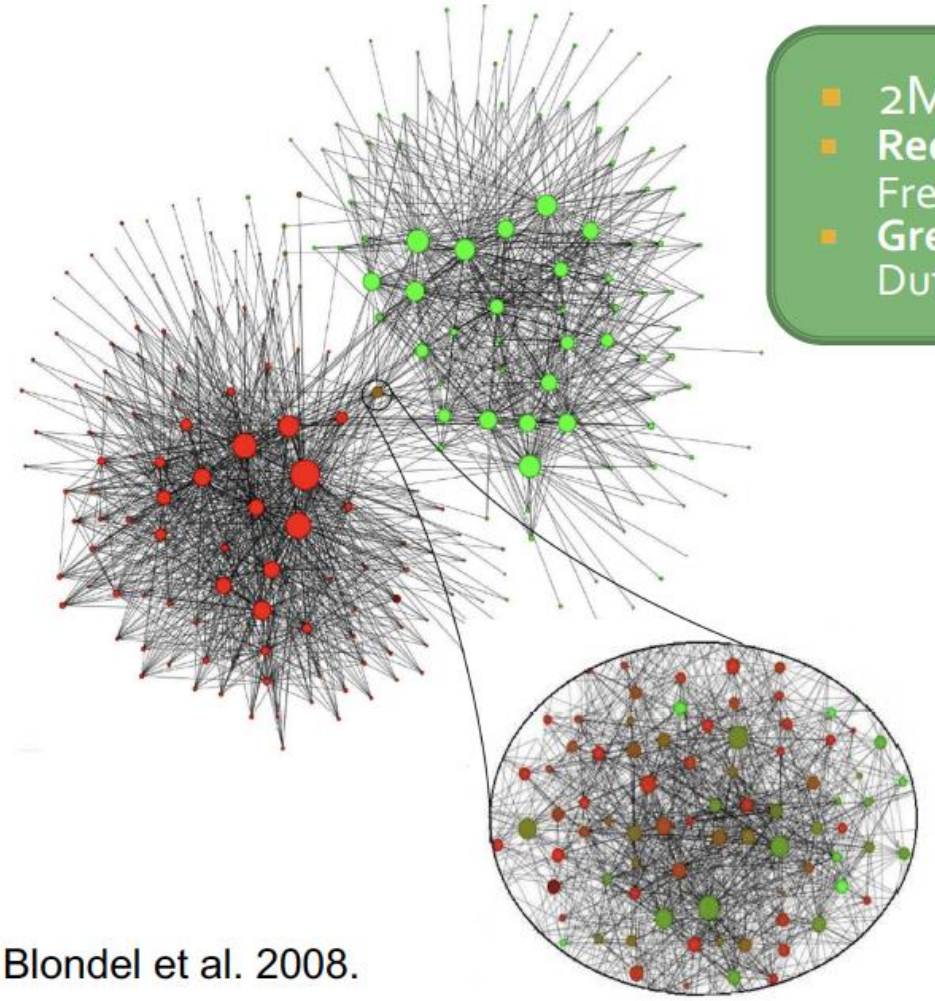
- Super nodes are constructed by merging nodes in the same community.



Louvain Algorithm



Belgian Mobile Phone Network



- 2M nodes
- **Red nodes**: French speakers
- **Green nodes**: Dutch speakers

Adopted from Blondel et al. 2008.

Summary: Modularity

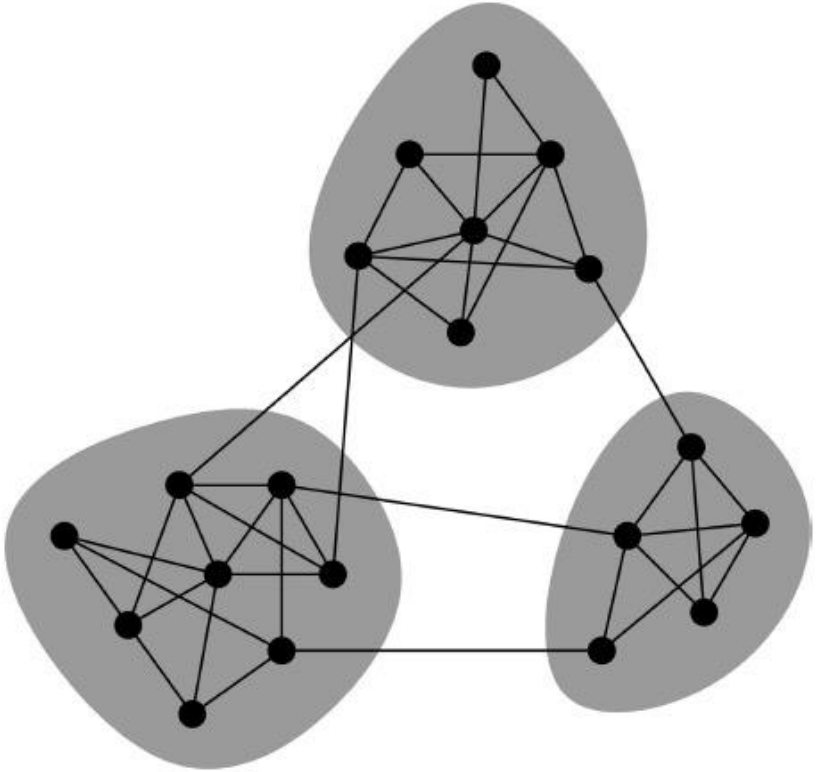
- Modularity:
  - Overall quality of the partitioning of a graph into communities
  - Used to determine the number of communities

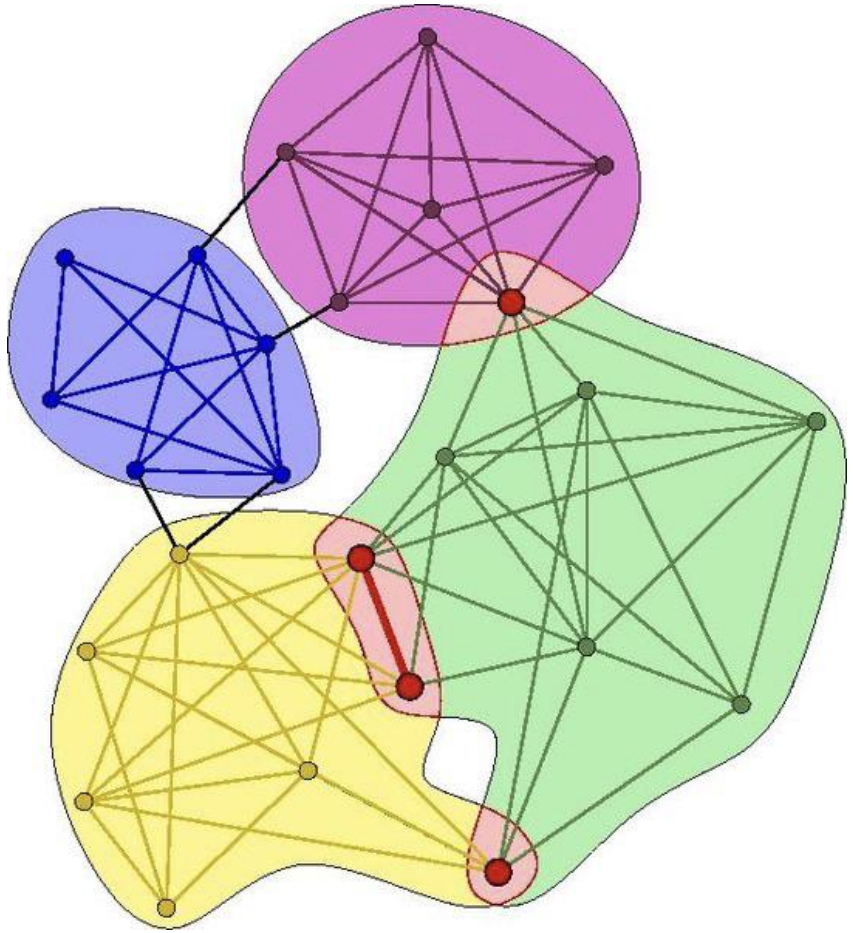
- Louvain modularity maximization:
  - Greedy strategy
  - Great performance, scales to large networks

Detecting Overlapping Community: AGM & NOCD

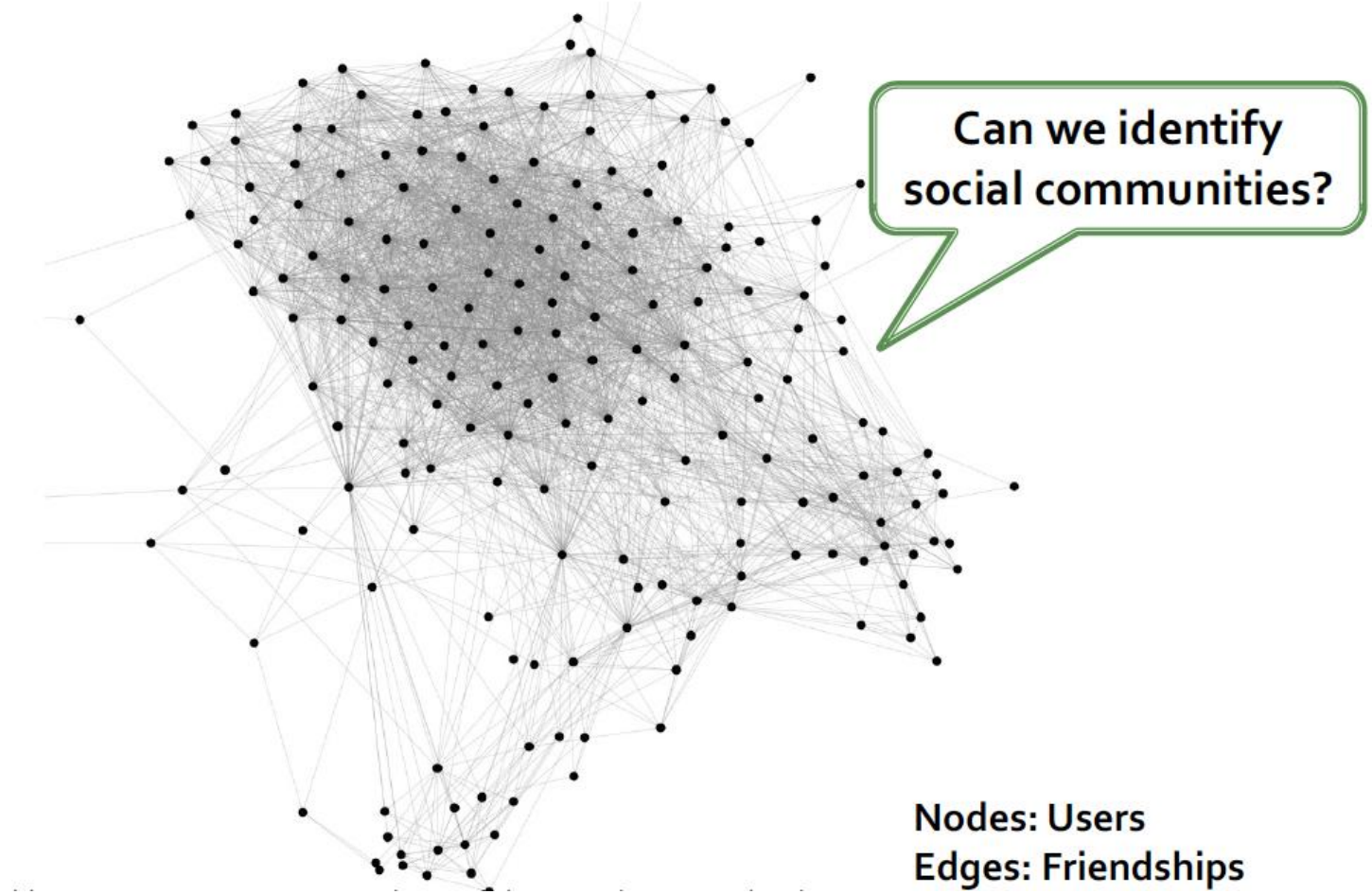
Overlapping Communities

- Non-overlapping vs. overlapping communities

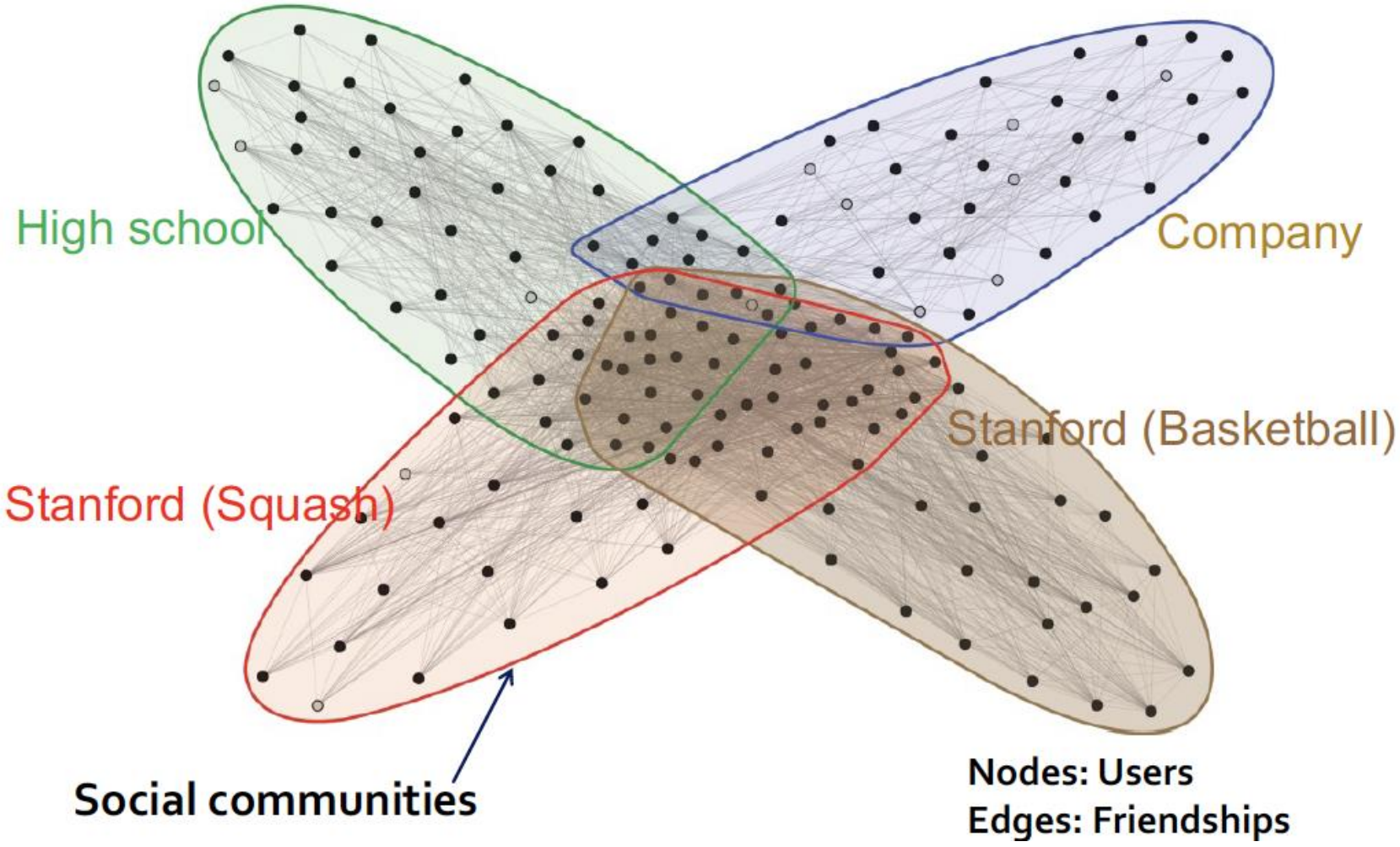




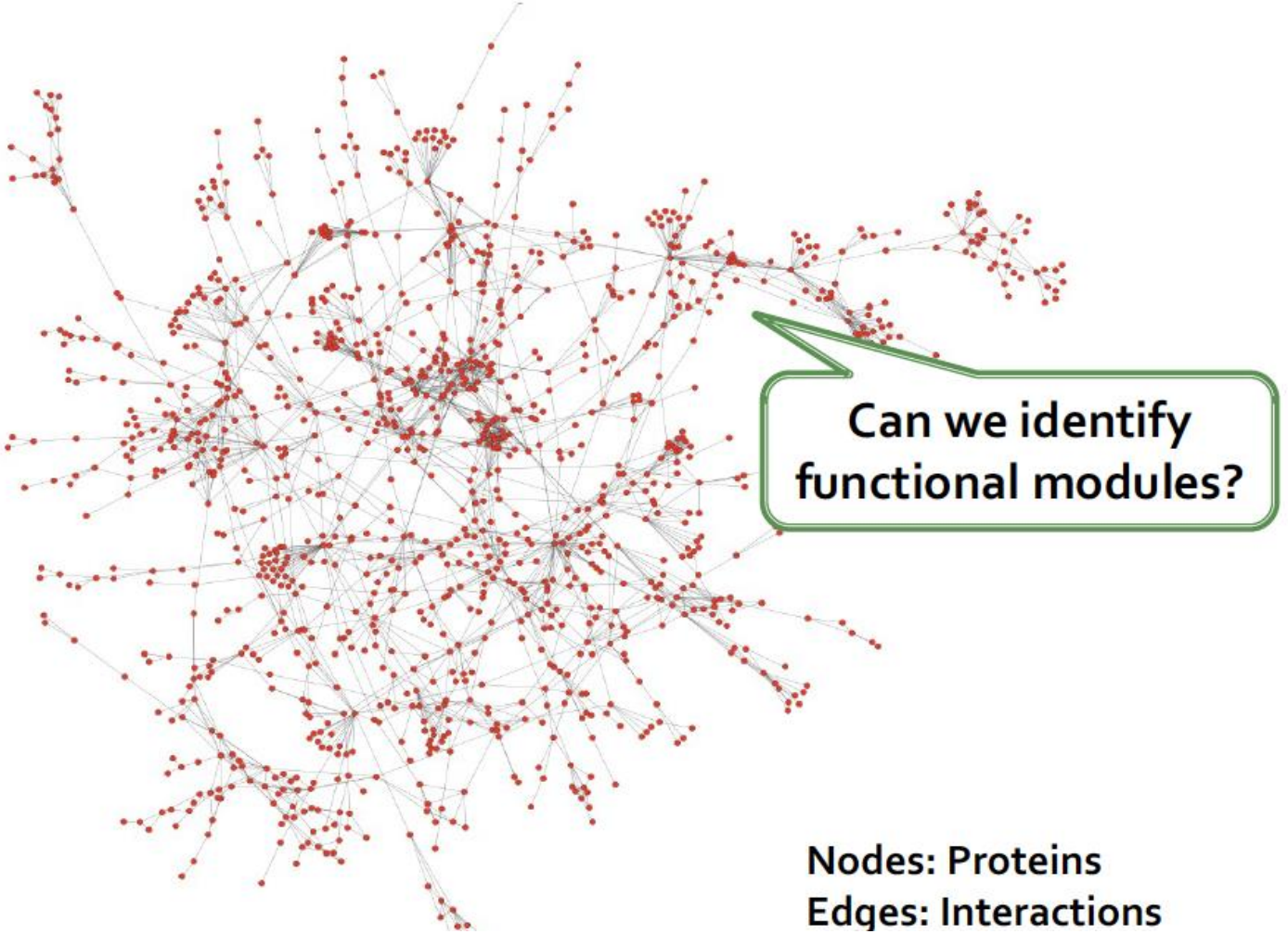
Facebook Ego-network



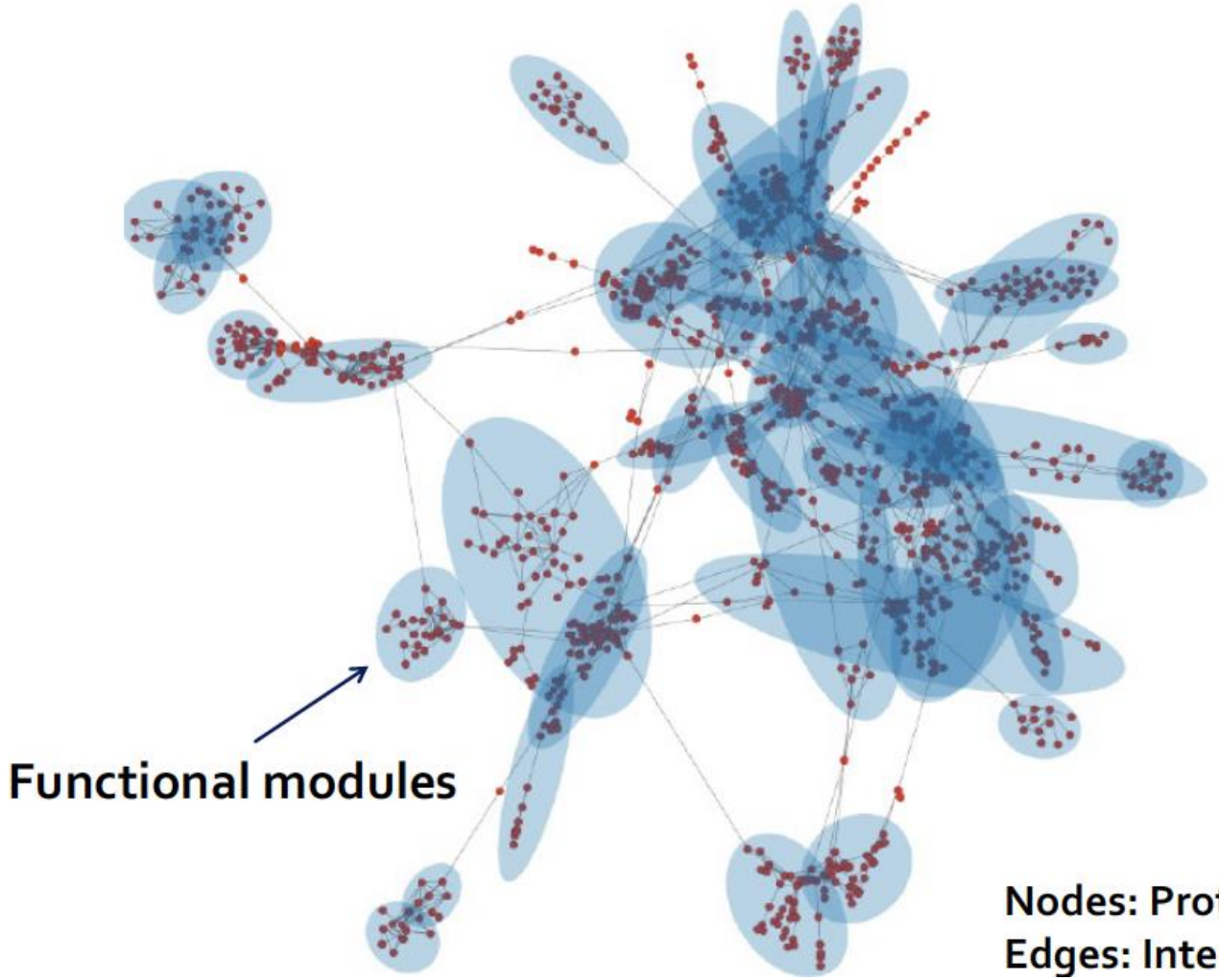
Facebook Ego-network



Protein-Protein Interaction



Protein-Protein Interaction



Plan of Actions

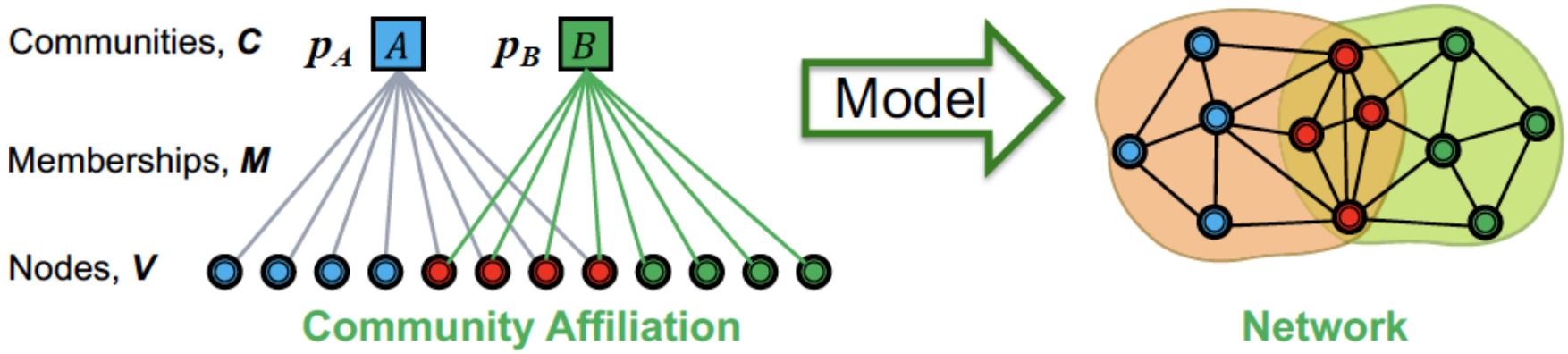
Step 1)

- Define a generative model for graphs that is based on node community affiliations
  - Community Affiliation Graph Model (AGM)

Step 2)

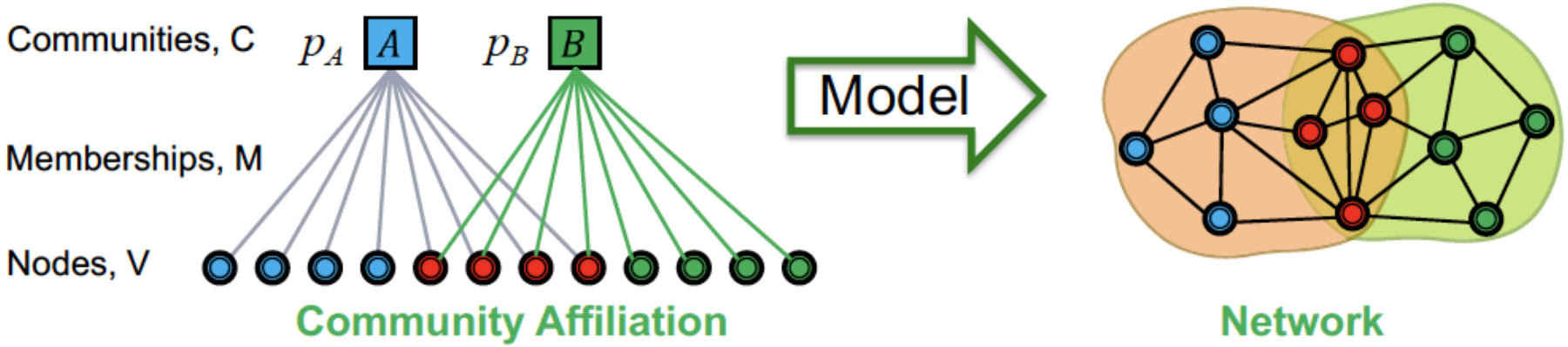
- Given graph G, make the assumption that G was generated by AGM
- Find the best AGM that could have generated G
- And this way we discover communities

Community-Affiliation Graph Model (AGM)



- Generative model: How is a network generated from community affiliations?
- Model parameters:
  - Nodes V, Communities C, Memberships M
  - Each community c has a single probability p_c

AGM: Generative Process

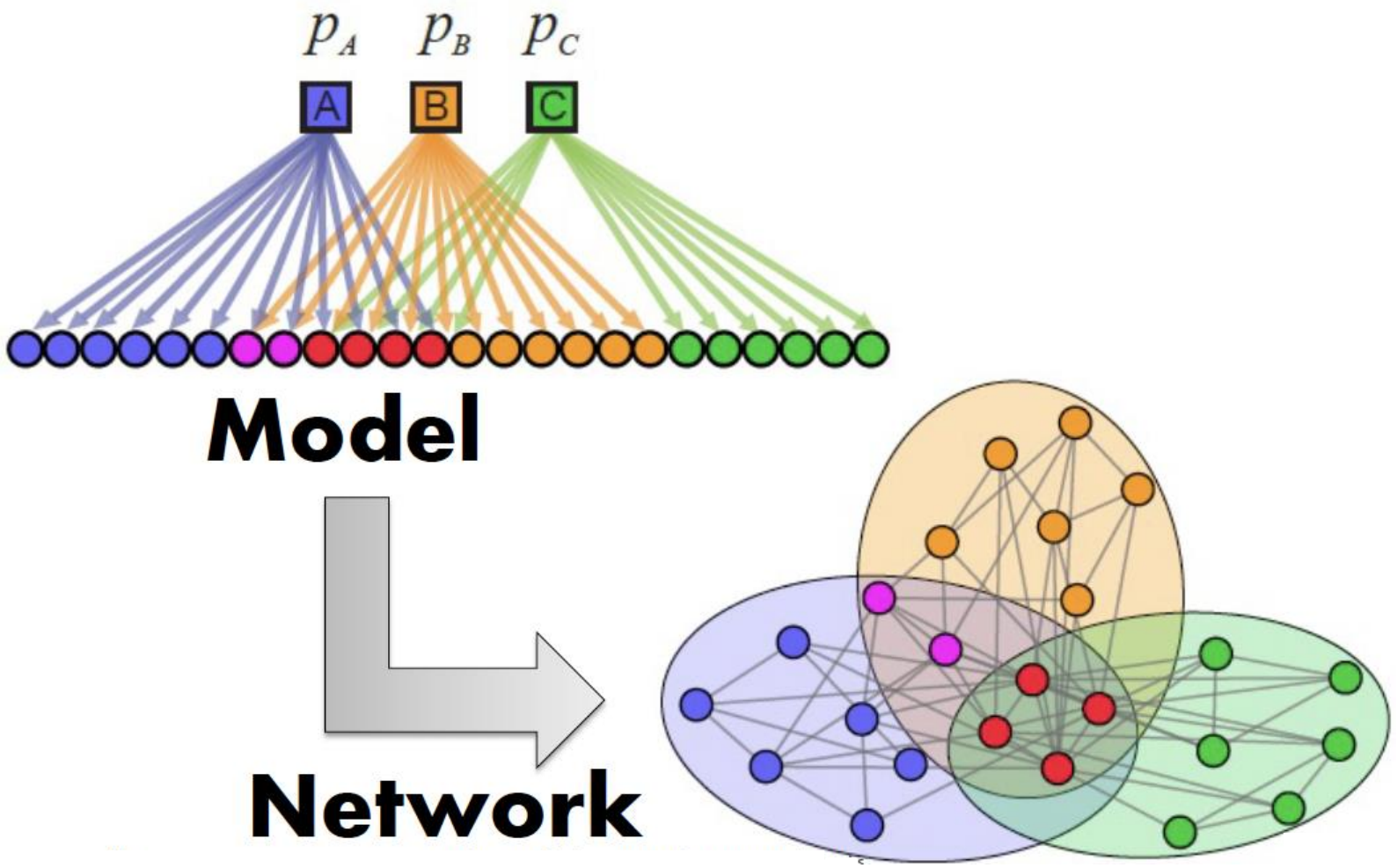


- Given parameters (V, C, M, {pc})
  - Nodes in community c connect to each other by flipping a coin with probability pc
  - Nodes that belong to multiple communities have multiple coin flips
    - If they "miss" the first time, they get another chance through the next community

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

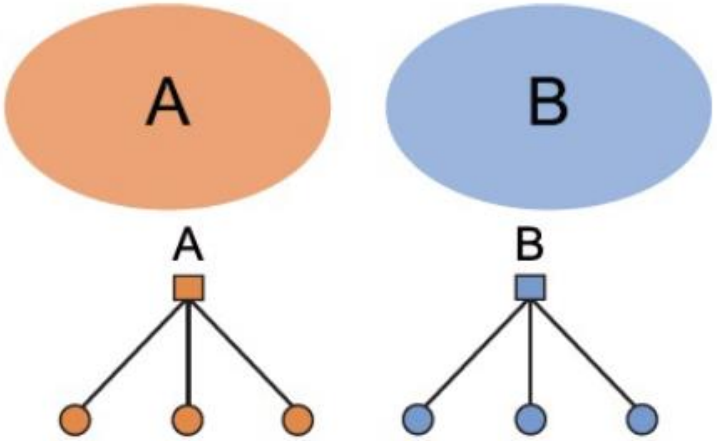
Note: If nodes u and v have no communities in common, then p(u,v)=0. We resolve this by having a background "epsilon" community that every node is a member of.

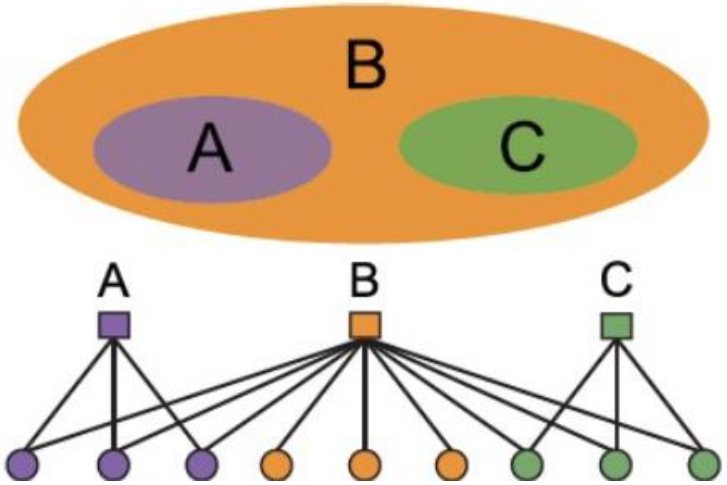
AGM: Dense Overlaps

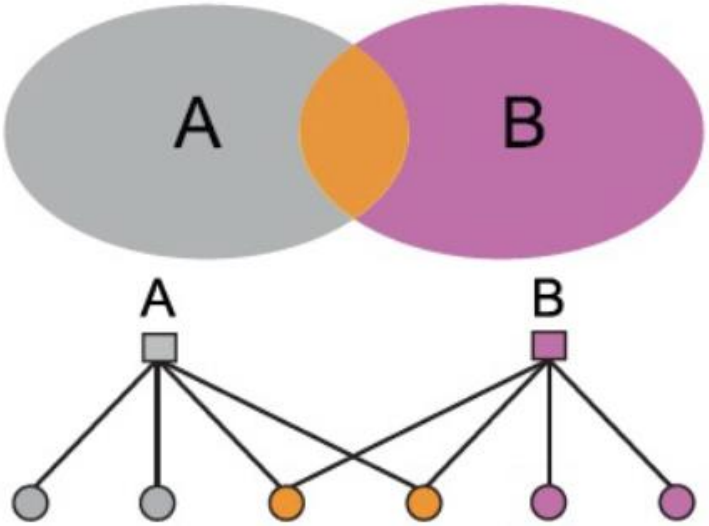


AGM: Flexibility

- **AGM can express a variety of community structures:**
  Non-overlapping,
  Overlapping, Nested

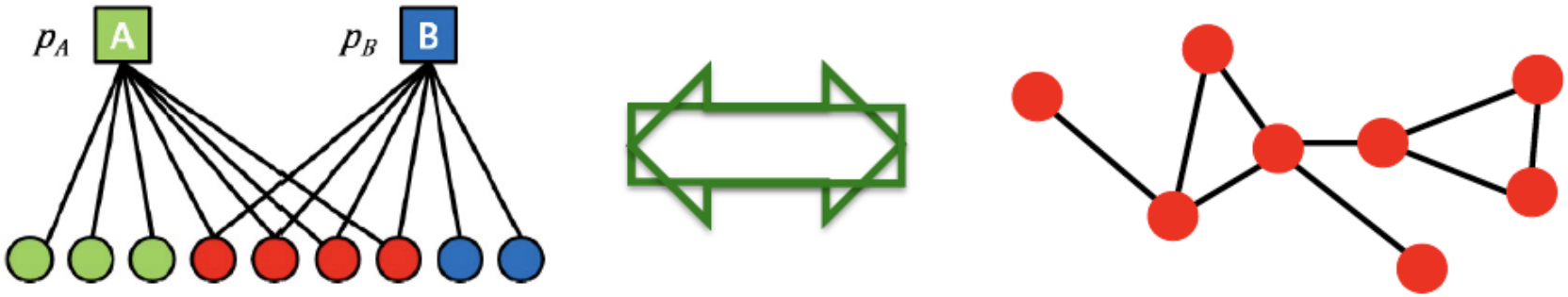






Detecting Community

- Detecting communities with AGM:



Given a Graph, find (fit) the model $\boldsymbol{F}$

Graph Fitting

How to estimate model parameters $\boldsymbol{F}$ given a $\boldsymbol{G}$?

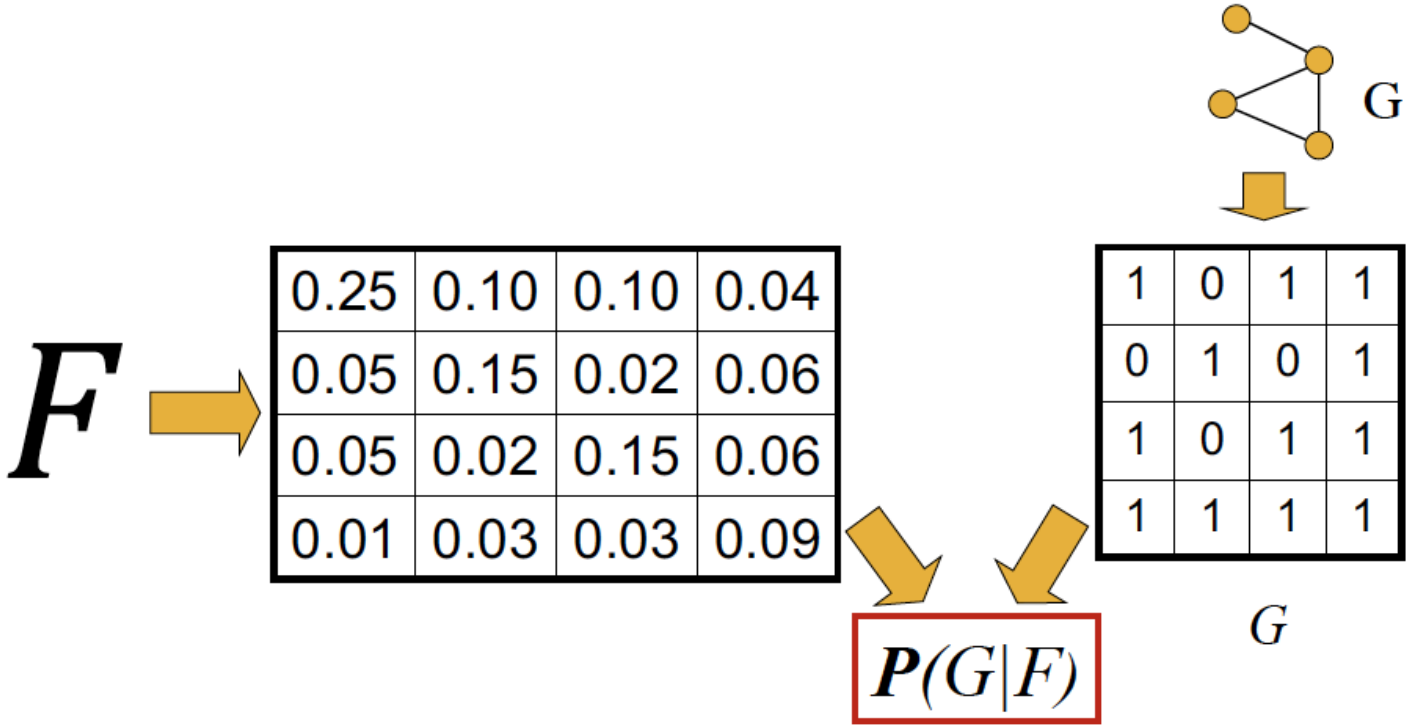
- Maximum likelihood estimation
- Given real graph G
- Find model/parameters F which

$$\arg\max_F P(\boldsymbol{G} \mid \text{Model} \leftarrow F)$$

- To solve this we need to:
  - Efficiently calculate $\boldsymbol{P(G \mid F)}$
  - Then maximize over $\boldsymbol{F}$ (e.g., using gradient descent)

Graph Likelihood P(G|F)

- Given G and F we calculate likelihood that F generated G: P(G | F)



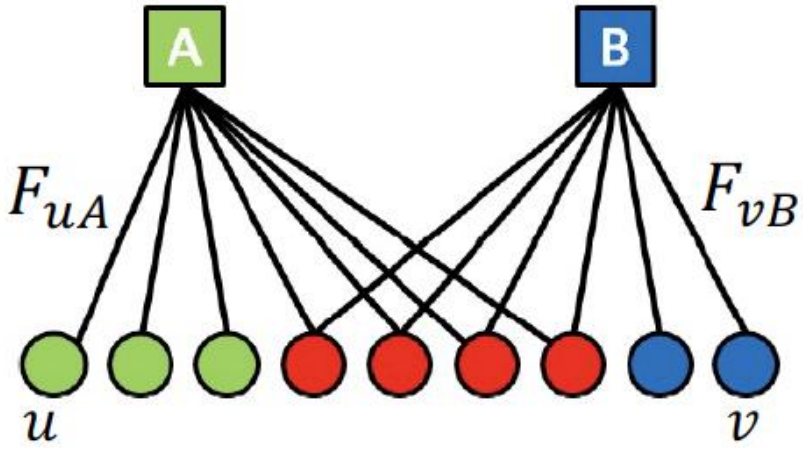
$$P(G|F) = \prod_{(u,v)\in G} P(u,v) \prod_{(u,v)\notin G} (1 - P(u,v))$$

Likelihood of edges in the graph

Likelihood of edges not in the graph

Relaxing AGM: Towards P(u,v)

- "Relax" the AGM: Memberships have strengths



  - F_{uA}: The membership strength of node u to community $\boldsymbol{A}$ ($F_{uA} = \mathbf{0}$: no membership)

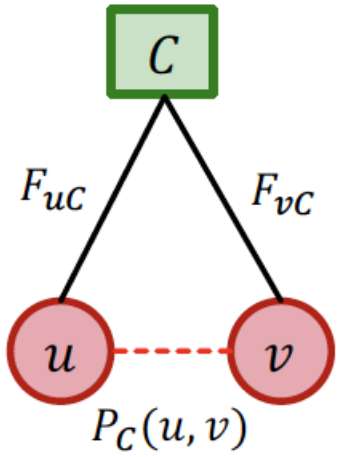
Relaxing AGM: Towards P(u,v)

- For community C, we model the probability of u and v being connected as

$$P_C(u,v) = 1 - exp(-F_{uC} \cdot F_{vC})$$

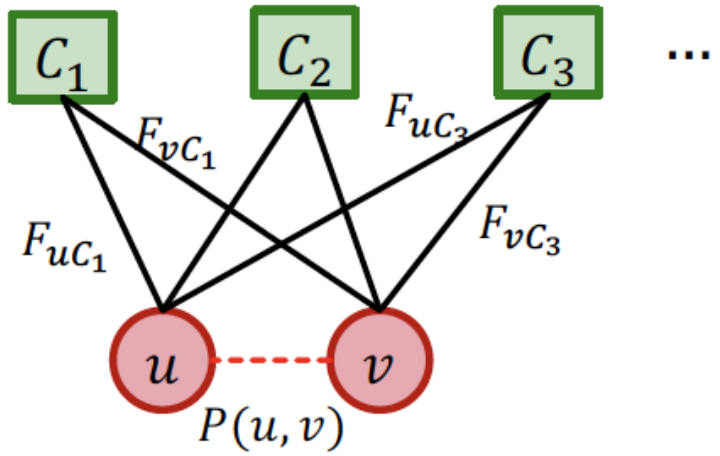
Non-negative membership strength

  - $P_C(u,v)$ satisfies $0 \leq P_C(u,v) \leq 1$ (valid probability) because $F_{uC} \cdot F_{vC} \geq 0$
    - $P_C(u,v) = 0$ iff $F_{uC} \cdot F_{vC} = 0$ (i.e., $F_{uC} = 0$ or $F_{vC} = 0$)
      - Nodes u or v are not connected via C iff at least one of them has zero membership strength for C.
    - $P_C(u,v) \approx 1$ iff $F_{uC} \cdot F_{vC}$ is large
      - Nodes u or v are connected via C iff both u and v have high membership strength for C.



Relaxing AGM: Towards P(u,v)

- Nodes u and v can connected via multiple communities $C_i \in \Gamma$ (a set of all communities).



- Probability that u and v are connected by at least one of the communities:

$$P(u,v) = 1 - \prod_{C \in \Gamma}(1 - P_C(u,v))$$

Probability that u and v are not connected by any communities

Relaxing AGM: Towards P(u,v)

- Expanding $\boldsymbol{P(u, v)}$:

$$P(u,v) = 1 - \prod_{C \in \Gamma} (1 - P_C(u,v))$$

$$1 - \exp(-F_{uC} \cdot F_{vC})$$

$$= 1 - \prod_{C \in \Gamma} \exp(-F_{uC} \cdot F_{vC})$$

$$= 1 - \exp\left(- \sum_{C \in \Gamma} F_{uC} \cdot F_{vC} \right)$$

$$= 1 - \exp(-\boldsymbol{F}_{\boldsymbol{u}}^{T} \boldsymbol{F}_{\boldsymbol{v}})$$

Dot product

$\boldsymbol{F}_{\boldsymbol{u}}$: A vector of $\{F_{uC}\}_{C \in \Gamma}$

$\boldsymbol{F}_{\boldsymbol{v}}$: A vector of $\{F_{vC}\}_{C \in \Gamma}$

NOCD Model

- Prob. of nodes $\boldsymbol{u}$, $\boldsymbol{v}$ linking is proportional to the strength of shared memberships:

$$\mathbf{P}(\boldsymbol{u},\boldsymbol{v}) = \mathbf{1} - \mathbf{exp}(-\boldsymbol{F}_{\boldsymbol{u}}^{\boldsymbol{T}}\boldsymbol{F}_{\boldsymbol{v}})$$

- Given a network $\boldsymbol{G}(\boldsymbol{V}, \boldsymbol{E})$, we maximize the likelihood (probability) of $\boldsymbol{G}$ under our model

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

$$= \prod_{(u,v)\in E} \left(\mathbf{1} - \mathbf{exp}(-\boldsymbol{F}_{\boldsymbol{u}}^{\boldsymbol{T}}\boldsymbol{F}_{\boldsymbol{v}})\right) \prod_{(u,v)\notin E} \mathbf{exp}(-\boldsymbol{F}_{\boldsymbol{u}}^{\boldsymbol{T}}\boldsymbol{F}_{\boldsymbol{v}})$$

NOCD Model

- Likelihood involves a product of many small probabilities →Numerically unstable.
- We consider the log likelihood:

$$\log(P(G|\boldsymbol{F}))$$

$$= \log\left(\prod_{(u,v)\in E}\left(1-\exp(-\boldsymbol{F}_u^T\boldsymbol{F}_v)\right)\prod_{(u,v)\notin E}\exp(-\boldsymbol{F}_u^T\boldsymbol{F}_v)\right)$$

$$= \sum_{(u,v)\in E}\log(1-\exp(-\boldsymbol{F}_u^T\boldsymbol{F}_v)) - \sum_{(u,v)\notin E}\boldsymbol{F}_u^T\boldsymbol{F}_v$$

$\equiv \ell(\boldsymbol{F})$: **Our objective**

NOCD Model

- Recall that

F →

0.25	0.10	0.10	0.04
0.05	0.15	0.02	0.06
0.05	0.02	0.15	0.06
0.01	0.03	0.03	0.09

- Given a graph $G = (A, X)$, how do we find F?
  - F… matrix of node-to-community membership

Key idea of Neural Overlapping Community Detection (NOCD)

Generate $\boldsymbol{F}$ using a GNN!
Train a GNN $\boldsymbol{A}$,$\boldsymbol{X}$ to output $\boldsymbol{F}$

Why Use a GNN?

- Benefits of GNN:
  - GNN can generalize to other graphs! Otherwise, we need to optimize F for each new graph.
  - GNN takes good use of the graph structure – adjacency matrix $\boldsymbol{A}$ and node feature $\boldsymbol{X}$.

For example: X includes occupation, hobbies, education of users on FB graph. After trained on FB graph, the GNN may generalize to another social network from Instagram / Twitter.

NOCD Model

- Consider a 2-layer GCN as an example

The first layer of GCN

$$\boldsymbol{F} = GCN(A, X) = \sigma\Big(\tilde{A}\,\sigma\big(\tilde{A}XW_1\big)\,W_2\Big)$$

where $\tilde{A} = D^{-1}A$.

The second layer of GCN

- Optimizing $\ell(\boldsymbol{F})$ w.r.t GCN parameters W_1 and W_2

$$\ell(\boldsymbol{F}) = \sum_{(u,v)\in E} \log(1 - \exp(-\boldsymbol{F}_u^T\boldsymbol{F}_v)) - \sum_{(u,v)\notin E} \boldsymbol{F}_u^T\boldsymbol{F}_v$$

NOCD Model

- Issues with $\ell(\boldsymbol{F})$

$$\ell(\boldsymbol{F}) = \sum_{(u,v)\in E} \log(1 - \exp(-\boldsymbol{F}_u^T \boldsymbol{F}_v)) - \sum_{(u,v)\notin E} \boldsymbol{F}_u^T \boldsymbol{F}_v$$

- Real-world graphs are often extremely sparse.

$$\frac{\#\text{ existing edges}}{\#\text{ possible edges}} = \frac{|E|}{n^2} \ll 1$$

FB ego-network
Ratio: 0.0054

PPI network
Ratio: 0.000736

NOCD Model

- Issues with $\ell(\boldsymbol{F})$

$$\ell(\boldsymbol{F}) = \sum_{(u,v)\in E} \log(1 - \exp(-\boldsymbol{F}_u^T \boldsymbol{F}_v)) - \sum_{(u,v)\notin E} \boldsymbol{F}_u^T \boldsymbol{F}_v$$

- Real-world graphs are often extremely sparse.
- The second term has a much larger contribution.
- Solution: Take average of both terms

$$\ell(\boldsymbol{F}) = \frac{1}{|E|} \sum_{(u,v)\in E} \log(1 - \exp(-\boldsymbol{F}_u^T \boldsymbol{F}_v)) - \frac{1}{n^2 - |E|} \sum_{(u,v)\notin E} \boldsymbol{F}_u^T \boldsymbol{F}_v$$

$\frac{1}{|E|}$: # existing edges

$\frac{1}{n^2 - |E|}$: # non-existing edges

NOCD Model

- After optimizing $\ell(\boldsymbol{F})$

$$\ell(\boldsymbol{F}) = \frac{1}{|E|} \sum_{(u,v)\in E} \log(1 - \exp(-\boldsymbol{F}_u^T \boldsymbol{F}_v)) - \frac{1}{n^2 - |E|} \sum_{(u,v)\notin E} \boldsymbol{F}_u^T \boldsymbol{F}_v$$

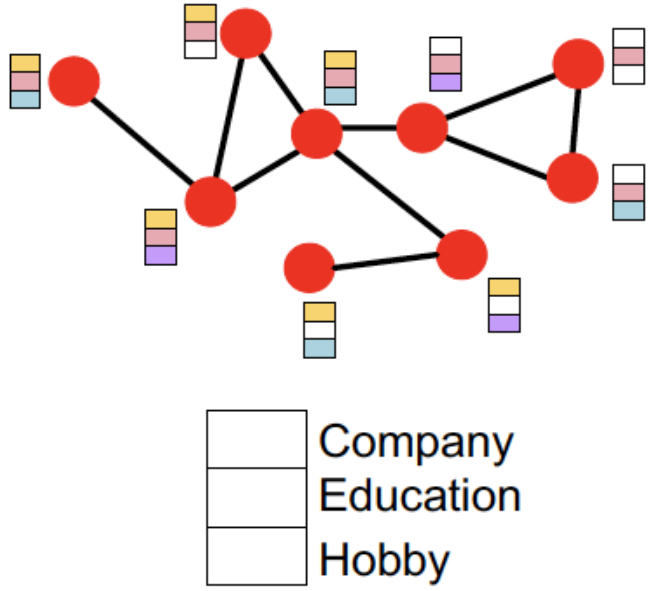
- Assign nodes to communities
  - Set a threshold ρ:
    - a hyperparameter, NOCD picks $\rho = 0.5$
  - Assign node u to community C if $\boldsymbol{F}_{uC} > \rho$

The strength of node u belong to C

Synthetic Example

- Given
  - a synthetic social network
  - #nodes = 9
  - set #communities = 2 (hyperparameters)
- NOCD uses a GNN to output $\boldsymbol{F}$:

0.65	0.10
0.75	0.15
0.54	0.02
0.66	0.73
0.8	0.72
0.77	0.66
0.59	0.85
0.2	0.88
0.15	0.9



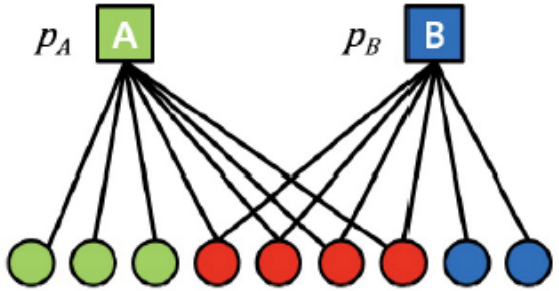
F is a #node by #community matrix.
An entry $F_{ij} \in [0,1]$ represents the strength of node i in community j

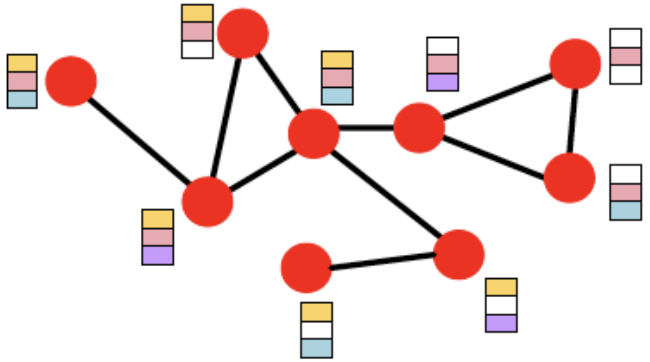
Synthetic Example

- Given
  - a synthetic social network
  - #nodes = 9
  - set #communities = 2 (hyperparameters)
- NOCD uses a GNN to output $\boldsymbol{F}$:

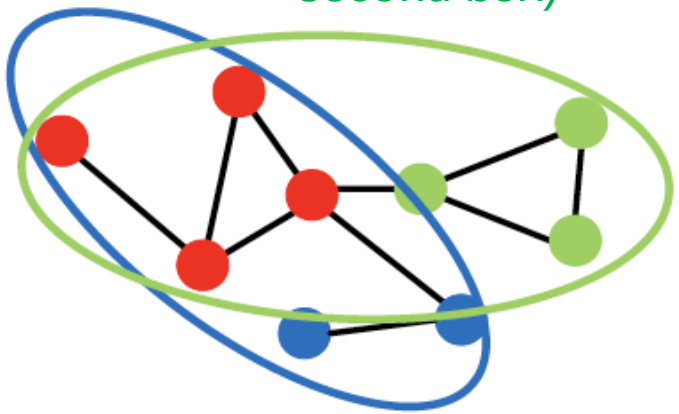
0.65	0.10
0.75	0.15
0.54	0.02
0.66	0.73
0.8	0.72
0.77	0.66
0.59	0.85
0.2	0.88
0.15	0.9

Set threshold $\rho = 0.5$, we can derive the affiliation graph.





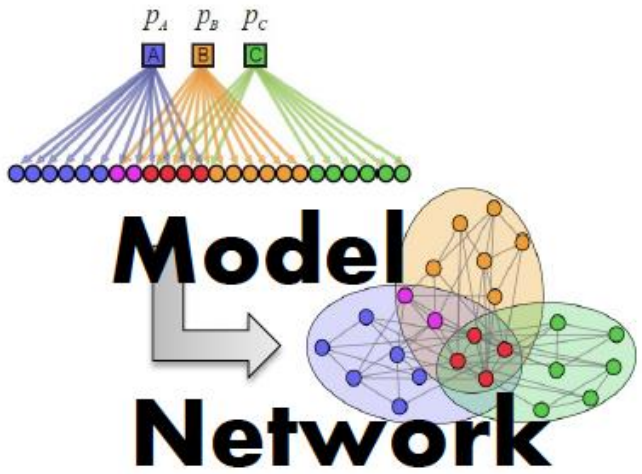
same education background (the second box)



same company (the first box)

NOCD Summary

- NOCD defines the model to generate a network with overlapping community structure.



- Given a graph, NOCD's parameters (membership strength of each node) can be estimated by maximizing the log-likelihood of generating the graph under the model.