Online Social Networks and Media

Community detection

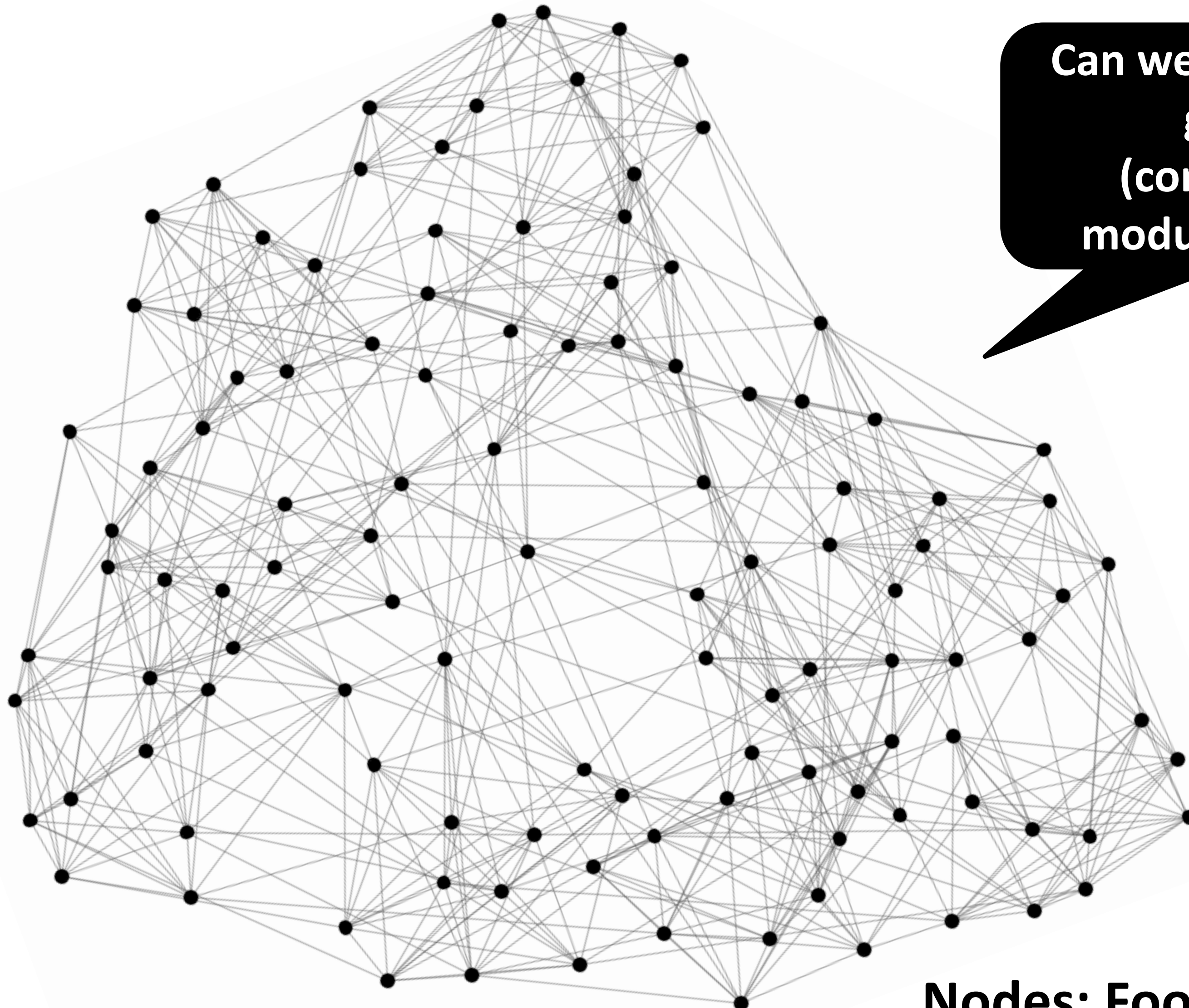
Introduction

Real networks are *not random graphs*

Communities
aka: groups, clusters, cohesive subgroups, modules

(informal) Definition: groups of vertices which probably share *common properties* and/or play *similar roles* within the graph

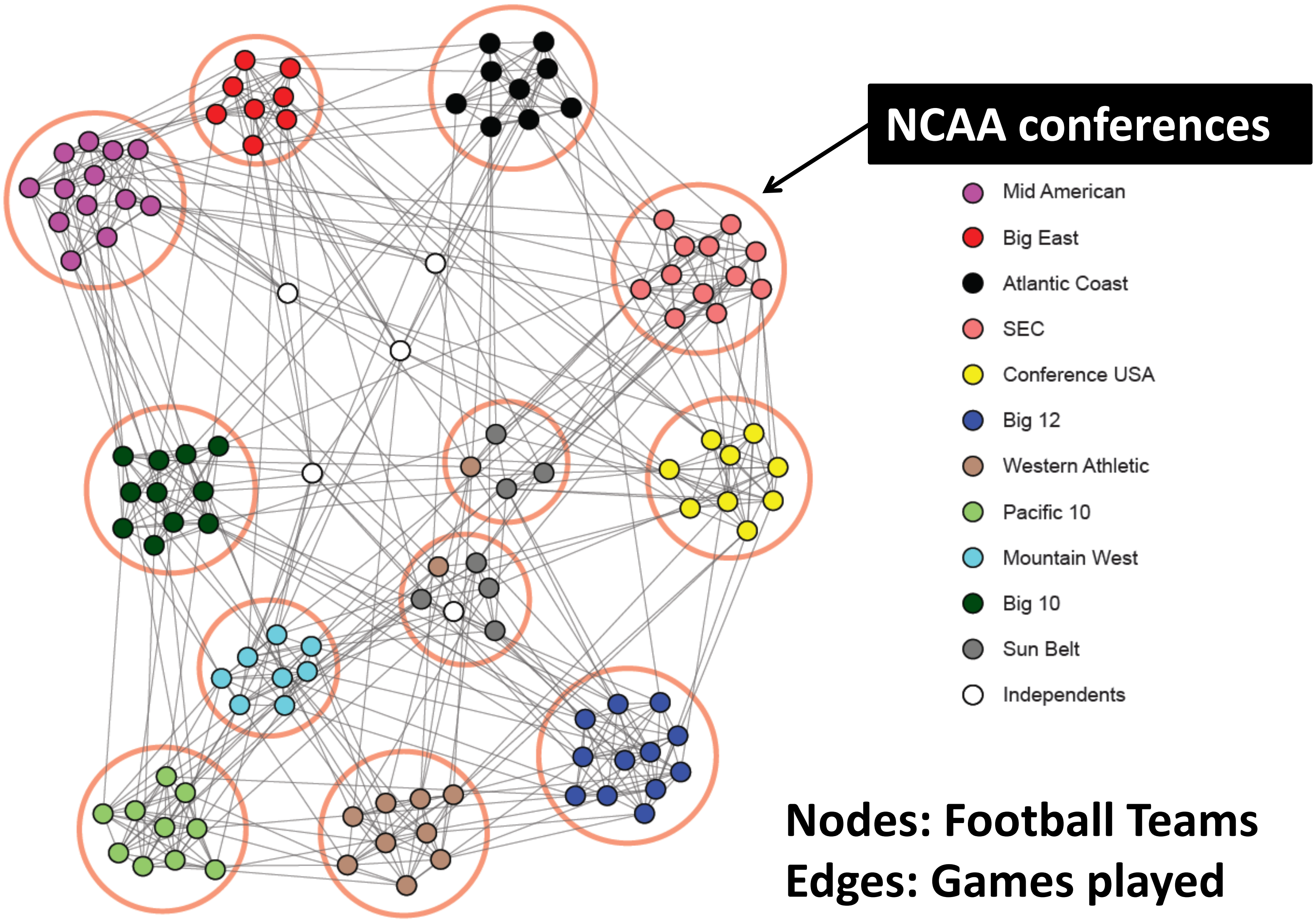
Some are *explicit (emic)* (e.g., Facebook (groups), LinkedIn (groups, associations), etc), we are interested in *implicit (etic)* ones



Can we identify node groups?
(communities, modules, clusters)

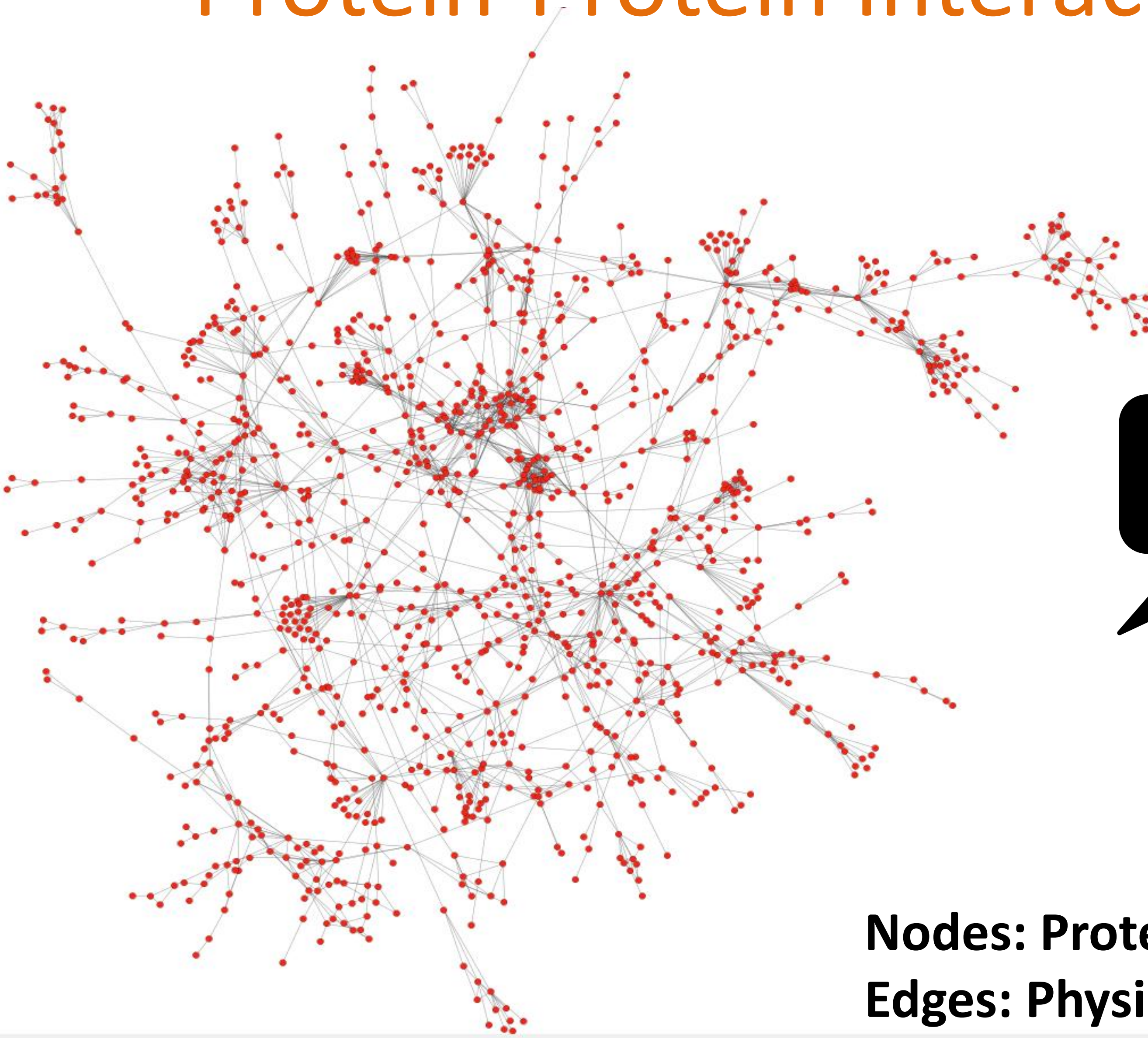
Nodes: Football Teams
Edges: Games played

NCAA Football Network



Nodes: Football Teams
Edges: Games played

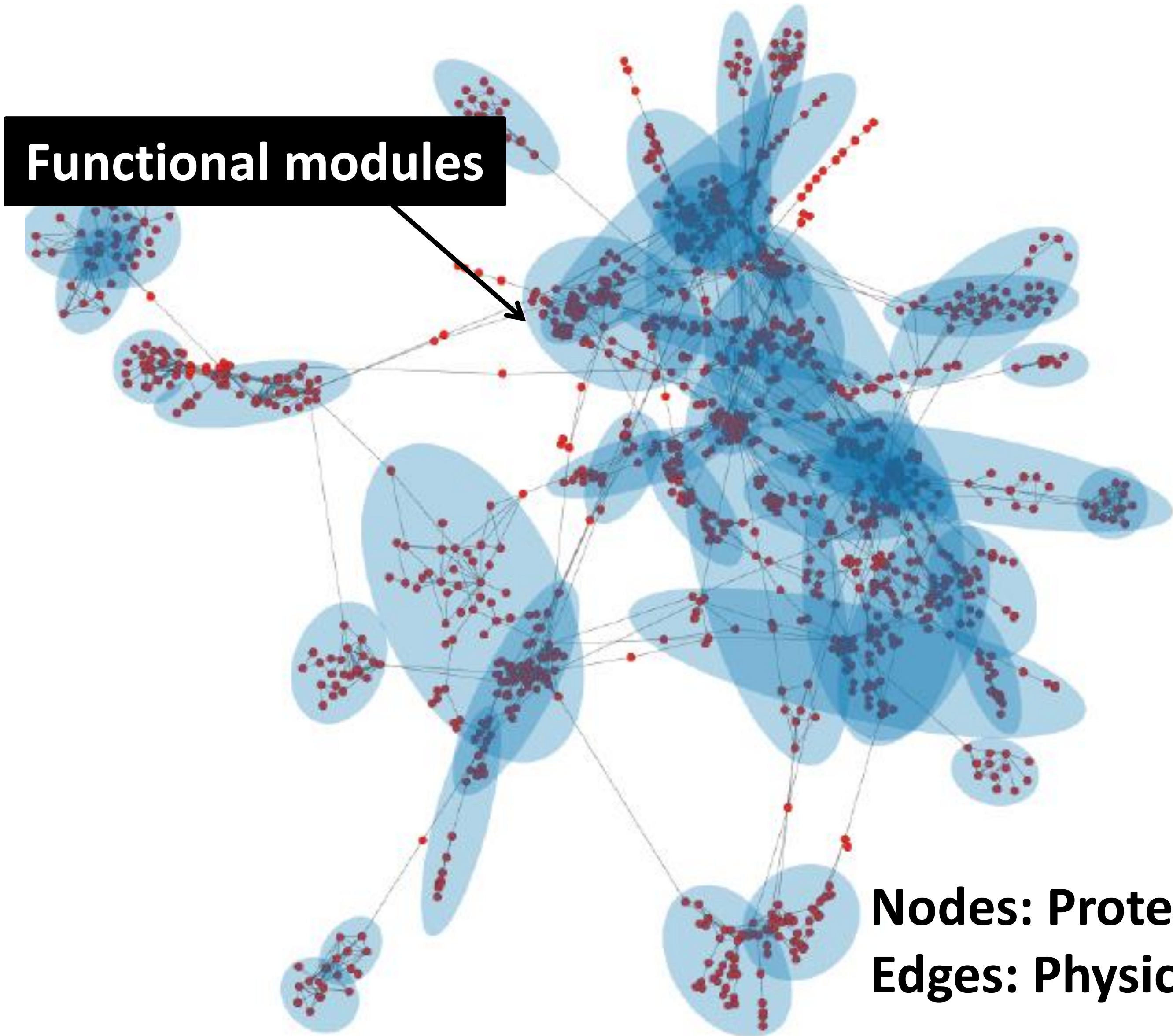
Protein-Protein Interactions



Can we identify functional modules?

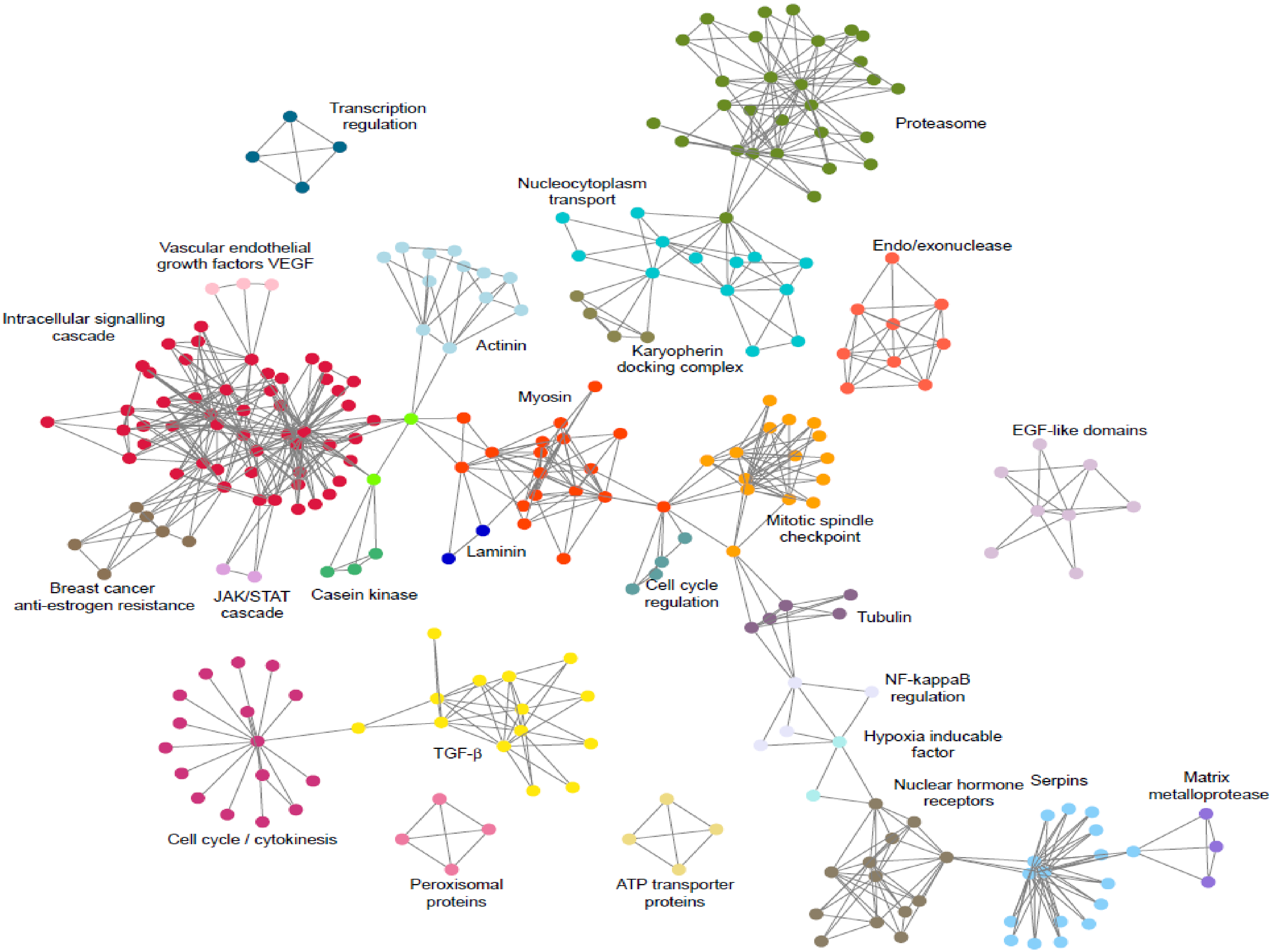
Nodes: Proteins
Edges: Physical interactions

Protein-Protein Interactions

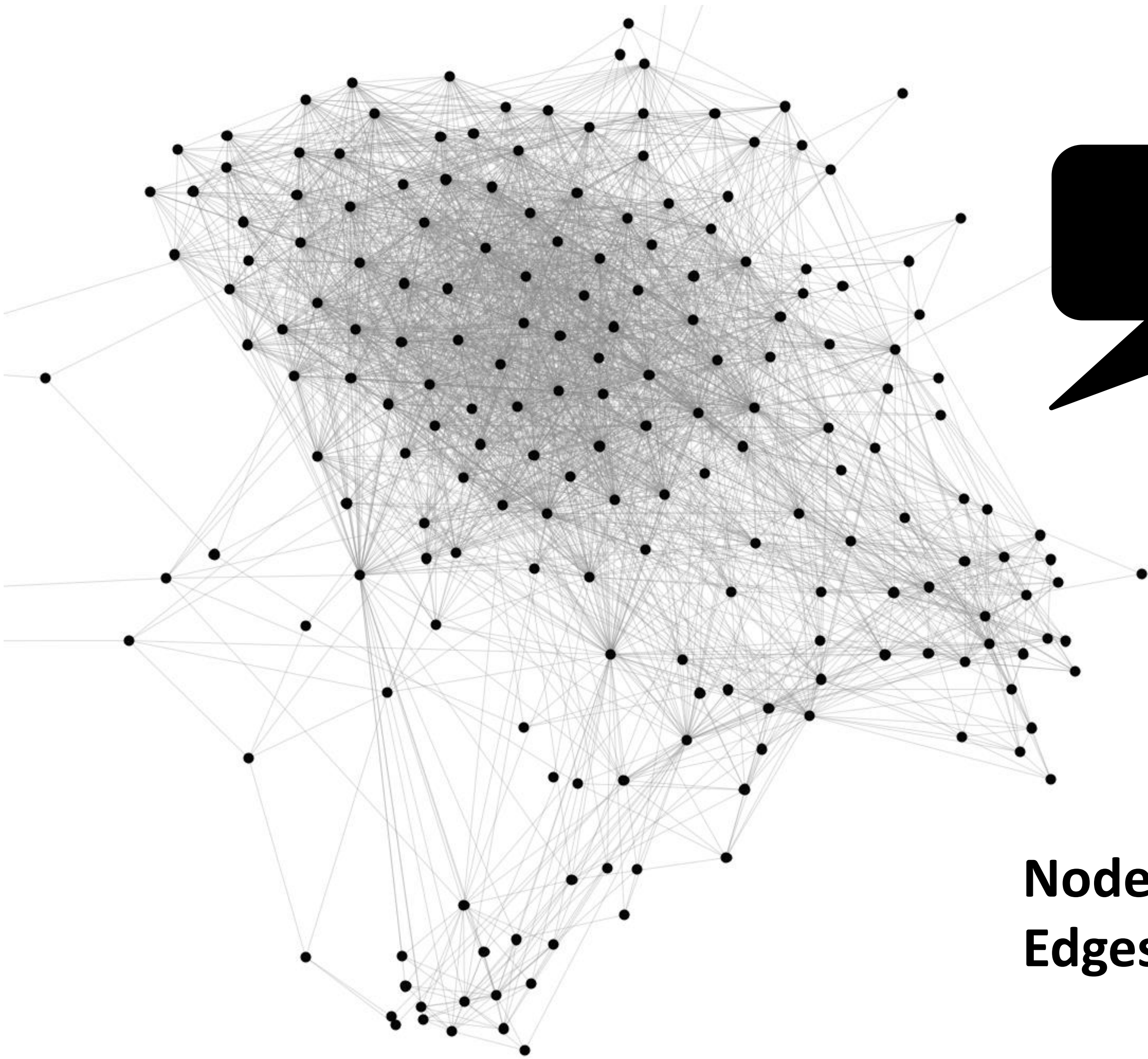


Nodes: Proteins
Edges: Physical interactions

Protein-Protein Interactions



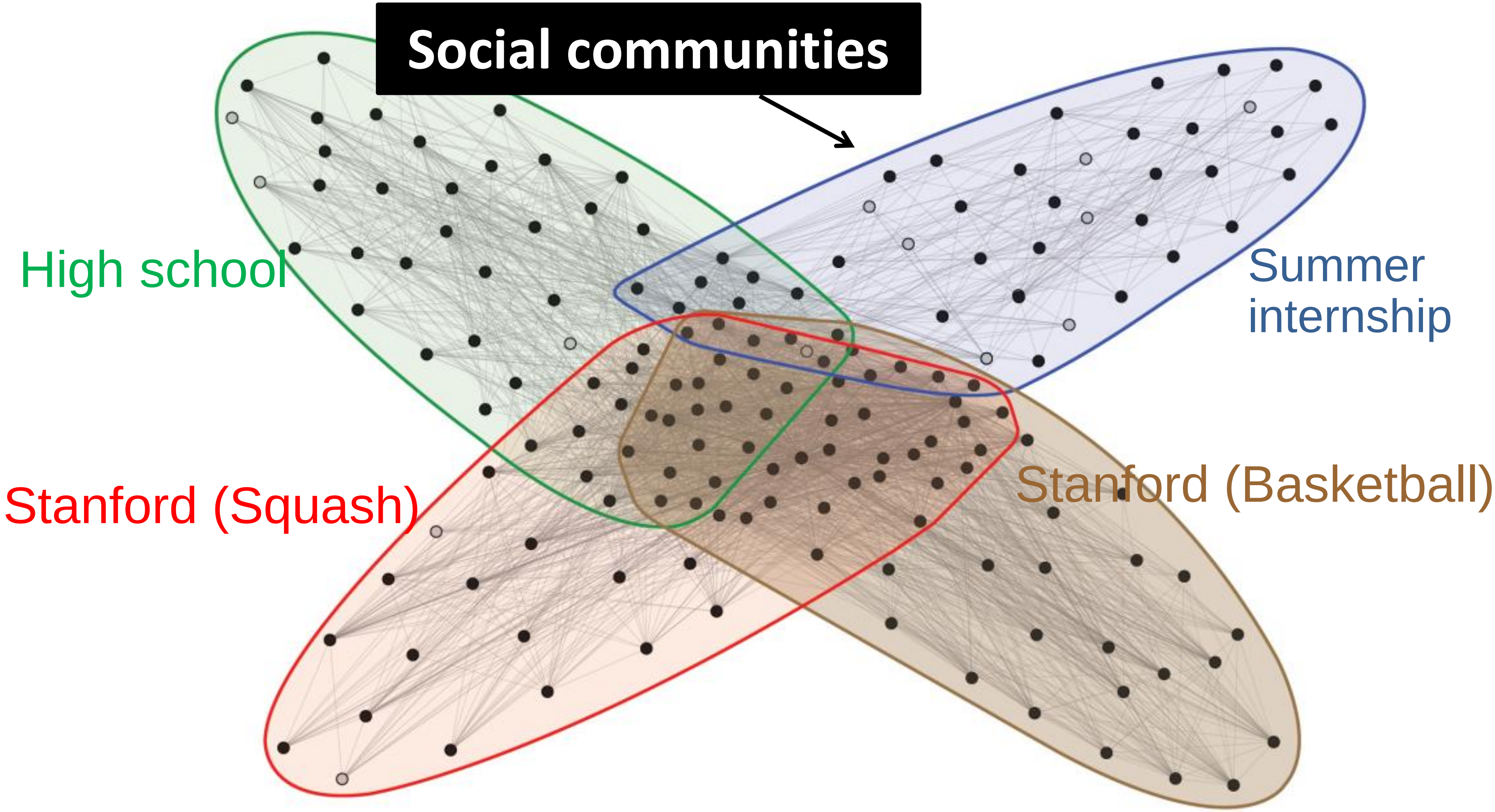
Facebook Network



Can we identify social communities?

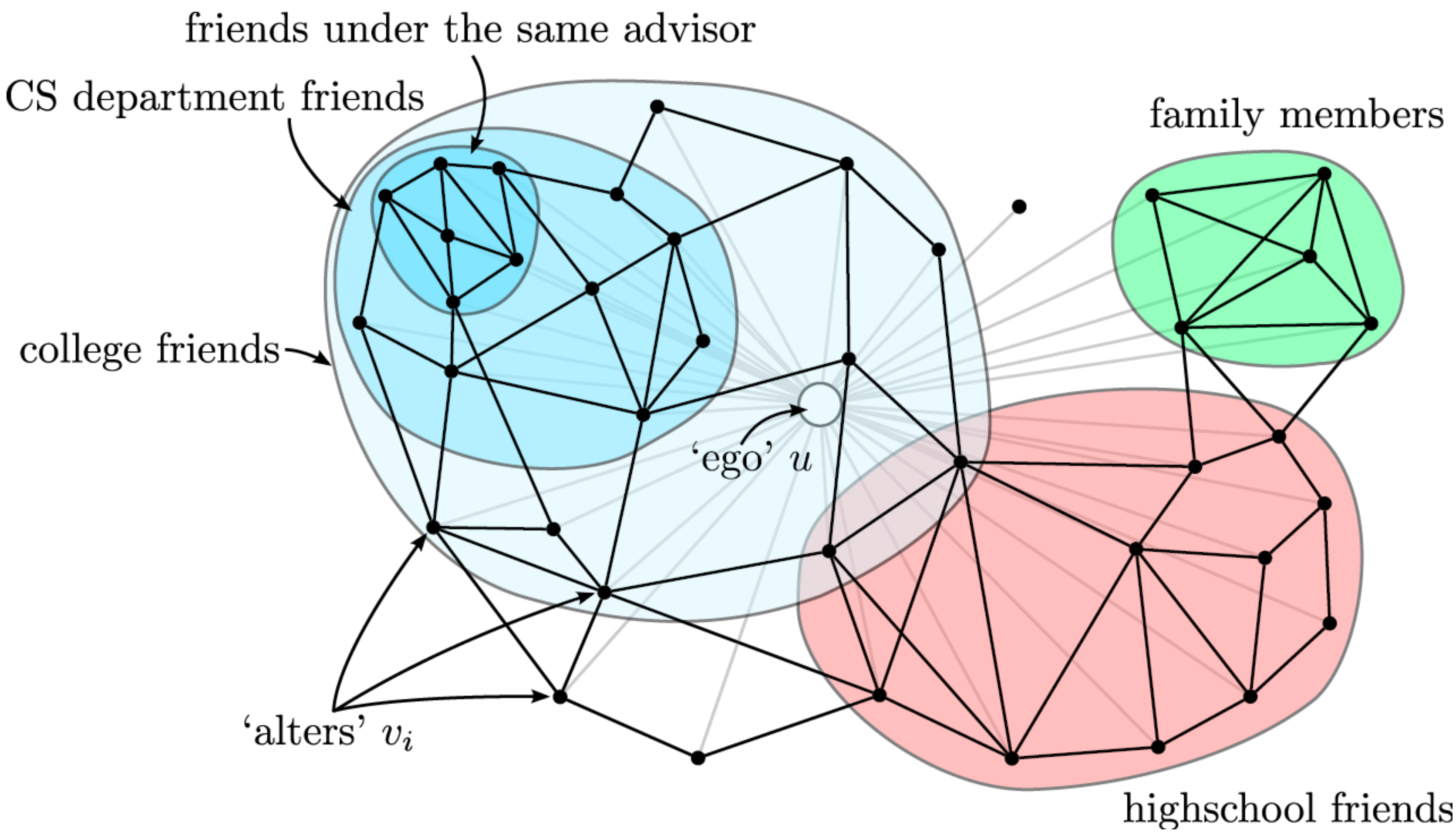
Nodes: Facebook Users
Edges: Friendships

Facebook Network



Nodes: Facebook Users
Edges: Friendships

Twitter & Facebook

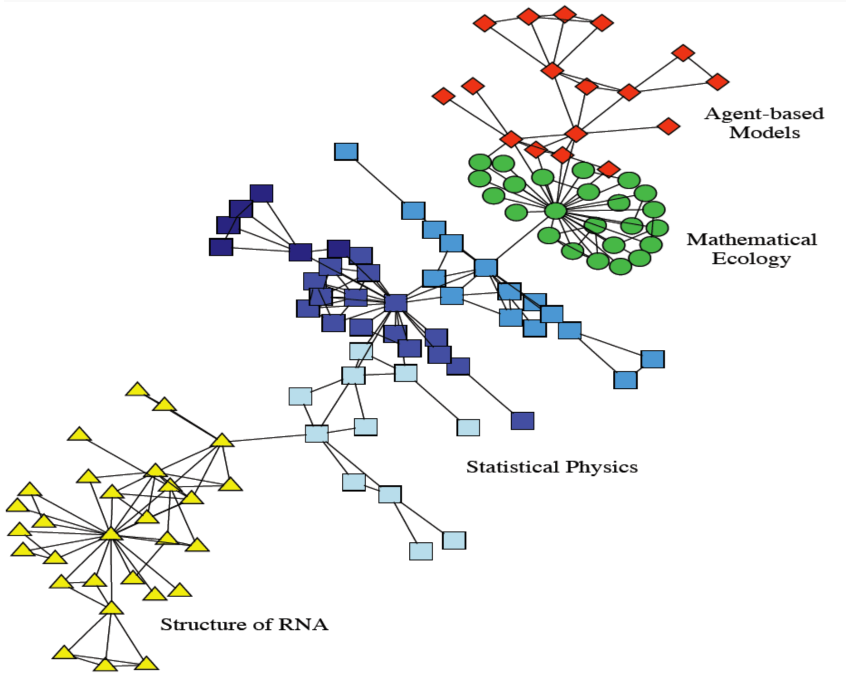


social circles, circles of trust

Collaboration Network

Collaboration network between scientists working at the Santa Fe Institute.

The colors indicate high level communities and correspond to research divisions of the institute



Outline

PART I

1. Introduction: what, why, types?
2. Cliques
3. Background: How it relates to "cluster analysis" (node/edge similarity)
4. Betweeness centrality
5. Modularity, label propagation

Outline

PART II (next lecture)

Cuts and Spectral clustering,

Denser subgraphs

How to evaluate

We will revisit the issue when we talk about Graph ML

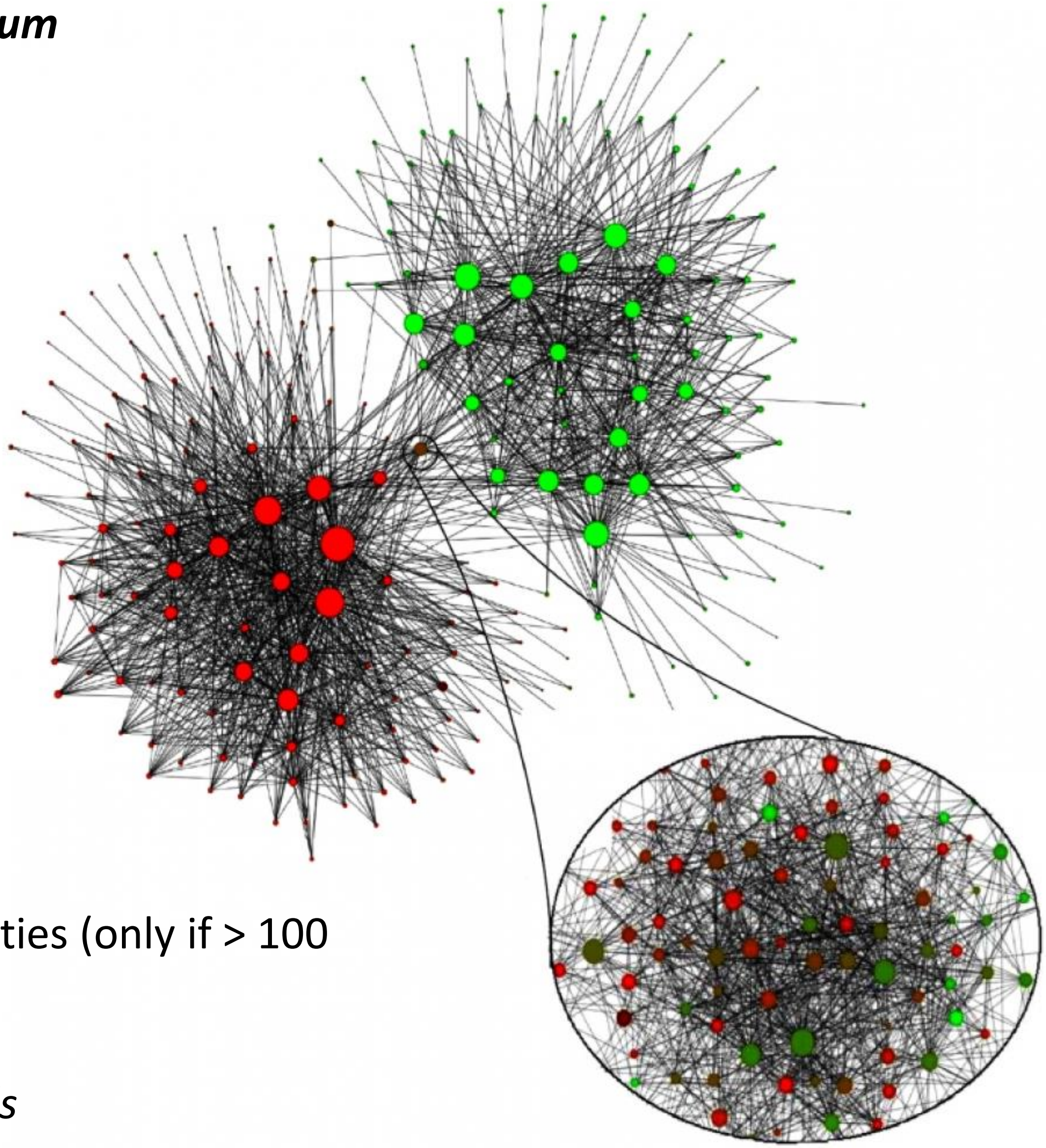
Why? (some applications)

- *Knowledge discovery*
- Groups based on common interests, behavior, etc (e.g., Canadians who call USA, readings tastes, etc)
  - *Recommendations, marketing*
- *Collective behavior* observable at the group, not the individual level, local view is noisy and ad hoc
- *Classification of the nodes* by identifying modules and their boundaries
- To improve *performance:* partition a large graph into many machines, assigning web clients to web servers, routing in ad hoc networks, etc
- Summary, *visual representation* of the graph

Example: communities in Belgium

59% Flemish, speaking Dutch 40% Walloons speaking French

Community structure in Belgium



2 million mobile phone users

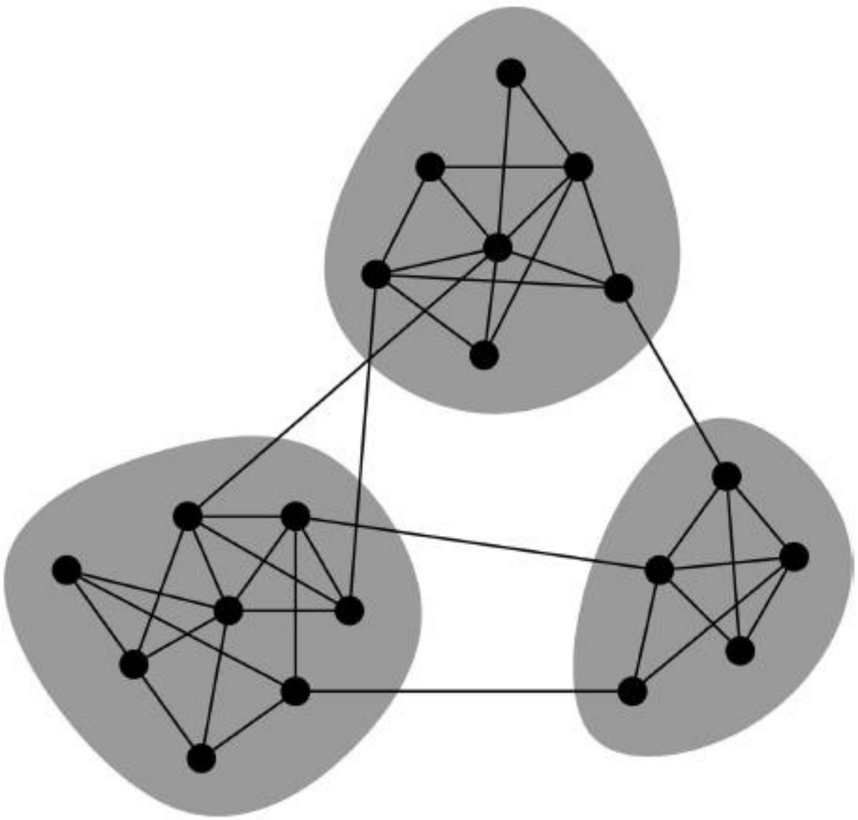
Nodes correspond to communities (only if > 100 members)

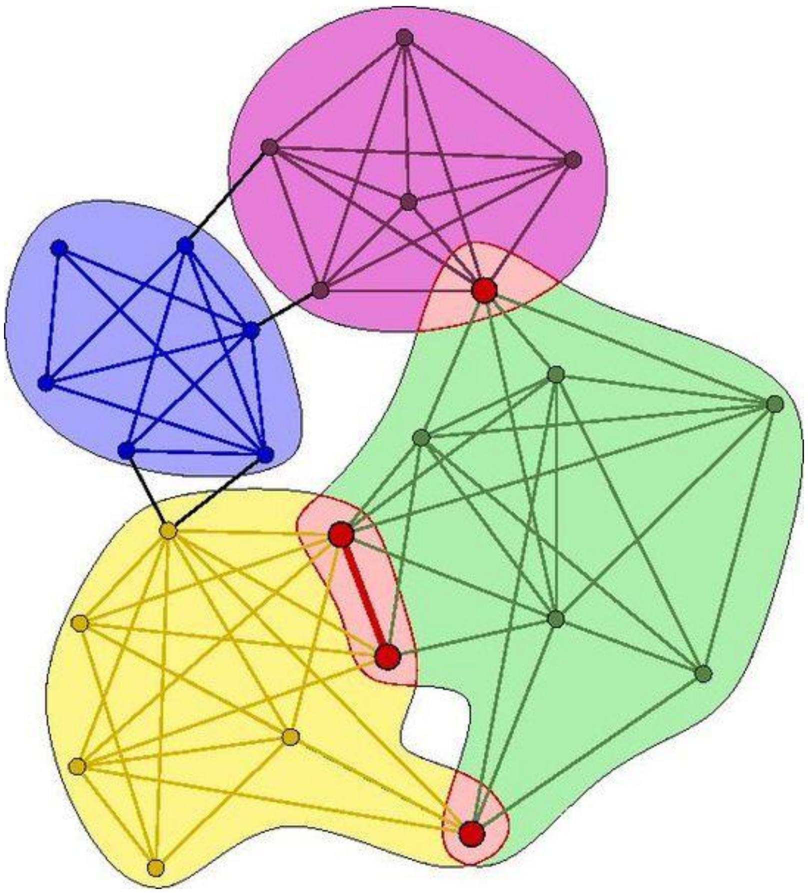
Red French, Green Dutch

Connecting community *Brussels*

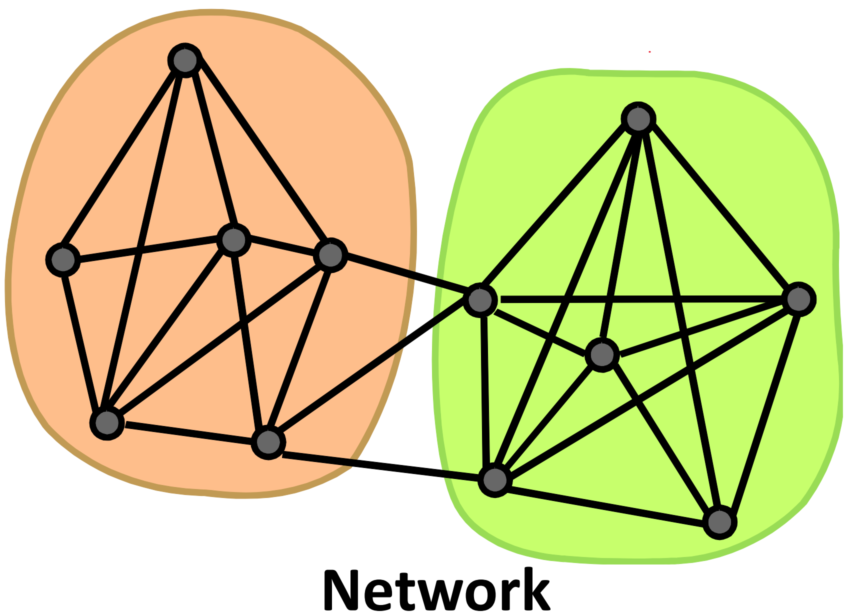
Community Types

Non-overlapping vs. overlapping communities

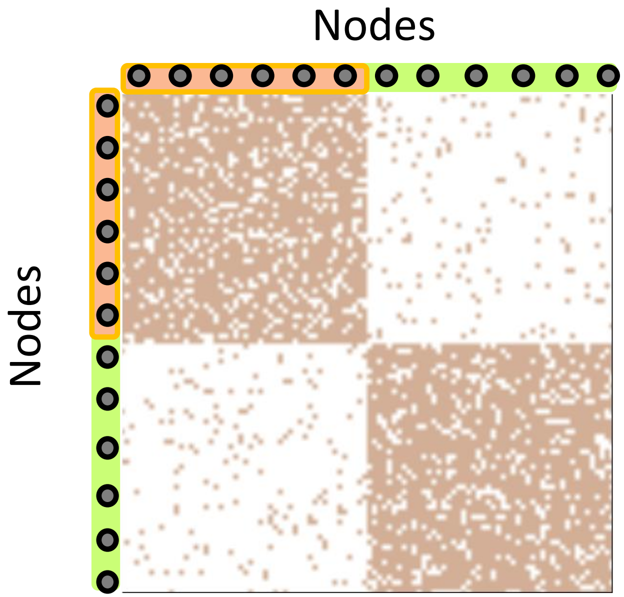




Non-overlapping Communities

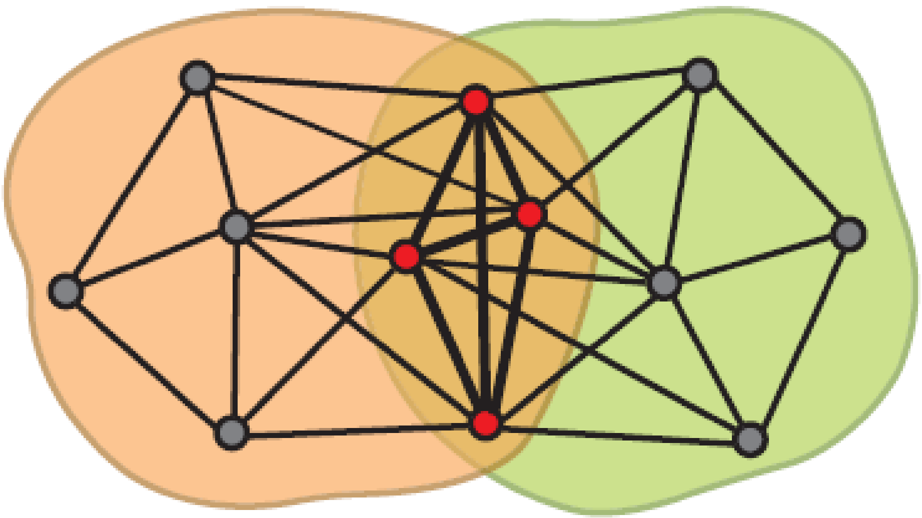


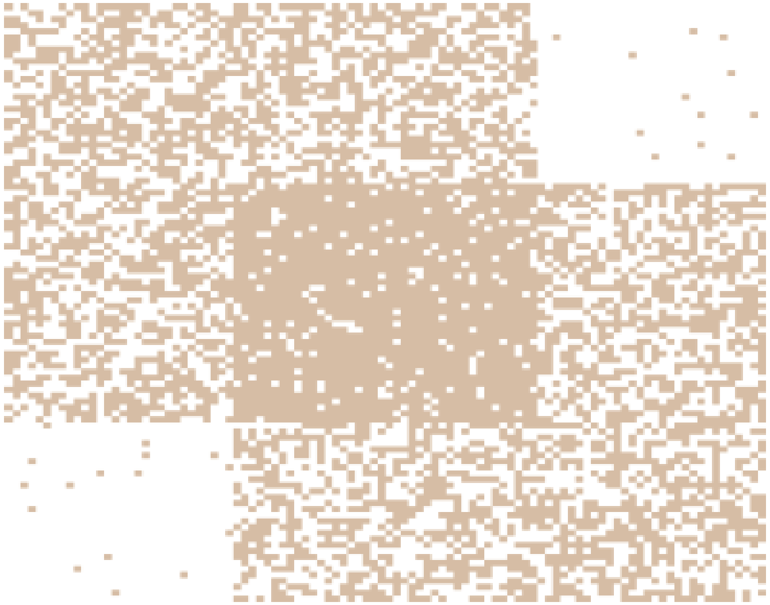
Adjacency matrix



Overlapping Communities

What is the structure of community overlaps:
Edge density in the overlaps is higher!

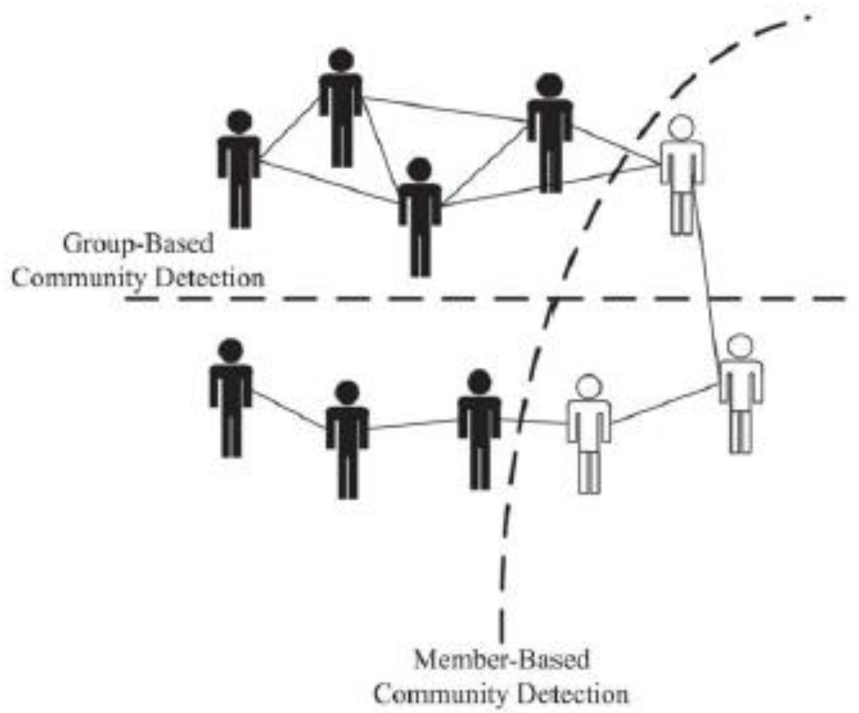




Communities as “tiles”

Community Types

Member-based (local) vs. group-based



Community Detection

Given a graph $G(V, E)$, find subsets C_i of V, such that $\bigcup_i C_i \subseteq$ V

Assumptions

- Undirected graphs
- Edges may have
  - weights, (easily extended)
  - labels
  - content or attributes shared by individuals (in the same location, of the same gender, etc)
- Nodes may have labels, attributed, or labeled graphs

Multipartite graphs – e.g., affiliation networks, citation networks, customers-products: reduced to unipartited projections of each vertex class

Hardness

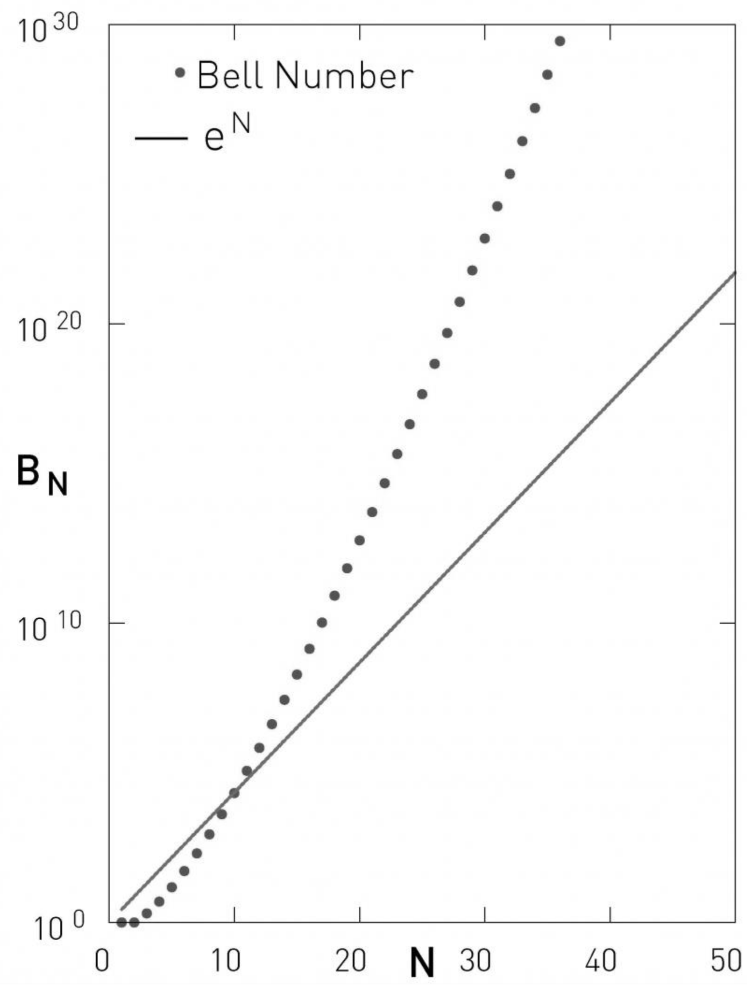
Bell Number

Number of all possible partitions of *N* nodes

For example, $B_3 = 5$

$$B_N = \frac{1}{e} \sum_{j=0}^{\infty} \frac{j^N}{j!}$$

For *N* = 50, 1040 partitions



Community Detection

We will see three approaches

- Node *degree* (familiarity)
  - Cliques
  - Density (next lecture)
- *Similarity*
  - Cluster
- Node *reachability*
  - Betweeneness

Outline

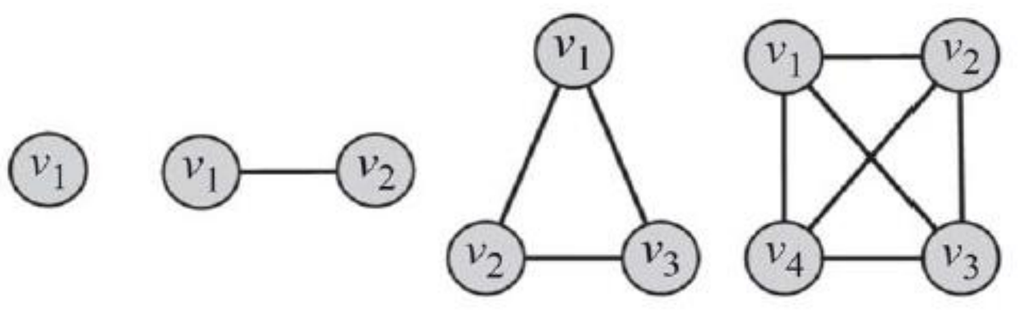
PART I

1. Introduction: what, why, types?
2. Cliques
3. Background: cluster analysis (node/edge similarity)
4. Hierarchical clustering (betweenness)
5. Modularity

Cliques (degree similarity)

Clique: a maximum *complete subgraph* in which all pairs of vertices are connected by an edge.

A *clique of size k* is a subgraph of ***k*** vertices where the degree of all vertices in the induced subgraph is ***k -1*** .



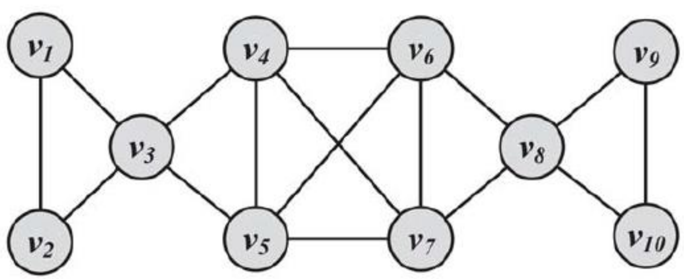
✓ Cliques vs complete graphs

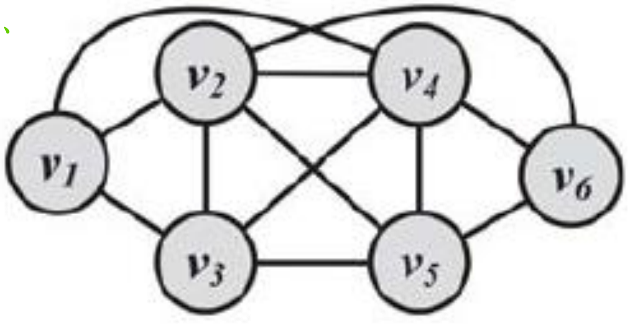
Cliques (degree similarity)

Search for:

- the *maximum clique:* the one with the largest number of vertices) or
- all *maximal cliques:* cliques that are not subgraphs of a larger clique; i.e., cannot be expanded further.

Both problems are NP-hard, as is verifying whether a graph contains a clique larger than size *k*.





Cliques

Algorithm 6.1 Brute-Force Clique Identification

Require: Adjacency Matrix A, Vertex v_x

return Maximal Clique C containing v_x
CliqueStack = $\{\{v_x\}\}$, Processed = $\{\}$;
while CliqueStack not empty **do**
  C=pop(CliqueStack); push(Processed,C);
  v_{last} = Last node added to C;
  $N(v_{last}) = \{v_i | A_{v_{last}, v_i} = 1\}$.
  for all $v_{temp} \in N(v_{last})$ **do**
    if $C \bigcup \{v_{temp}\}$ is a clique **then**
      push(CliqueStack, $C \bigcup \{v_{temp}\}$);
    end if
  end for
end while
Return the largest clique from Processed

/* Check all neighbors of last node sequentially
if connected with all members in the clique
new clique -> push */

Enumerate all cliques (in alphabetical order)

Checks all permutations!

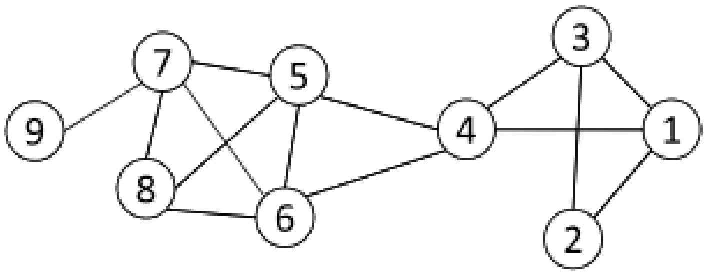
For (complete graph) 100 vertices, 2^{99}- 1 different cliques

Cliques

Pruning

- Prune all vertices (and incident edges) with degrees less than *k* - 1.
- Effective due to the power-law distribution of vertex degrees

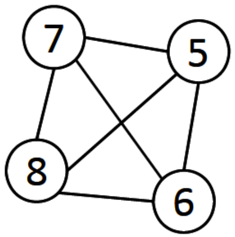
Example. to find a clique ≥ 4, remove all nodes with degree $\leq (4-1)-1=2$



Remove nodes 2 and 9

Remove nodes 1 and 3

Remove node 4



Relaxing Cliques

Exact cliques are *rarely observed* in real networks.

E.g., a clique of 1,000 vertices has (999x1000)/2 = 499,500 edges.

- A single edge removal results in a subgraph that is no longer a clique.
- That represents less than 0.0002% of the edges

Relaxing Cliques I

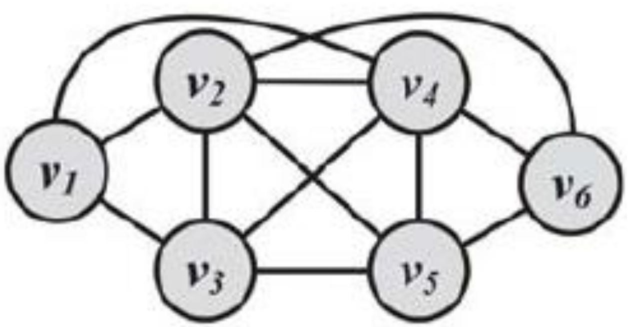
All vertices have *a minimum degree* but not necessarily k -1

k-plex

For a set of vertices V_0, for all u, $d_u \geq |V_0| - k$
where d_u is the degree of v in the induced subgraph

What is k for a clique?

Maximal



1-plex : $\{v_2, v_3, v_4, v_5\}$

2-plex : $\{v_1, v_2, v_3, v_4, v_5\}, \{v_2, v_3, v_4, v_5, v_6\}$

3-plex : $\{v_1, v_2, v_3, v_4, v_5, v_6\}$

k-core

a maximal connected subgraph in which all vertices have degree at least k

Relaxing Cliques II

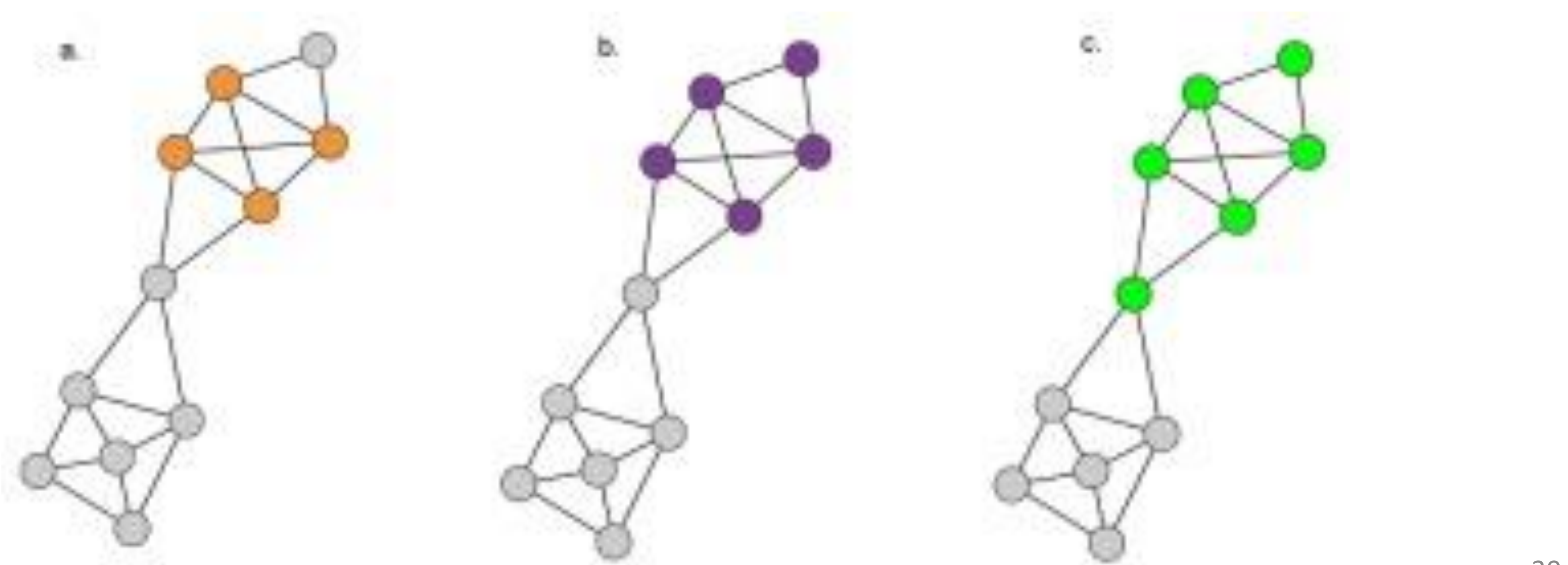
Clique $\forall\, i \in C, d_i^{int} = |C| - 1$

Strong community $\forall\, i \in C, d_i^{int} > d_i^{ext}$

Weak community $\sum_{i \in C} d_i^{int} > \sum_{i \in C} d_i^{ext}$

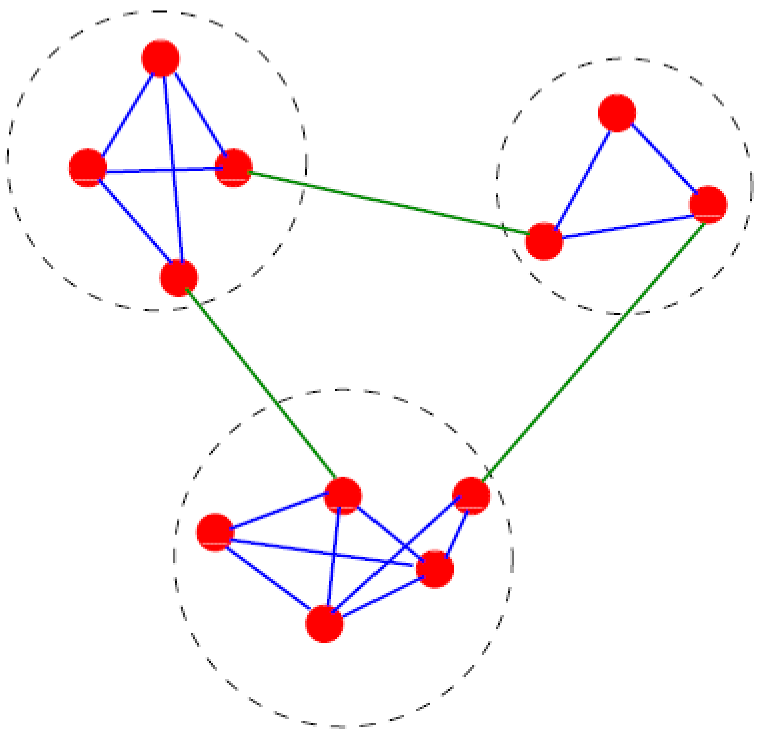
d_i^{int} degree (#edges) of node *i* with nodes inside *C*

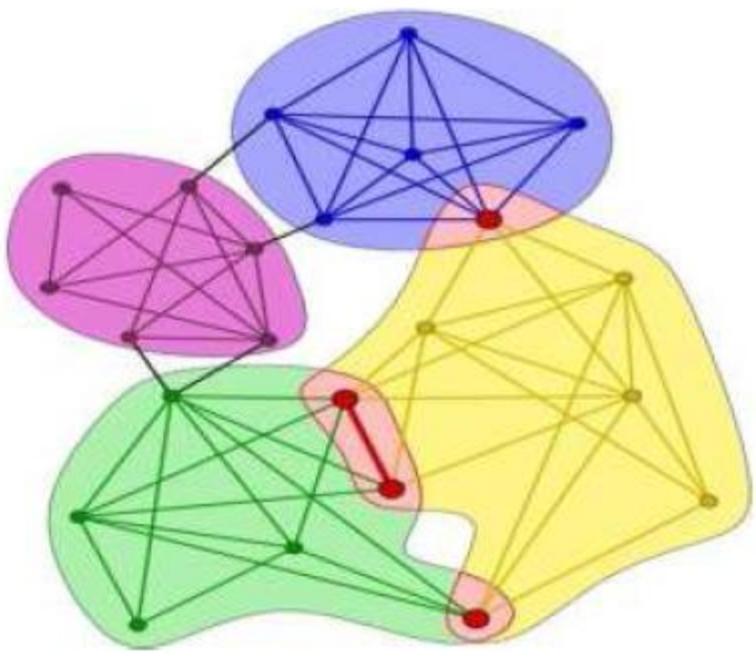
d_i^{ext} degree (#edges) of node *i* with nodes outside *C*



Clique Percolation Method (CPM): Using cliques as seeds

Assumption: communities are formed from a set of cliques and edges that connect these cliques.





$k = 4$

Clique Percolation Method (CPM): Using cliques as seeds

Algorithm 6.2 Clique Percolation Method (CPM)

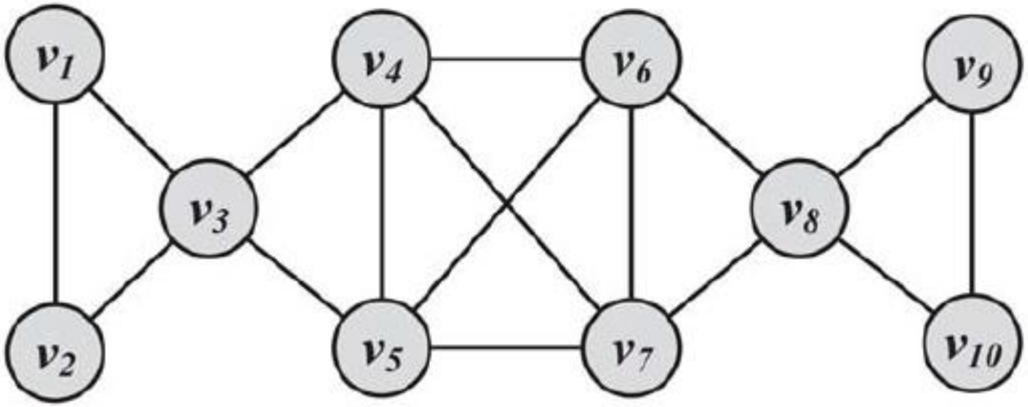
Require: parameter k
1: **return** Overlapping Communities
2: $Cliques_k$ = find all cliques of size k
3: Construct clique graph $G(V, E)$, where $|V| = |Cliques_k|$
4: $E = \{e_{ij} \mid$ clique i and clique j share $k - 1$ nodes$\}$
5: Return all connected components of G

---

1. Given k, find all cliques of size k.
2. Create graph (clique graph) where all cliques are vertices, and two cliques that share k - 1 vertices are connected via an edge.
3. Communities are the connected components of this graph.

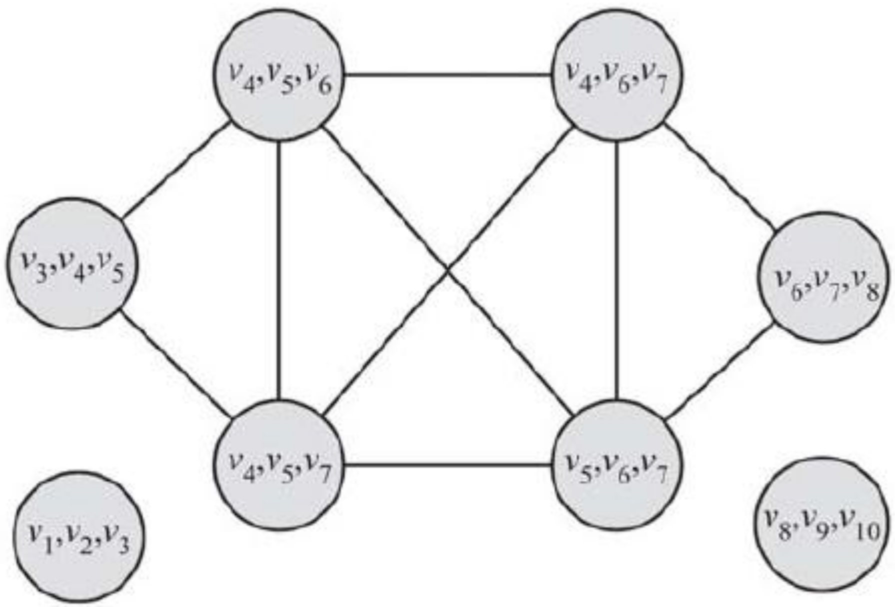
Clique Percolation Method (CPM): Using cliques as seeds

Input graph, let $k = 3$



Clique Percolation Method (CPM): Using cliques as seeds

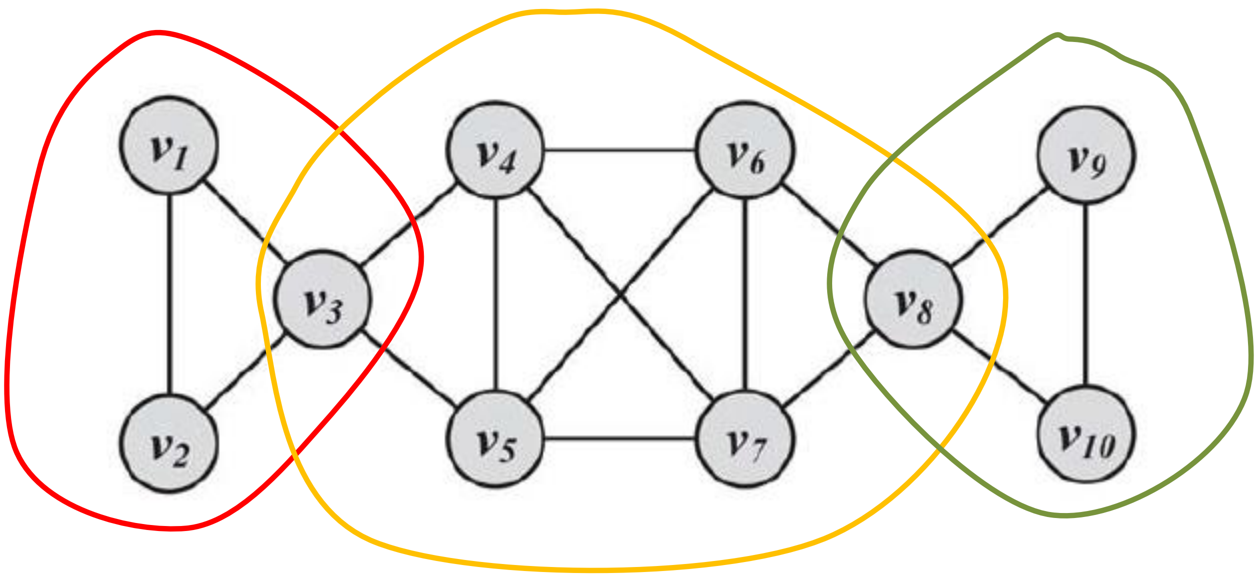
Clique graph for k = 3



(v1, v2, ,v3), (v8, v9, v10), and (v3, v4, v5, v6, v7, v8)

Clique Percolation Method (CPM): Using cliques as seeds

Result



(v1, v2, ,v3), (v8, v9, v10), and (v3, v4, v5, v6, v7, v8)

Note: the example protein network was detected using a CPM algorithm

Clique Percolation Method (CPM)

- By construction, *overlapping* communities
- Instead of $k = 3$, maximal cliques
- Theoretical complexity grows exponential with size, but *efficient on sparse graphs*

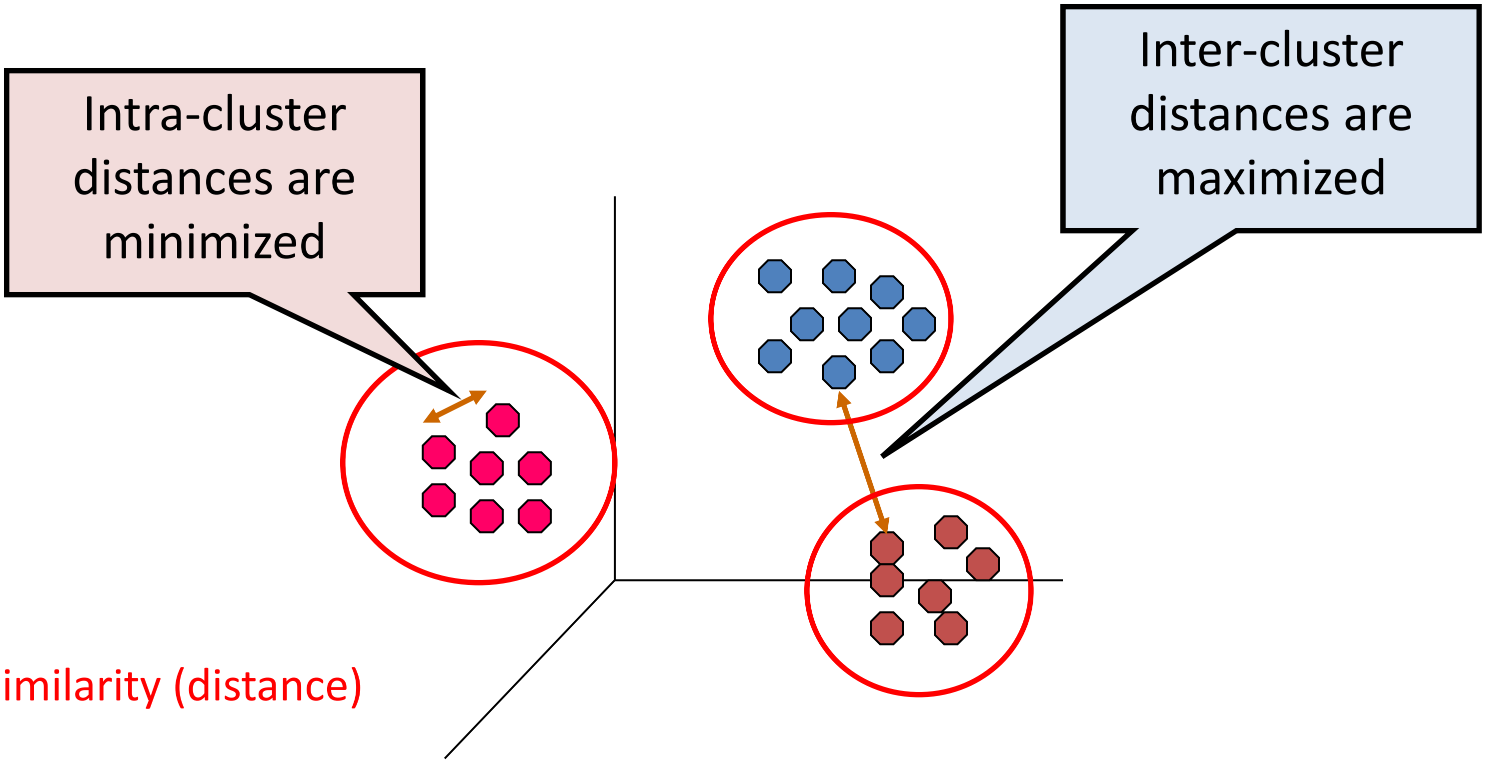
Outline

PART I

1. Introduction: what, why, types?
2. Cliques
3. Background: cluster analysis (vertex/edge similarity)
4. Hierarchical clustering (betweenness)
5. Modularity

What is Cluster Analysis?

Finding groups of objects such that the objects in a group are similar (or related) to one another and different from (or unrelated to) the objects in other groups

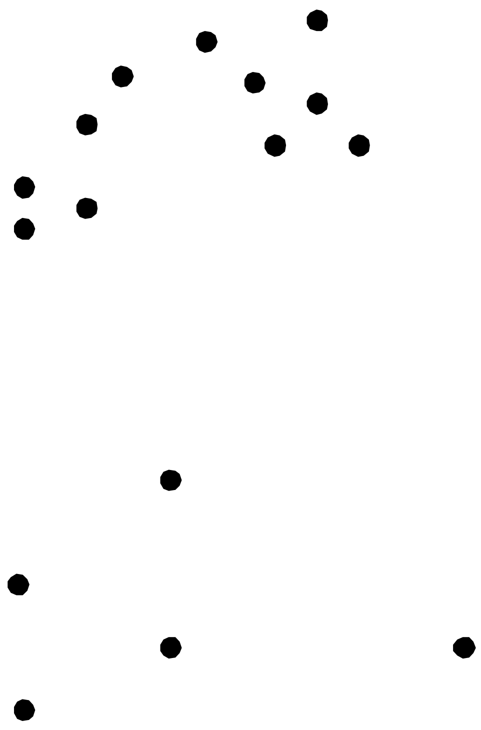


Based on similarity (distance)

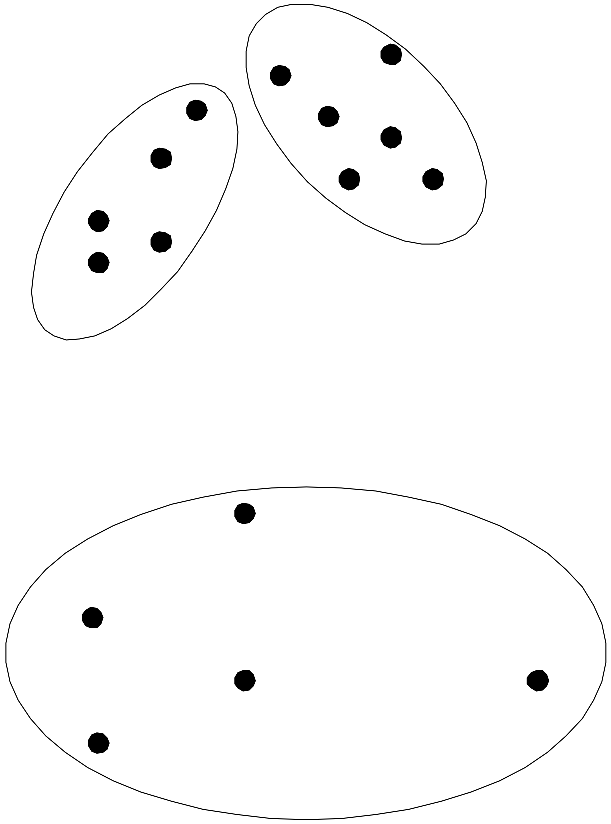
Types of Clustering

- Important distinction between hierarchical and partitional sets of clusters
- Partitional Clustering
  - Division of data objects into subsets (clusters)
  - Assumes that the *number of clusters is given*
- Hierarchical clustering
  - A set of *nested clusters* organized as a hierarchical tree

Partitional Clustering



Original Points



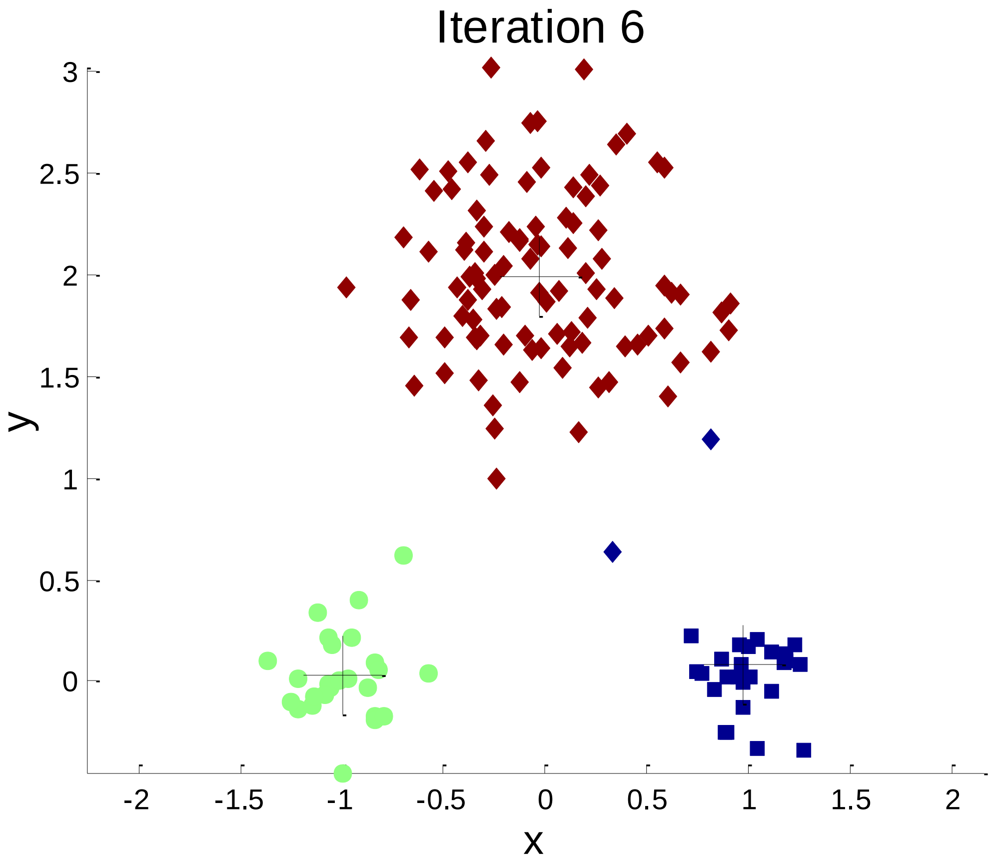
A Partitional Clustering

Example Partitioning: K-means Clustering

```
Select K points as the initial centroids.
repeat
    Form K clusters by assigning all points to the closest centroid.
    Recompute the centroid of each cluster.
until The centroids don't change
```

- Input: Number of clusters, K
- Each cluster is associated with a *centroid* (center point)
- Each point is assigned to the cluster with the closest centroid

Example



K-means Clustering

- Initial centroids are often chosen randomly.
  - Clusters produced vary from one run to another.
- The centroid is (typically) the mean of the points in the cluster.
- Closeness - Similarity is measured by Euclidean distance, cosine similarity, correlation, etc.
- K-means *will converge* for common similarity measures mentioned above.
  - Most of the convergence happens in the first few iterations.
  - Often the stopping condition is changed to 'Until relatively few points change clusters'
- Complexity is O(n * K * I * d)
  - n = number of points, K = number of clusters,
    I = number of iterations, d = number of attributes (cost of computing similarity)

K-means Clusters

- Most common measure is Sum of Squared Error (SSE)
  - For each point, the error is the distance to the nearest cluster
  - To get SSE, we square these errors and sum them.

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist^2(m_i, x)$$

  - *x* is a data point in cluster C_i and m_i is the representative point for cluster C_i
    - can show that m_i corresponds to the center (mean) of the cluster
  - Given two clusters, we can choose the one with the smallest error
  - One easy way to reduce SSE is to increase K, the number of clusters
    - A good clustering with smaller K can have a lower SSE than a poor clustering with higher K

Vertex similarity

- Define similarity between two vertices
- Place similar vertices in the same cluster

- Use traditional *cluster analysis*

Vertex similarity

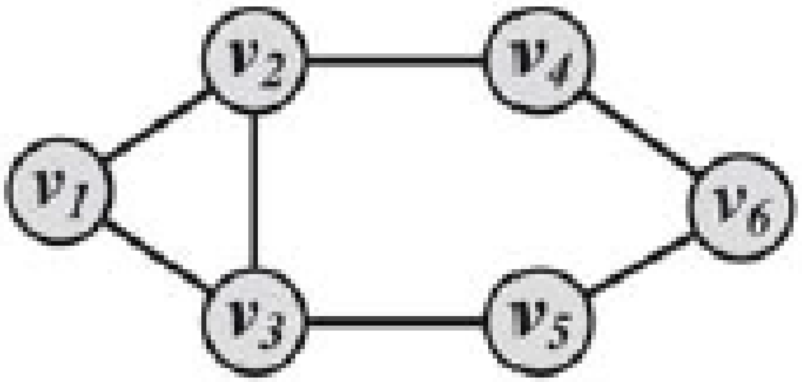
- Structural equivalence: based on the overlap between their neighborhoods

$$\sigma(v_i, v_j) = |N(v_i) \cap N(v_j)|$$

- Normalized to [0, 1], e.g.,

$$\sigma_{\text{Jaccard}}(v_i, v_j) = \frac{|N(v_i) \cap N(v_j)|}{|N(v_i) \cup N(v_j)|}$$

Vertex similarity



$$\sigma_{\text{Jaccard}}(v_2, v_5) = \frac{|\{v_1, v_3, v_4\} \cap \{v_3, v_6\}|}{|\{v_1, v_3, v_4, v_6\}|} = 0.25$$

Other definitions of vertex similarity

Use the adjacency matrix A,

$$d_{ij} = \sqrt{\sum_{k \neq i,j} (A_{ik} - A_{jk})^2}$$

Common neighbors
(paths of length two)

We can also use A^2

Other definitions of vertex similarity

If we map vertices u, v to n-dimensional points A, B in the Euclidean space,

$$d^E_{AB} = \sum_{k=1}^{n} \sqrt{(a_k - b_k)^2}$$

$$d^M_{AB} = \sum_{k=1}^{n} |a_k - b_k|$$

$$d^\infty_{AB} = \max_{k \in [1,n]} |a_k - b_k|$$

$$\rho_{AB} = \arccos \frac{\mathbf{a} \cdot \mathbf{b}}{\sqrt{\sum_{k=1}^{n} a_k^2} \sqrt{\sum_{k=1}^{n} b_k^2}}$$

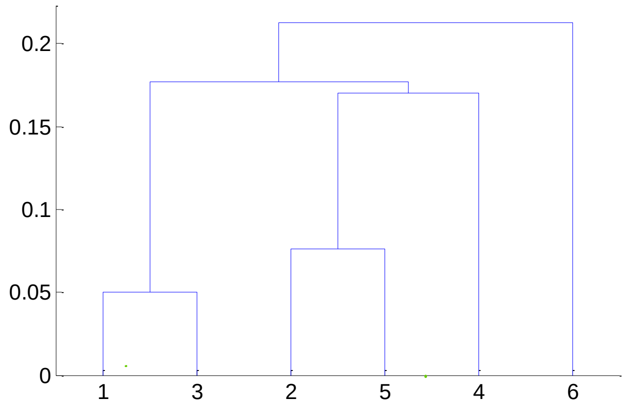
Other definitions of vertex similarity

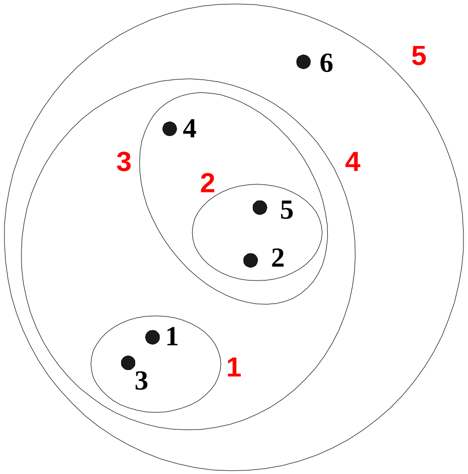
Many more – we shall revisit this issue when we talk about *graph embeddings*

Useful when there are *attributes* associated with nodes or edges to combine distances

Hierarchical Clustering

- Produces a set of nested clusters organized as a hierarchical tree
- Can be visualized as a dendrogram
  – A tree like diagram that records the sequences of merges or splits





Hierarchical Clustering

- Two main types of hierarchical clustering
  - Agglomerative:
    - Start with each node as an individual cluster (called singletons)
    - At each step, merge the closest pair of clusters until only one cluster (or k clusters) is left
  - Divisive:
    - Start with one, all-inclusive cluster = the whole graph
    - At each step, split a cluster until each cluster contains a single node (or there are k clusters)
- Traditional hierarchical algorithms use a similarity or distance matrix
  - Merge or split one cluster at a time

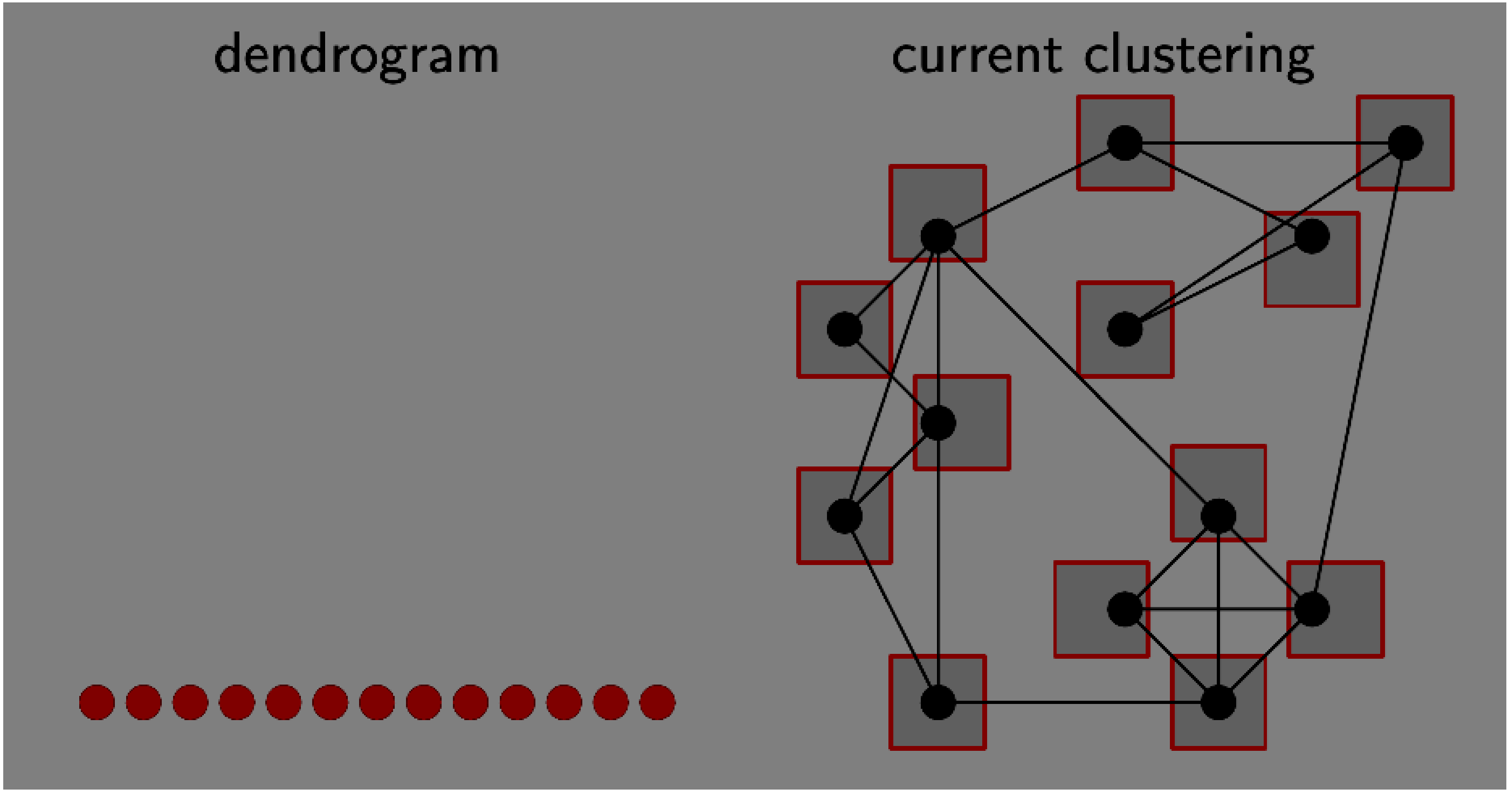
Agglomerative Clustering Algorithm

Popular hierarchical clustering technique

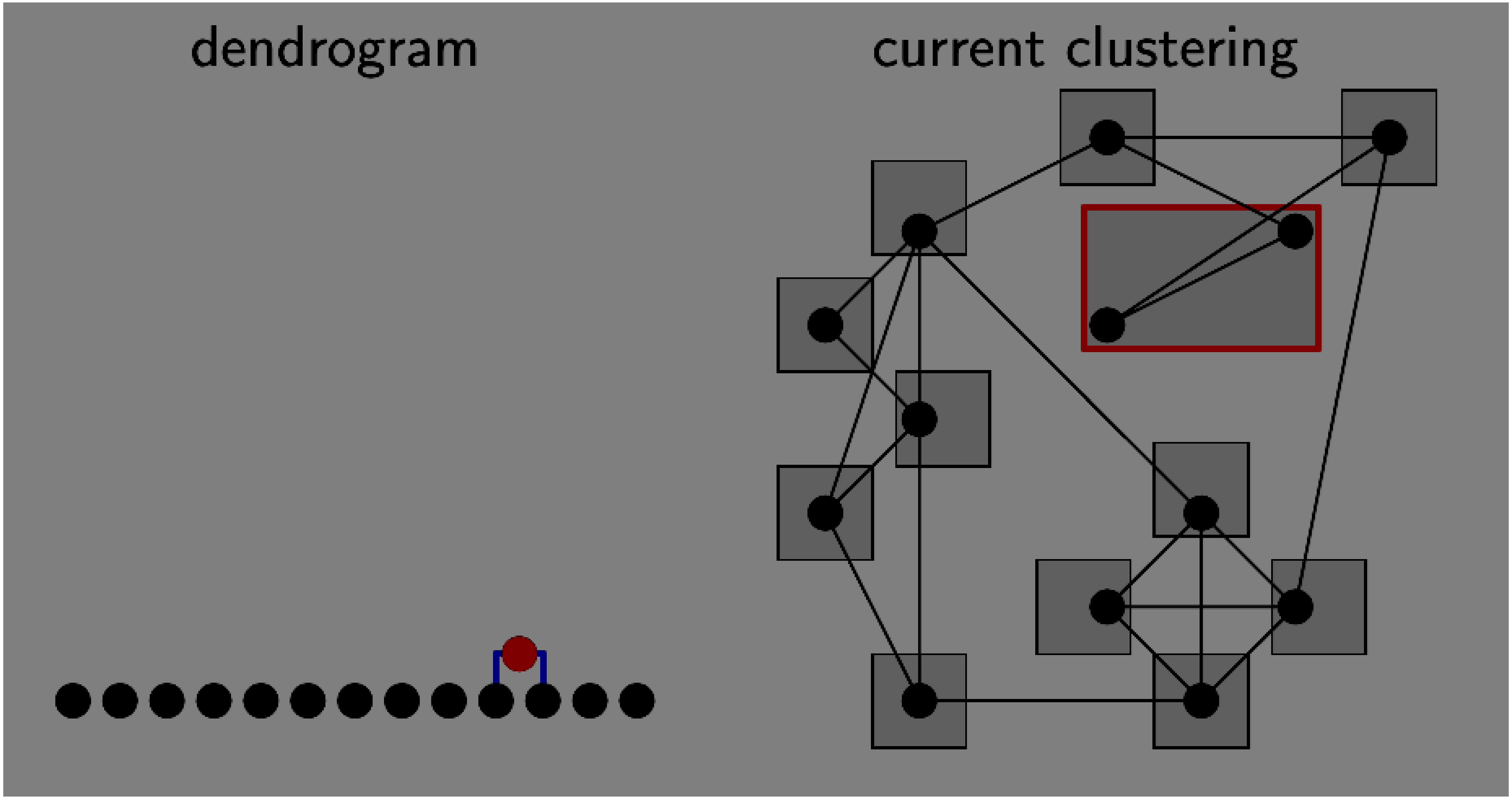
Basic algorithm is straightforward

1. [Compute the proximity matrix]
2. Let each node be a cluster
3. **Repeat**
4. Merge the *two closest clusters*
5. [Update the proximity matrix]
6. **Until** only a single cluster remains

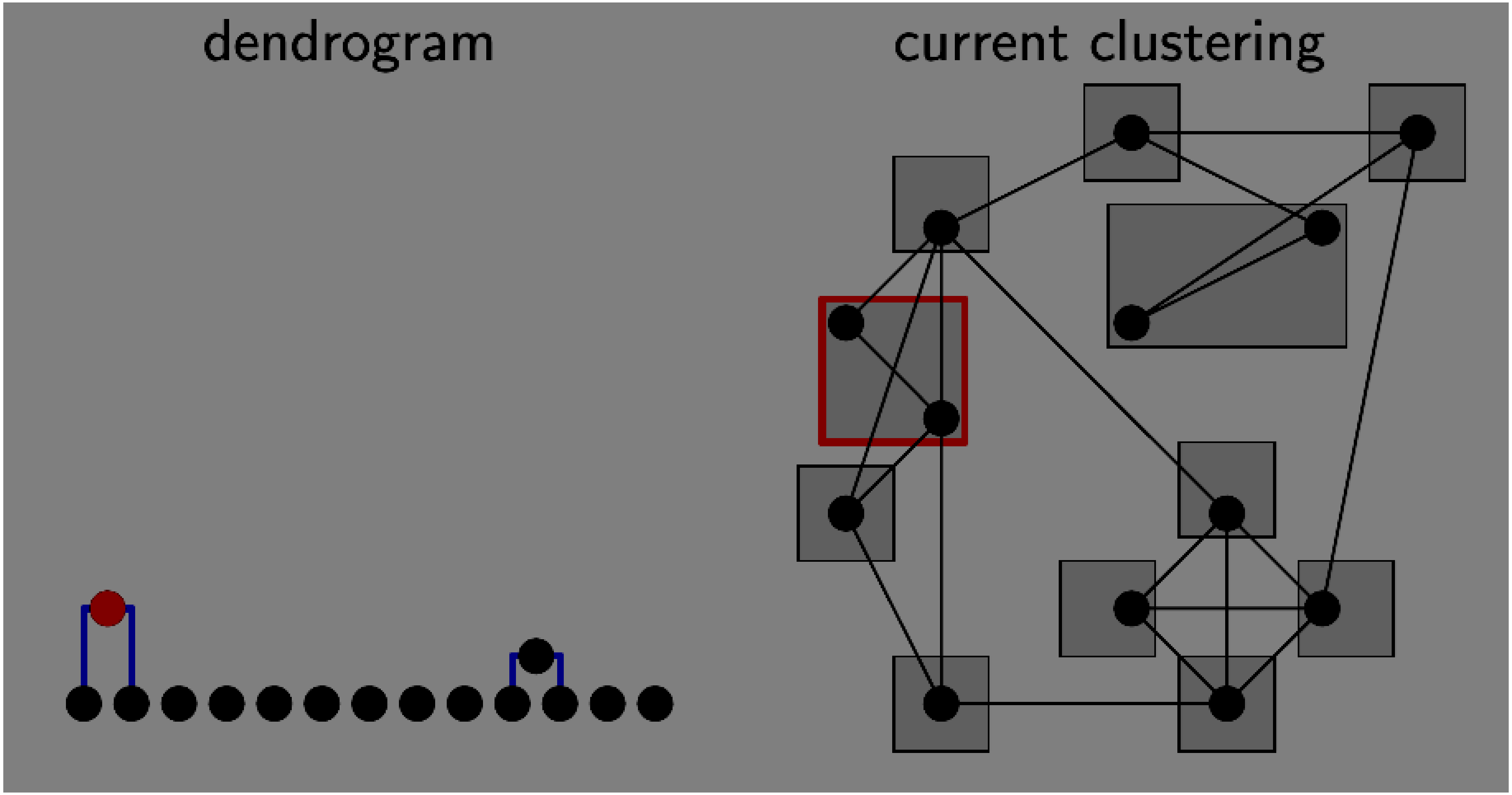
Agglomerative



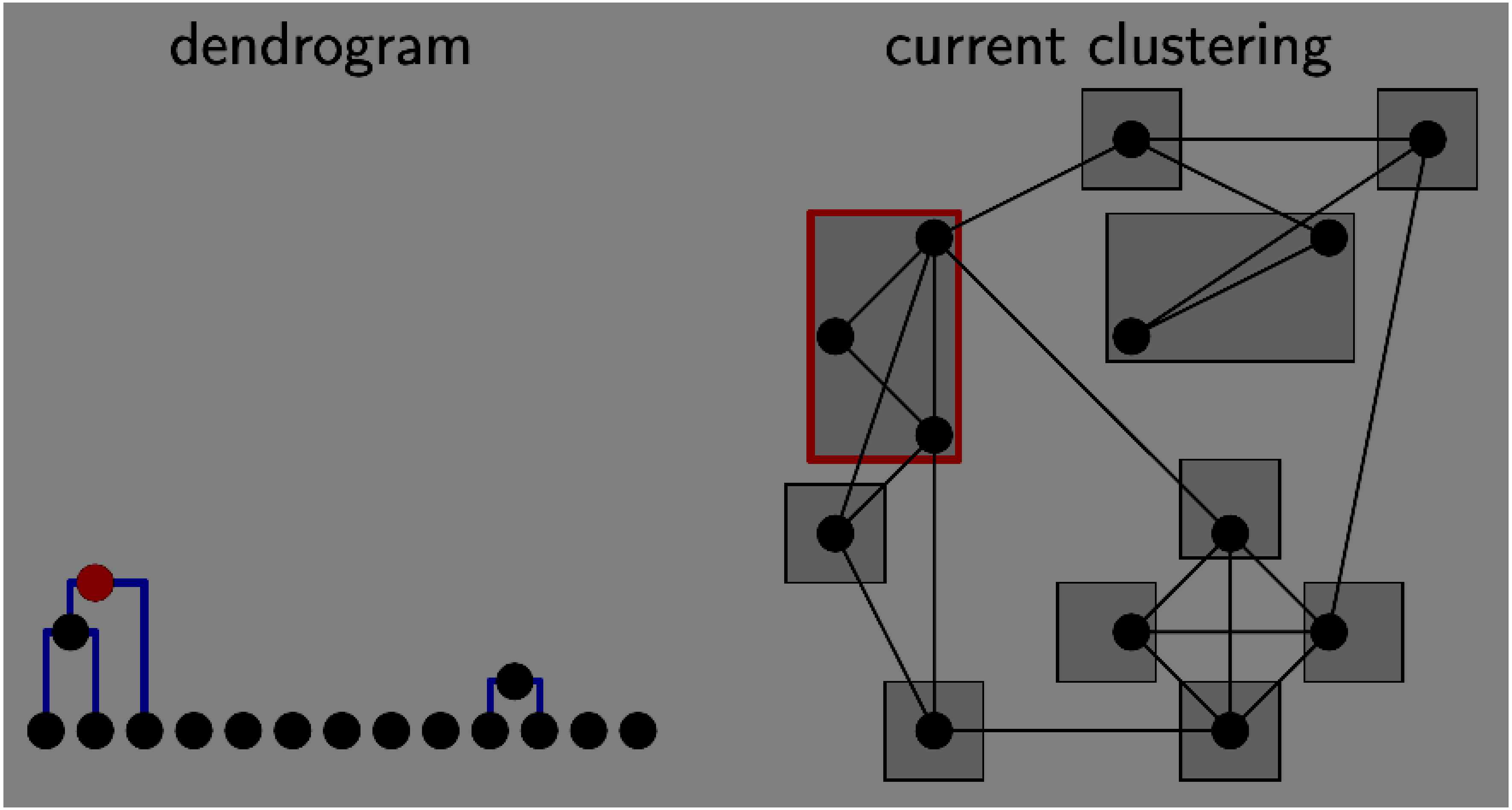
Agglomerative



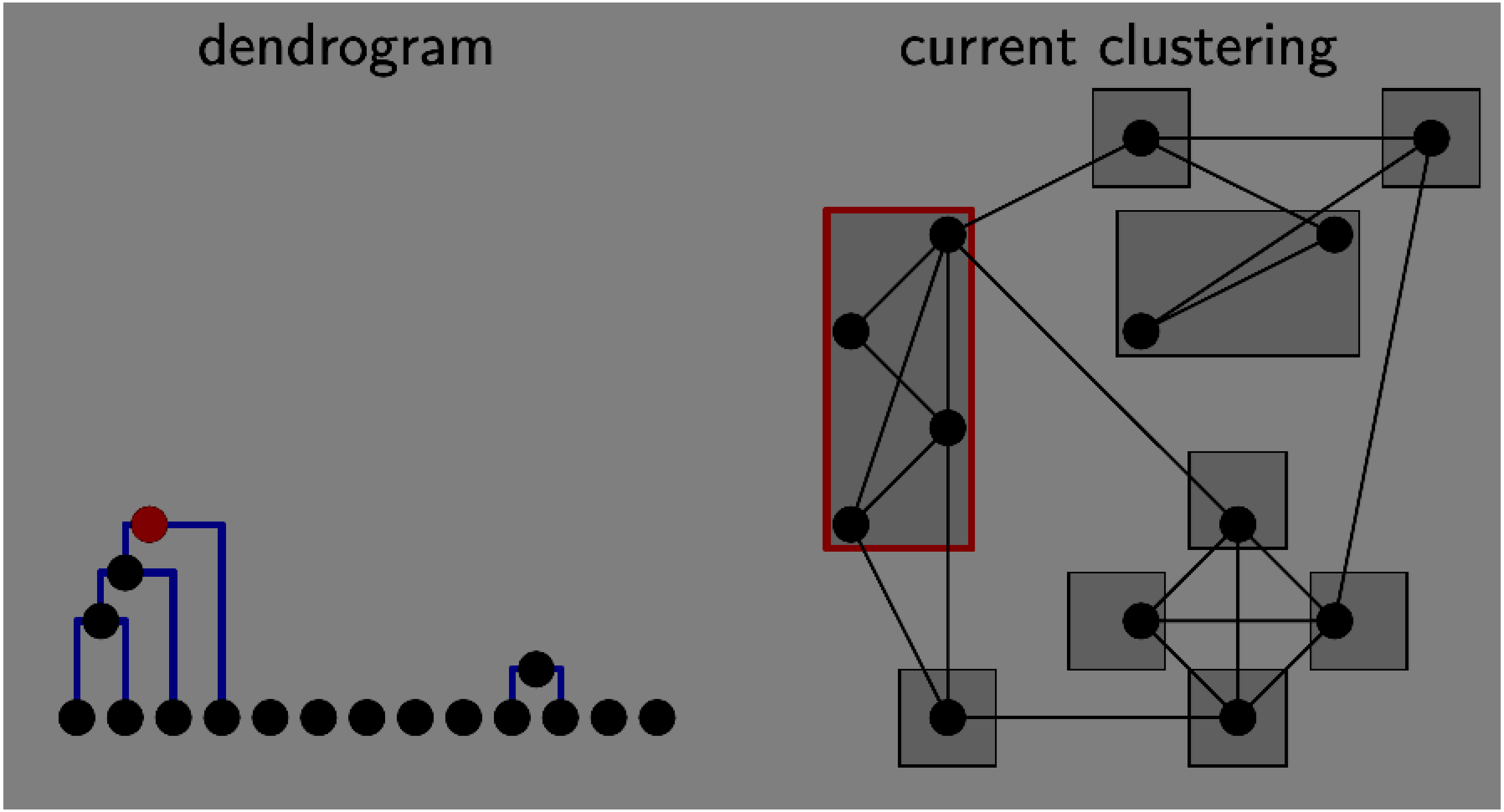
Agglomerative



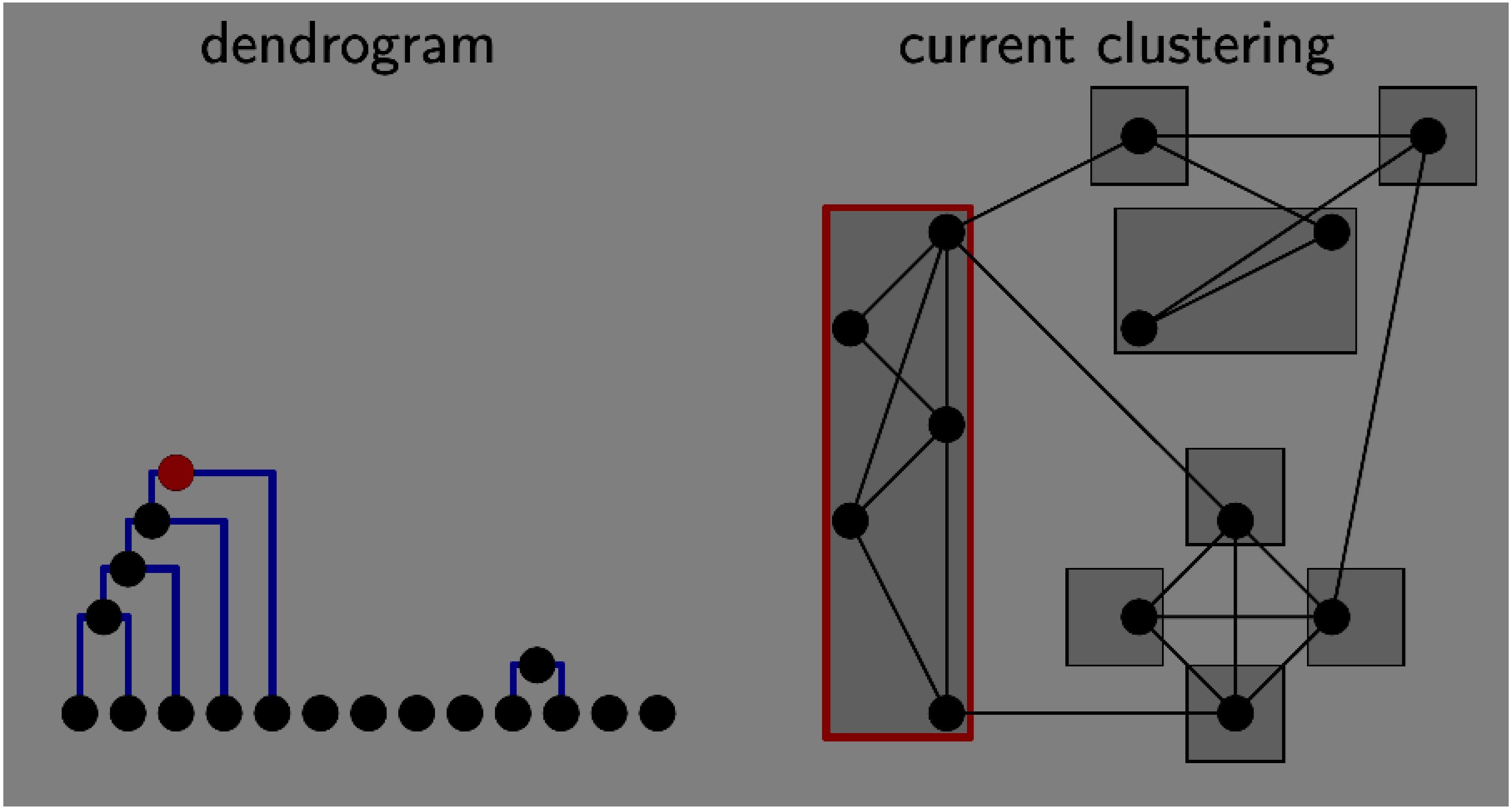
Agglomerative



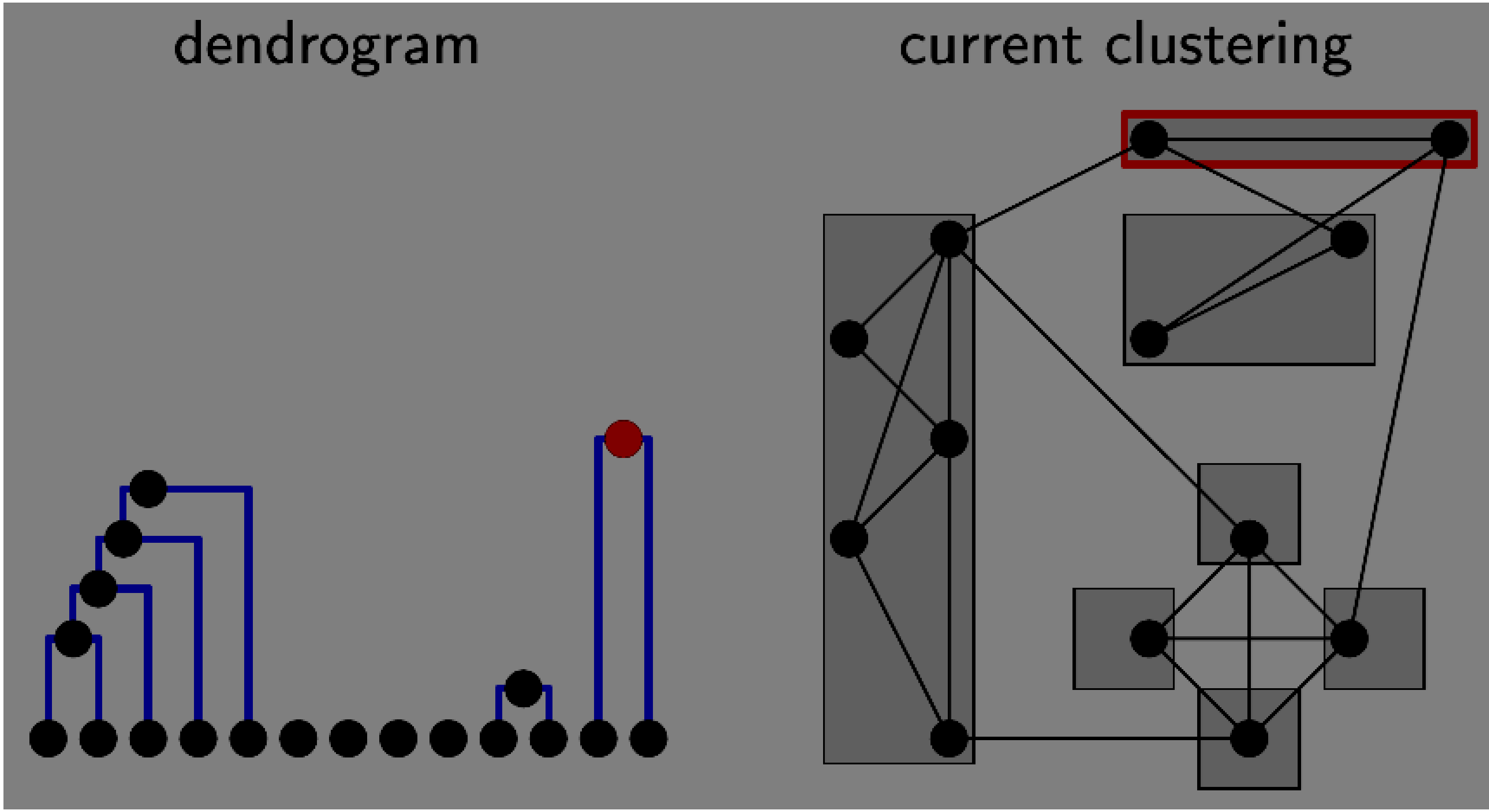
Agglomerative



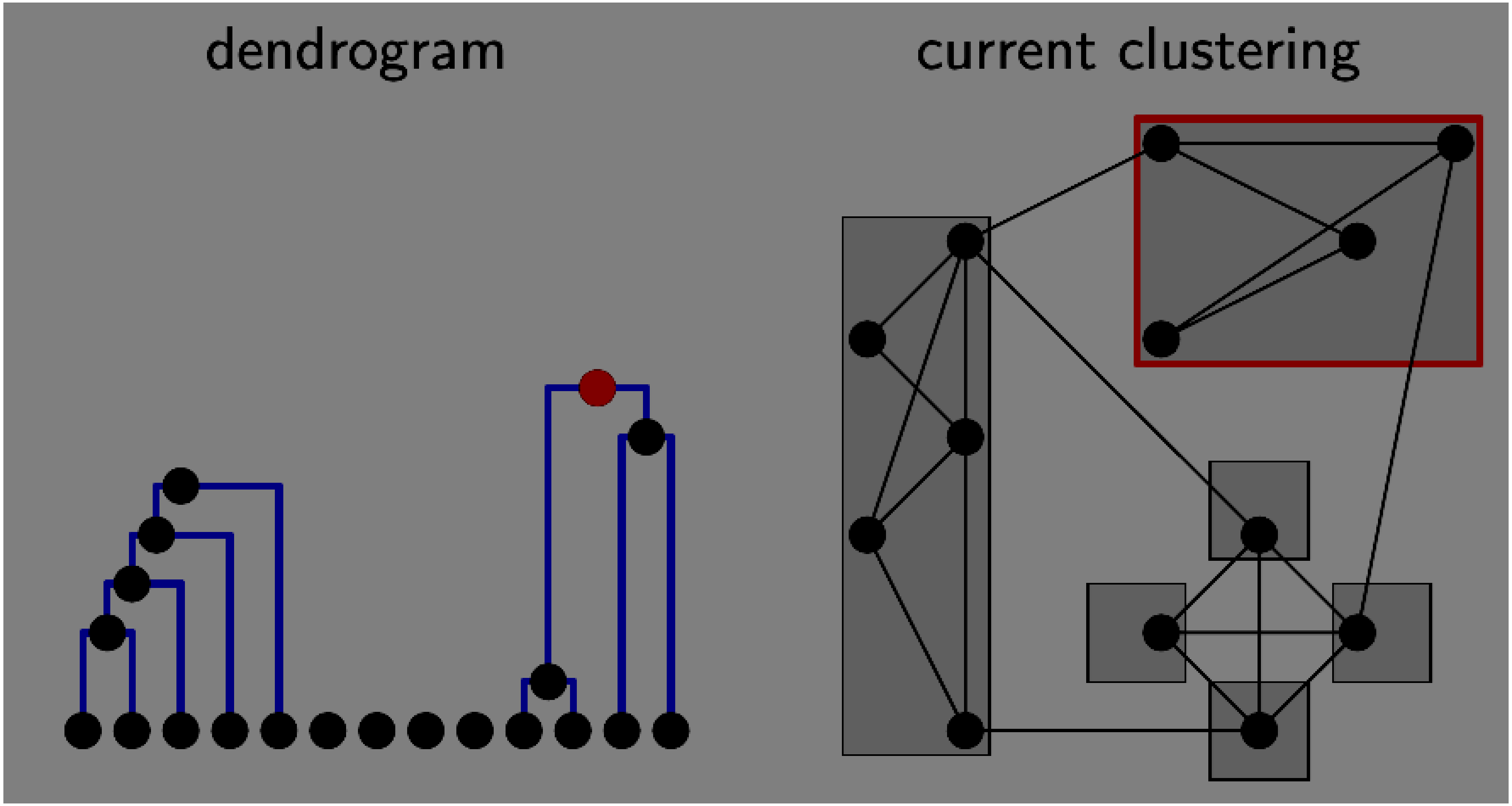
Agglomerative



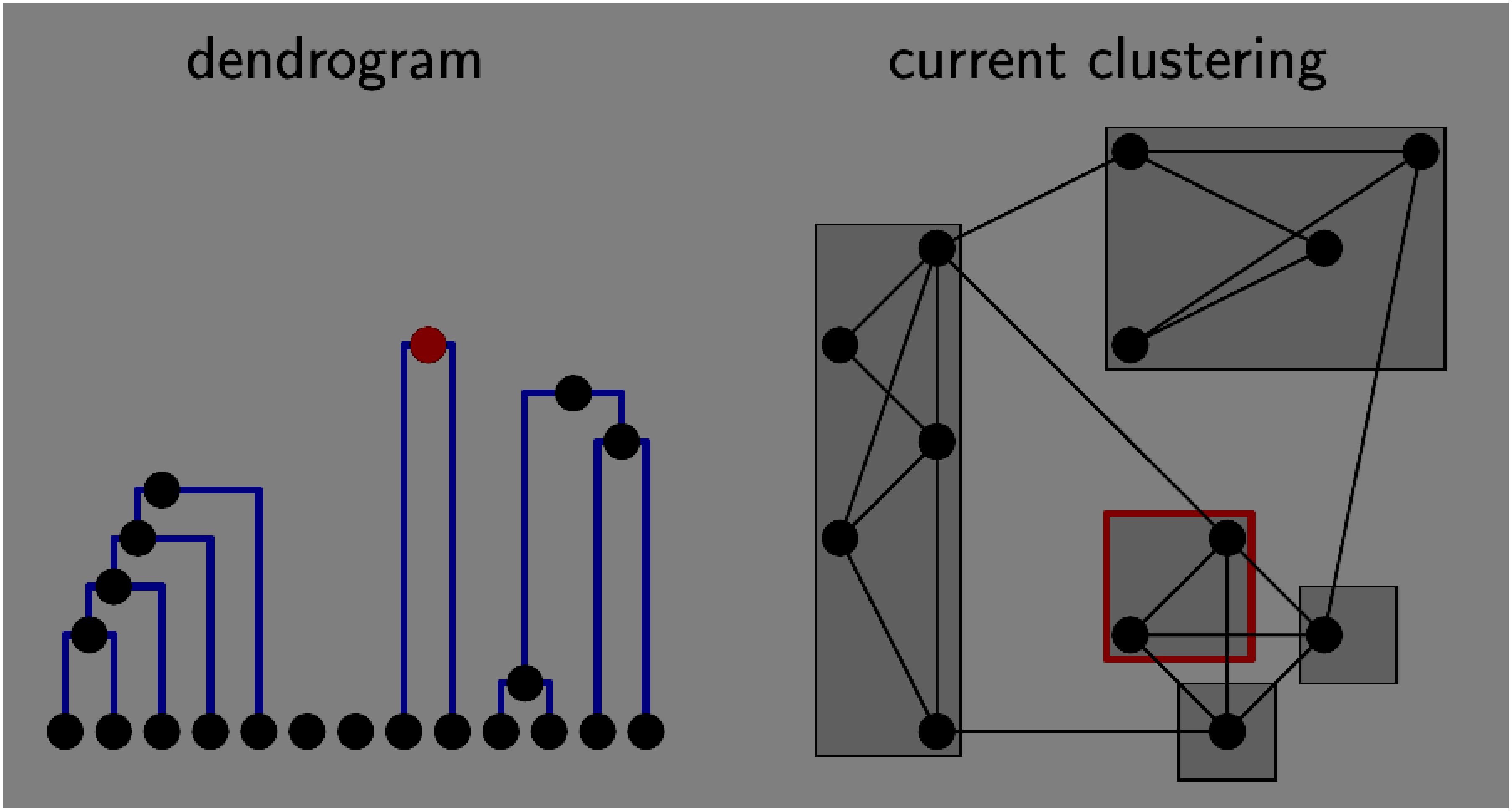
Agglomerative



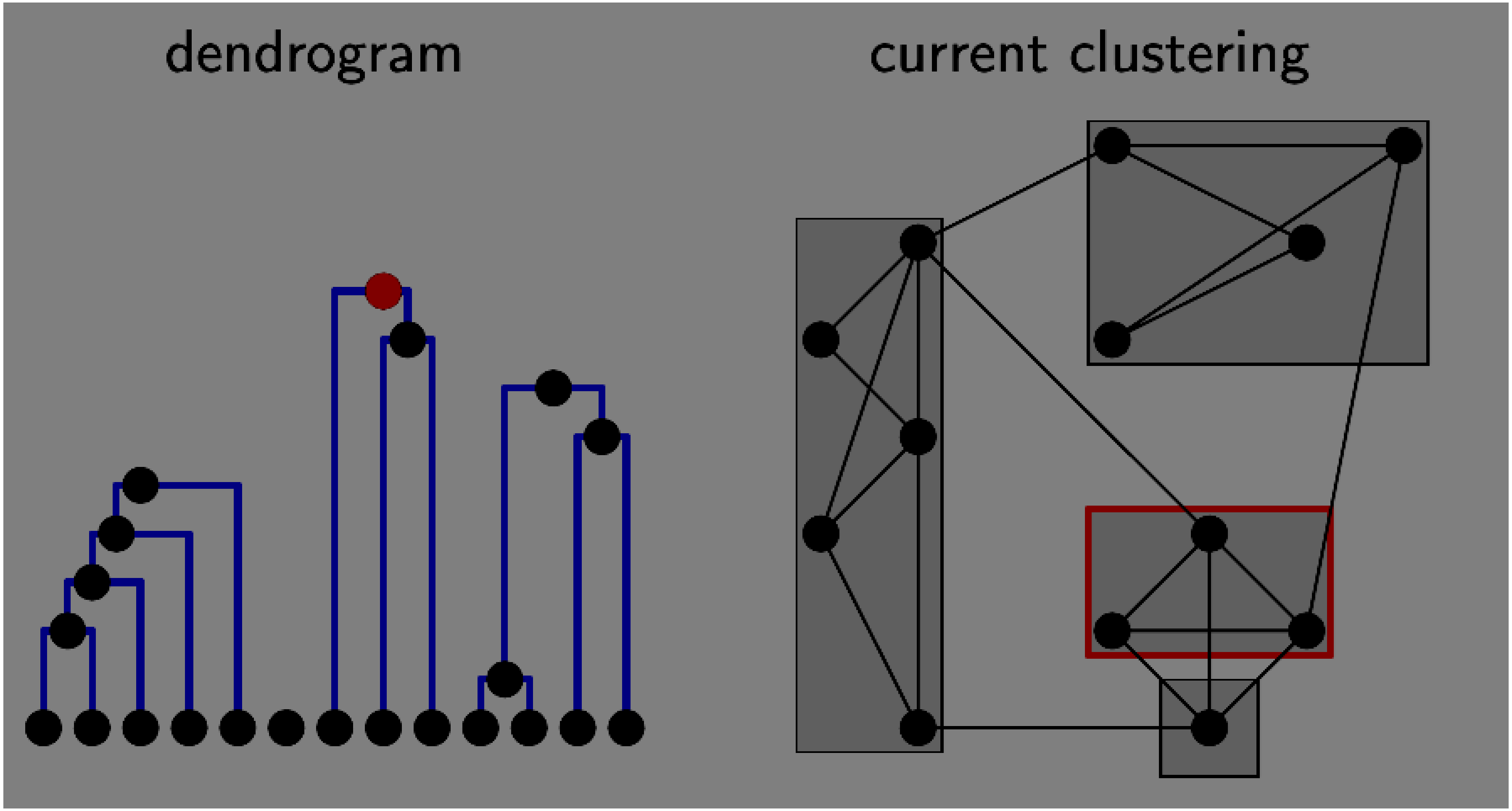
Agglomerative



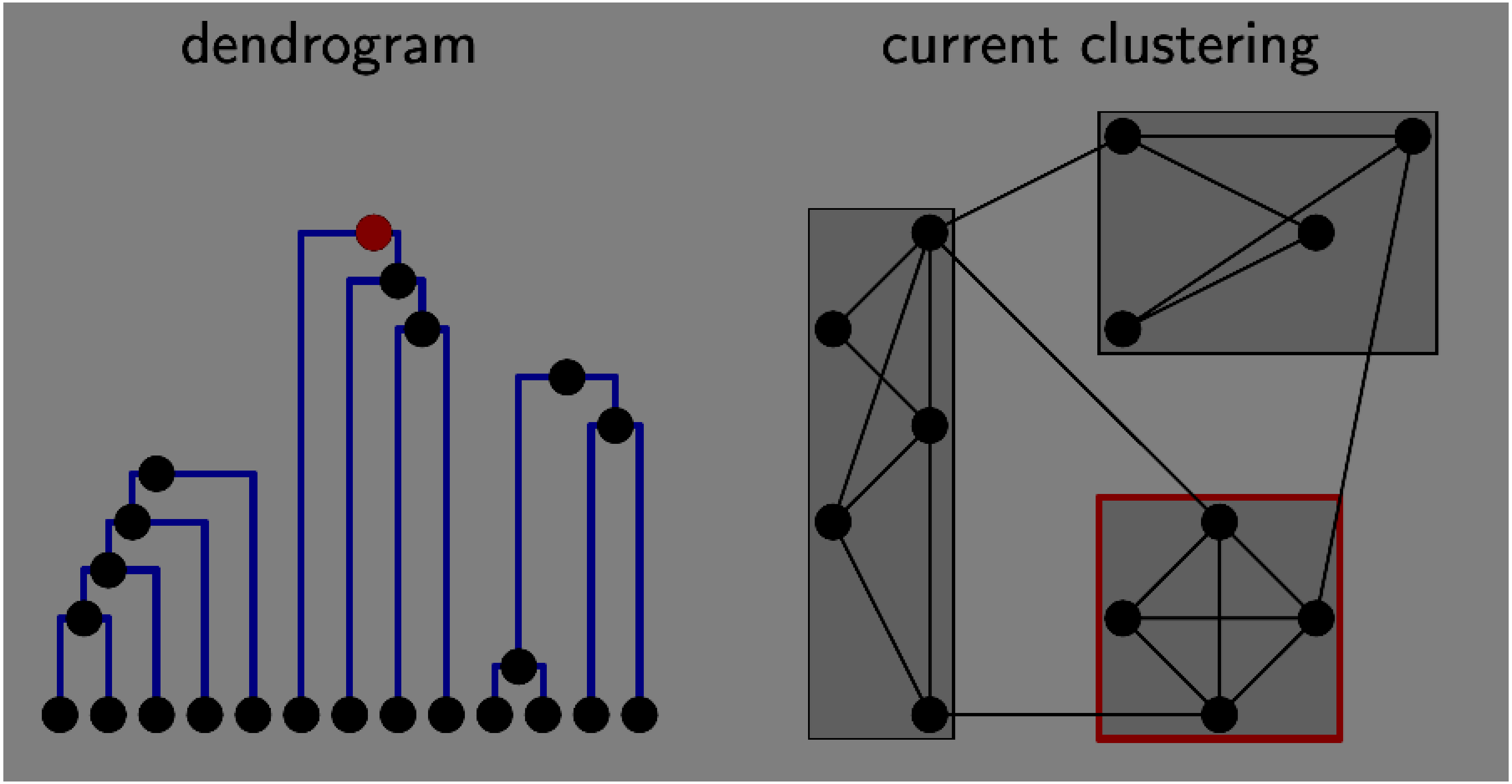
Agglomerative



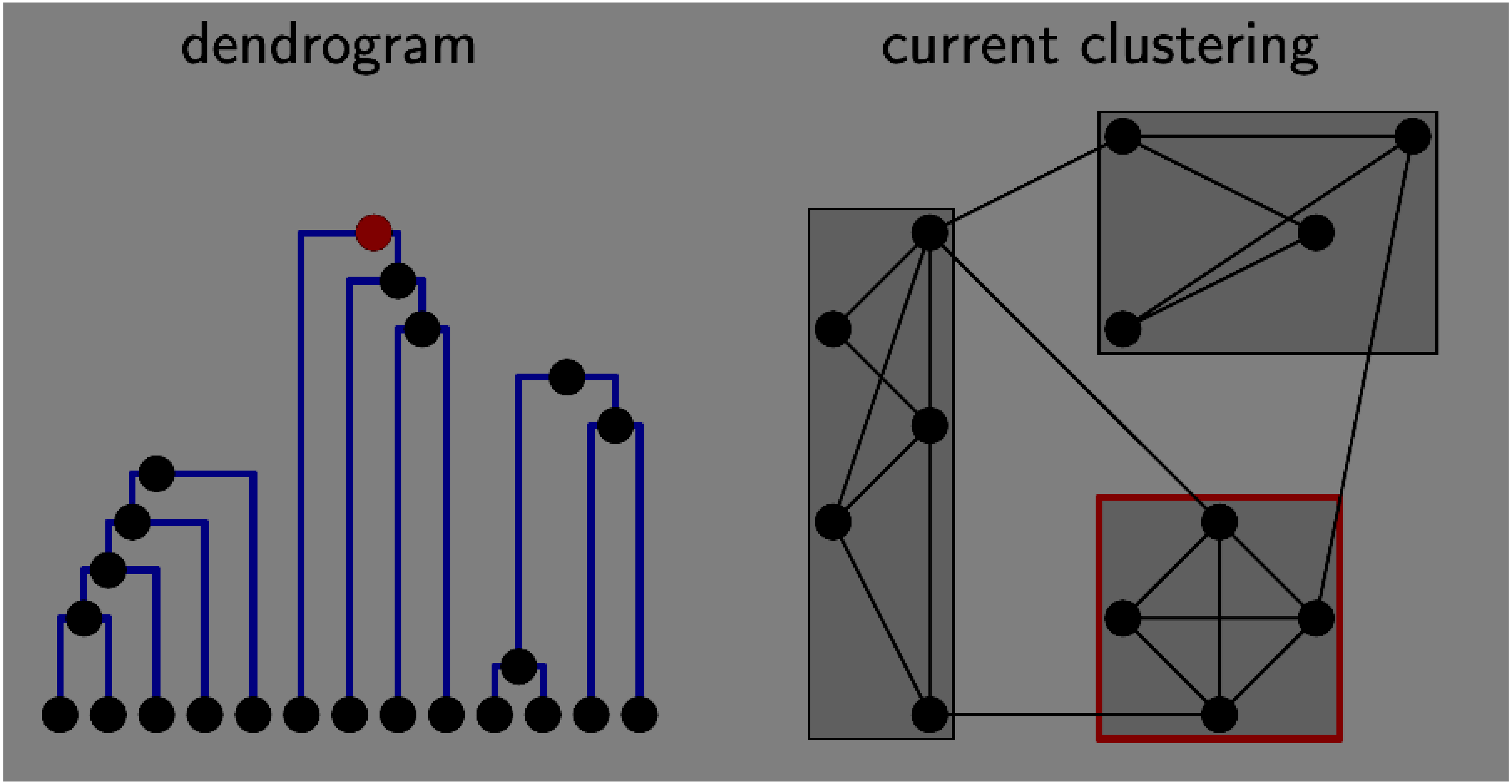
Agglomerative



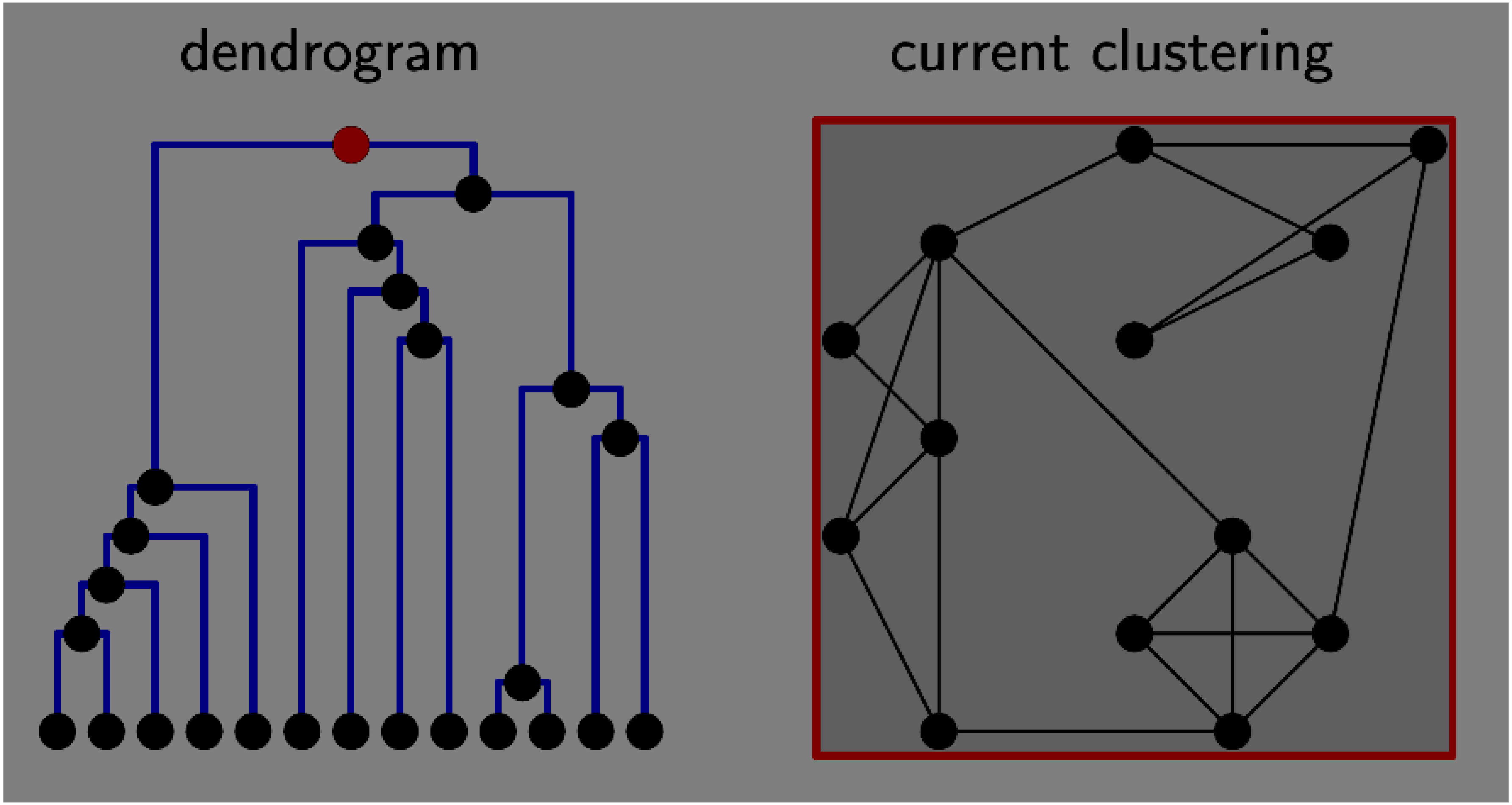
Agglomerative



Agglomerative



Agglomerative

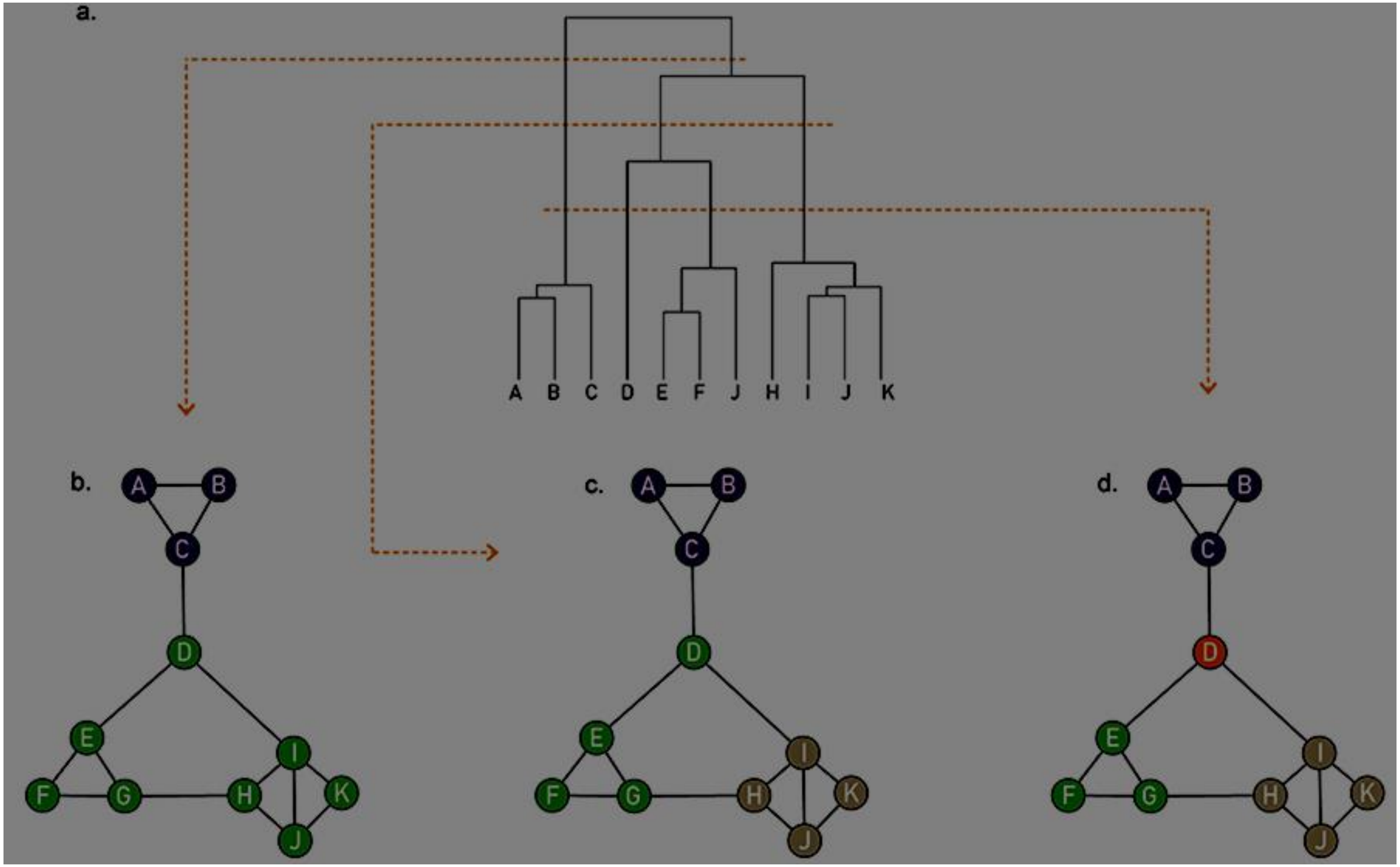


Strengths of Hierarchical Clustering

- Do not have to assume a specific number of clusters
  - Any desired number of clusters can be obtained by ‘cutting’ the dendogram at the proper level

- They may correspond to meaningful taxonomies
  - Example in biological sciences (e.g., animal kingdom, phylogeny reconstruction, …)

Where to cut?

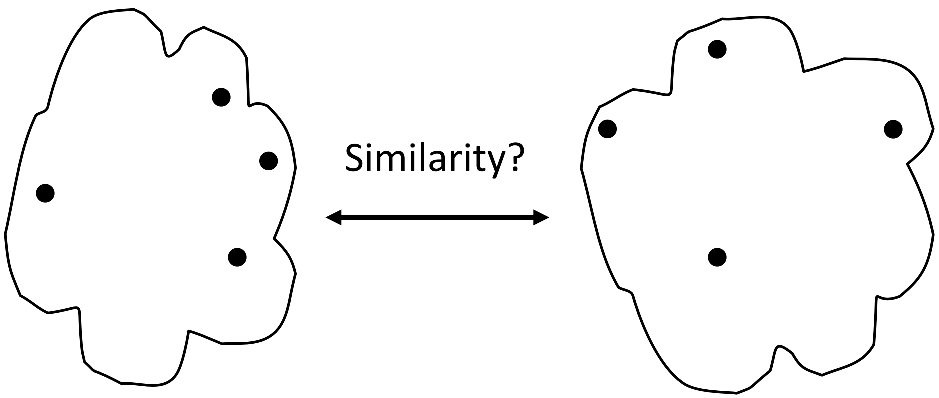


Agglomerative Clustering Algorithm

Key operation is the computation of the proximity of two clusters

- Different approaches to defining the distance between clusters distinguish the different algorithms

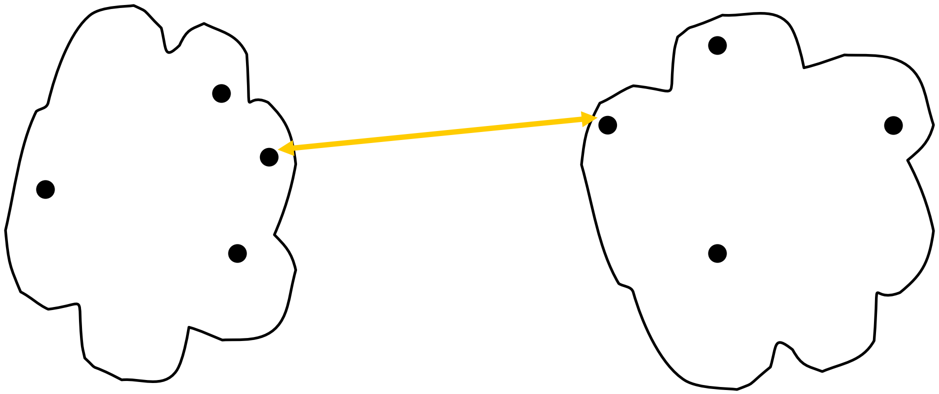
How to Define Inter-Cluster Similarity



	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

How to Define Inter-Cluster Similarity



	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

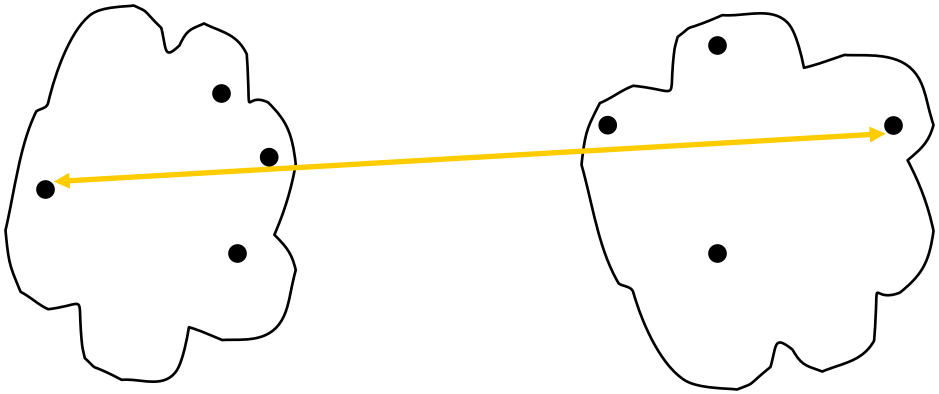
Proximity Matrix

MIN or single link

The two most similar (closest) points in the different clusters

sensitive to outliers

How to Define Inter-Cluster Similarity



	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

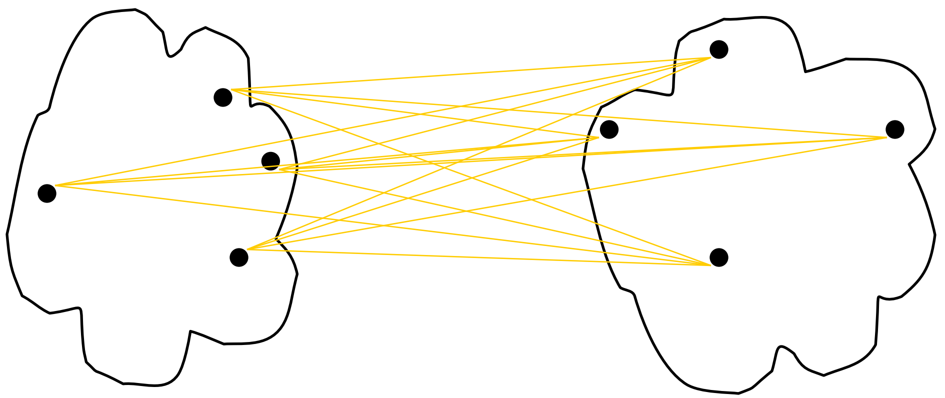
MAX or complete linkage

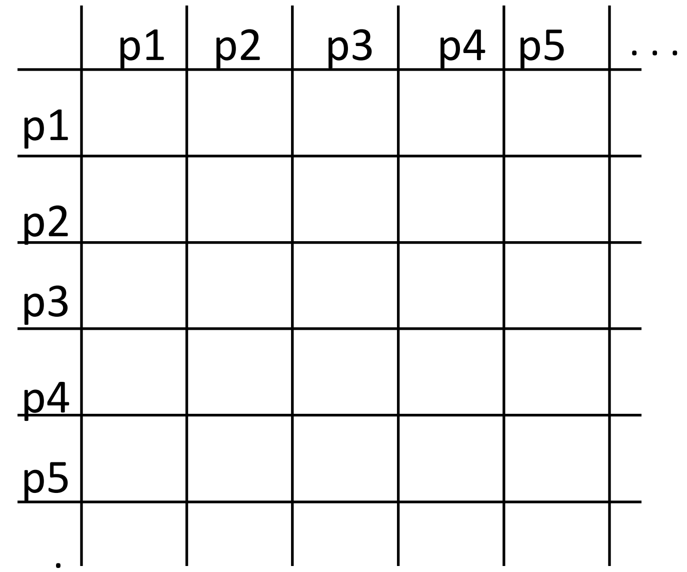
The two least similar (most distant) points in the different clusters

Tends to break large clusters

Biased towards globular clusters

How to Define Inter-Cluster Similarity





Proximity Matrix

Group Average

The average of pairwise proximity between points in the two clusters.

Clustering

- Data is often non-linked (matrix rows)
- Clustering works on the distance or similarity matrix, e.g., k-means.
- If you use k-means with adjacency matrix rows, you are only considering the ego-centric network

Community detection

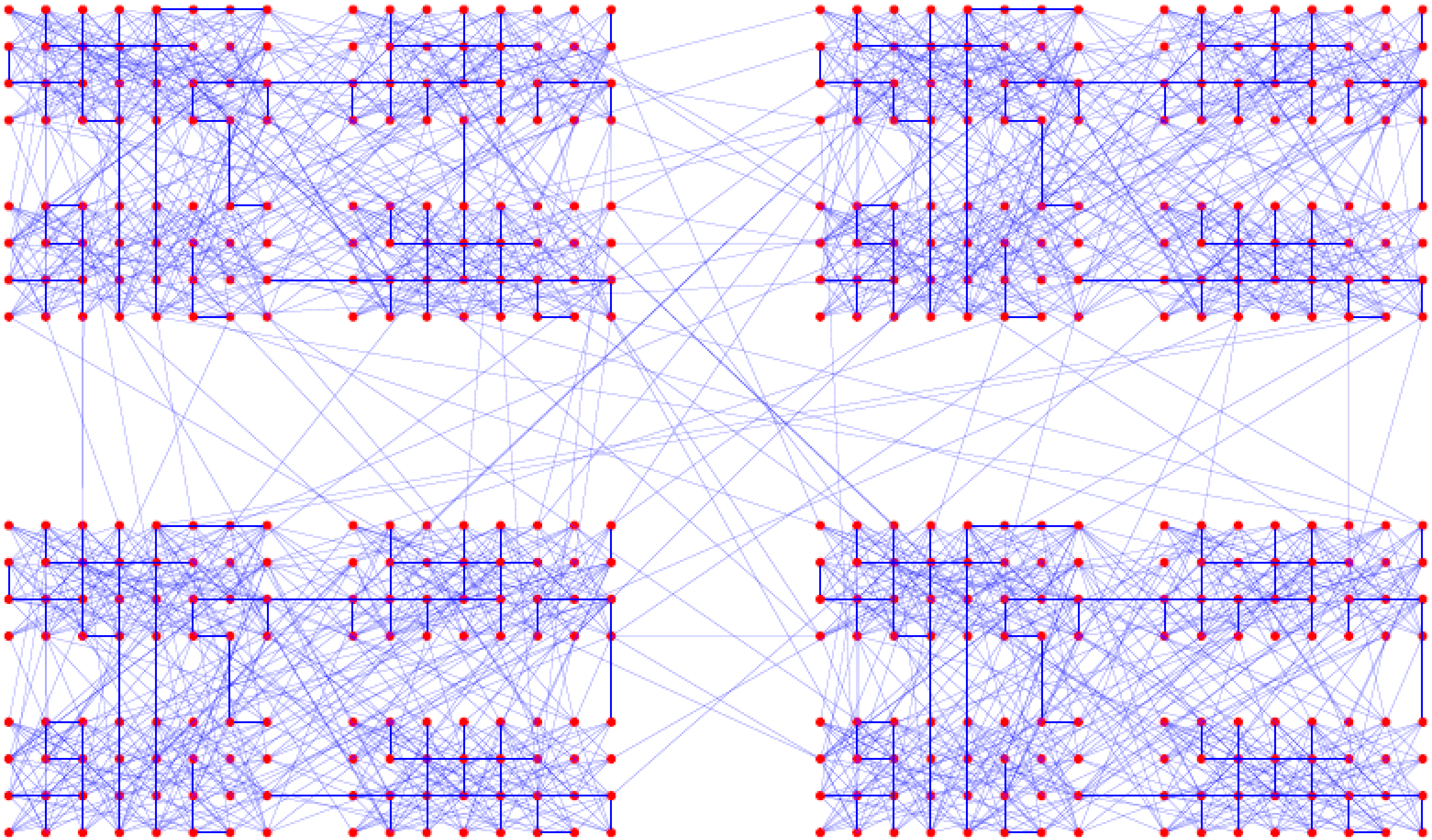
- Data is linked (a graph)
- Network data tends to be "discrete", leading to algorithms using the graph property directly
  - k-clique, quasi-clique, or edge-betweenness
  - But wait for embeddings

Outline

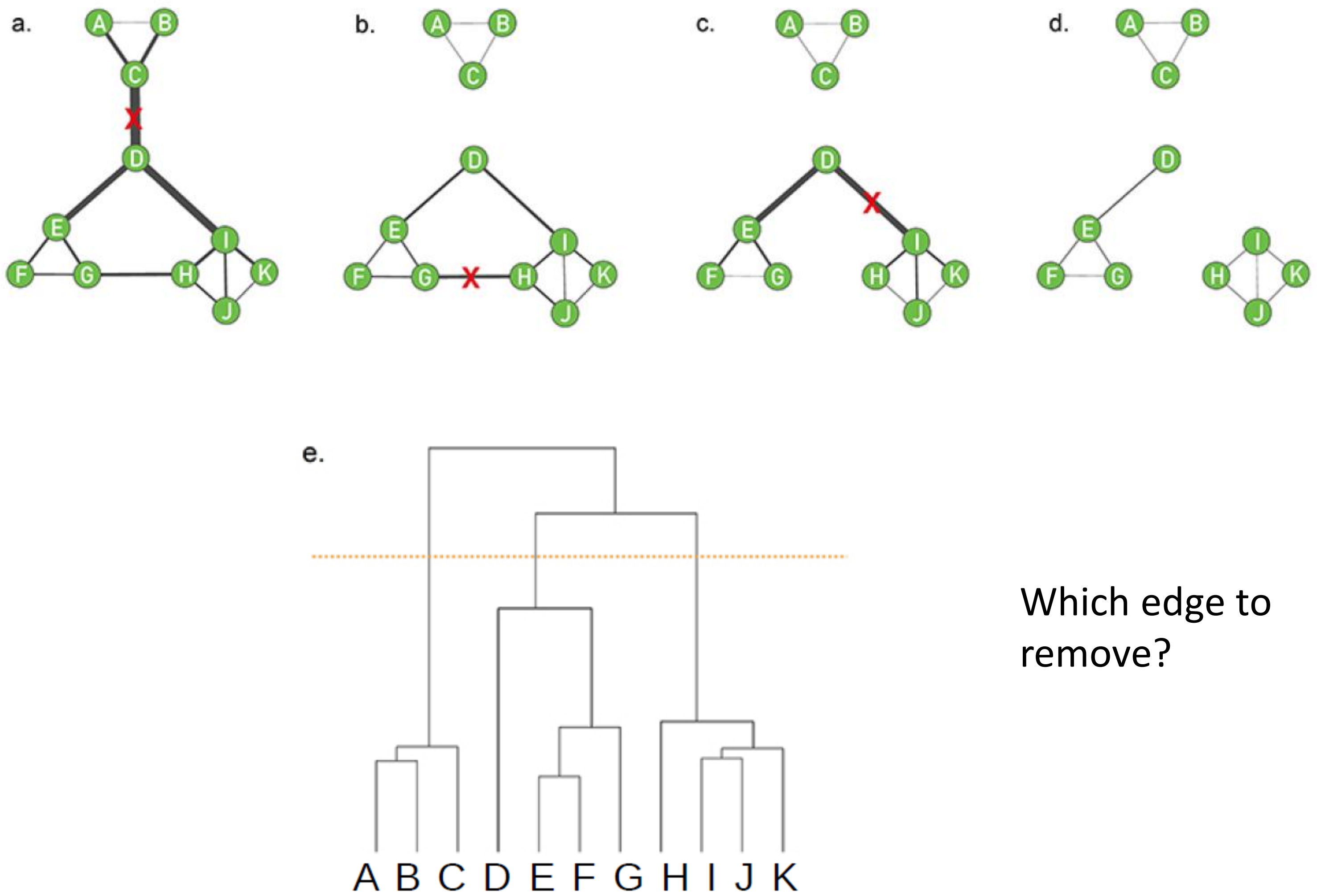
PART I

1. Introduction: what, why, types?
2. Cliques
3. Background: How it relates to “cluster analysis” (node/edge similarity)
4. Betweeness centrality
5. Modularity, label propagation

Example of a Hierarchically Structured Graph



Divisive Algorithms



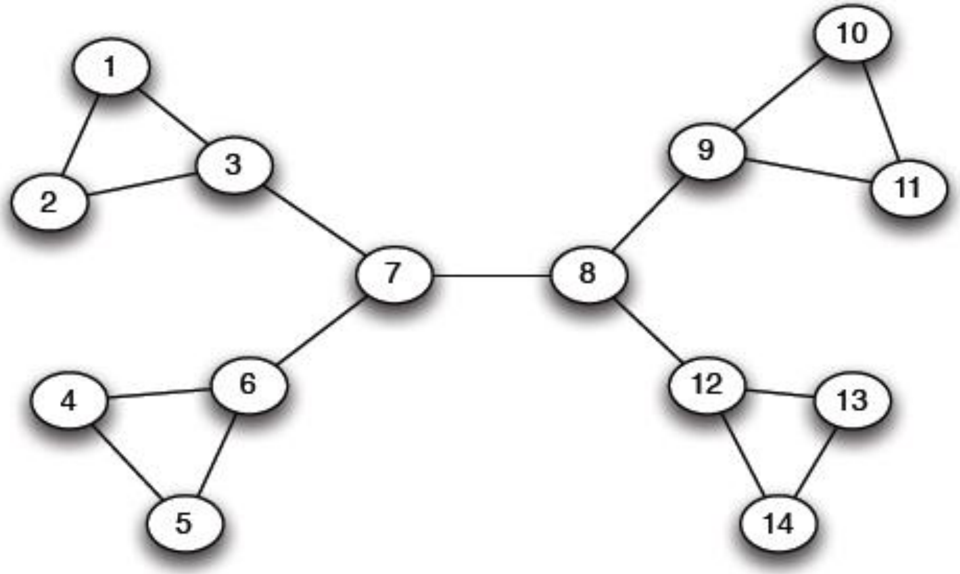
Which edge to remove?

The Girvan Newman method

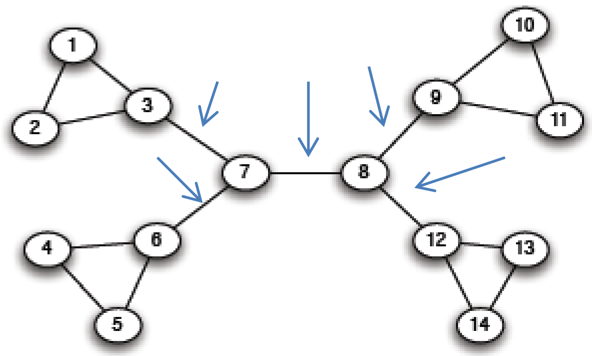
Hierarchical divisive method

- Start with the whole graph
- Find edges whose removal *"partitions"* the graph
- Repeat with each subgraph until single vertices

Which edge?



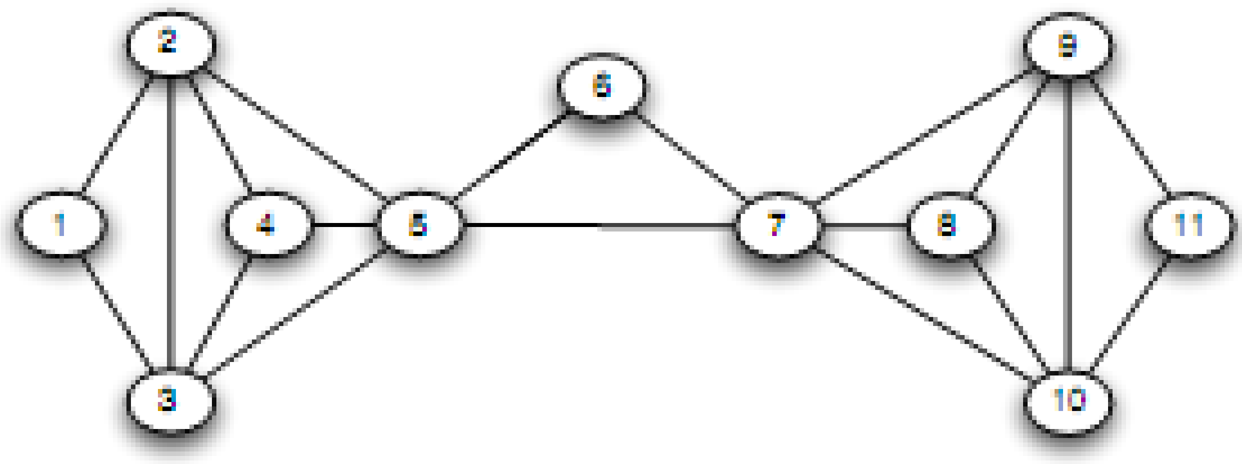
The Girvan Newman method



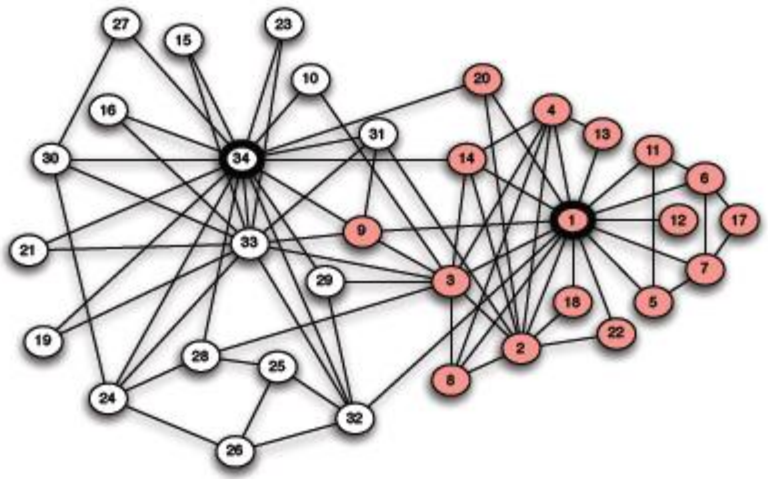
Use bridges or cut-edge (if removed, the nodes become disconnected)

Which one to choose?

The Girvan Newman method

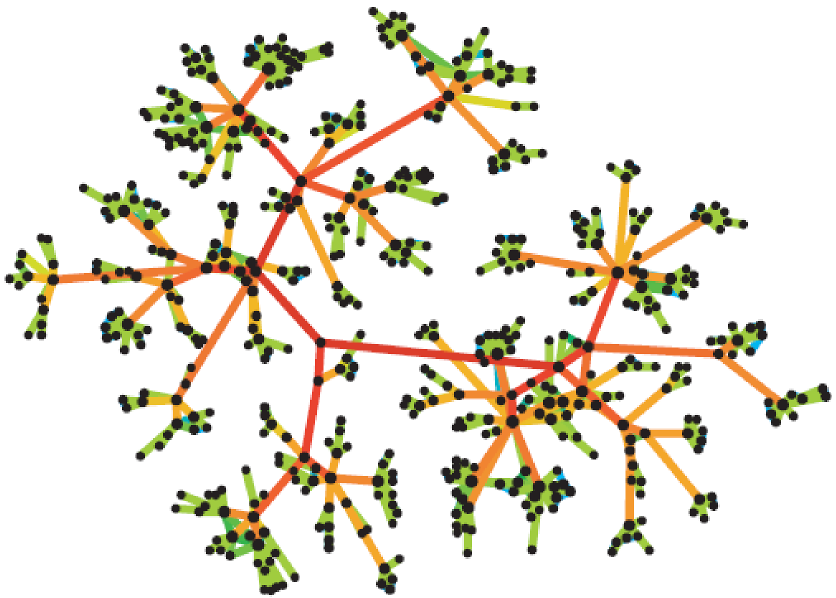


There may be none!



Strength of Weak Ties

- Edge betweenness: Number of shortest paths passing over the edge
- Intuition: Assuming communication through shortest paths, captures *traffic*



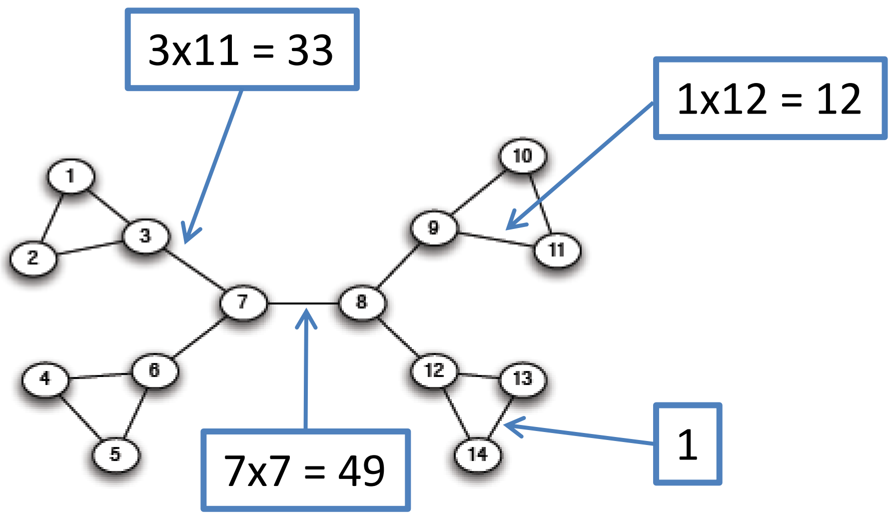
Edge betweenness in a real network

Edge Betweenness

Betweenness of an edge (a, b): number of pairs of nodes x and y such that the edge (a, b) lies on their shortest path

There can be multiple shortest paths, take the fraction that includes (a, b)

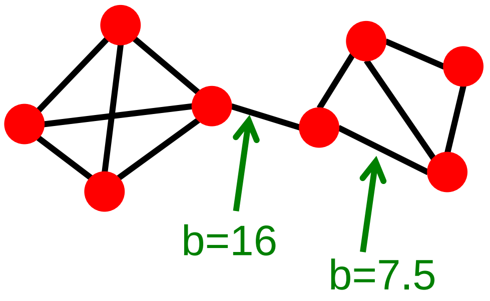
$$betweeness(a,b) = \sum_{(x,y)\in E} \frac{\#shortest\_paths(x,y)\_through(a,b)}{\#shortest\_paths(x,y)}$$



edges that have a high probability to occur on a randomly chosen shortest path between two randomly chosen nodes

Edge Betweenness

$$betweeness(a,b) = \sum_{(x,y)\in E} \frac{\#shortest\_paths(x,y)\_through(a,b)}{\#shortest\_paths(x,y)}$$

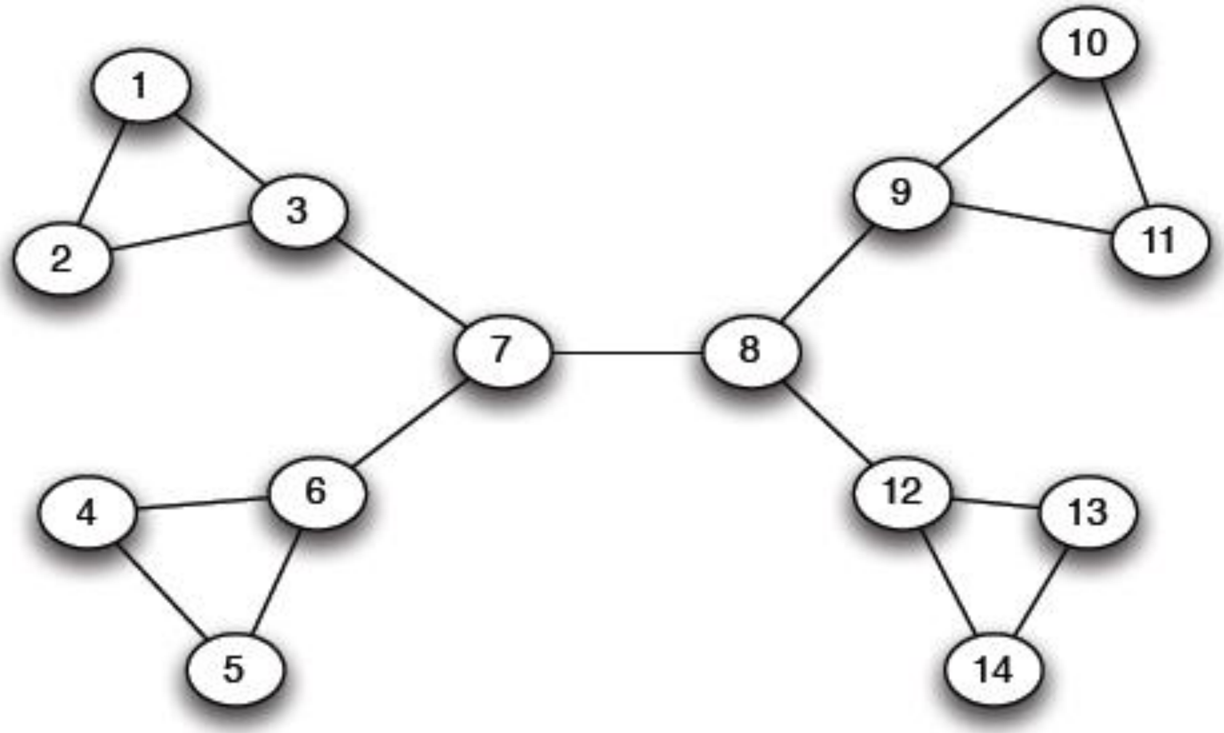


The Girvan Newman method

» **Undirected unweighted networks**

– Repeat until no edges are left:
  - Calculate betweenness of edges
  - Remove edges with highest betweenness
– Connected components are communities
– Gives a hierarchical decomposition of the network

Girvan Newman method: An example

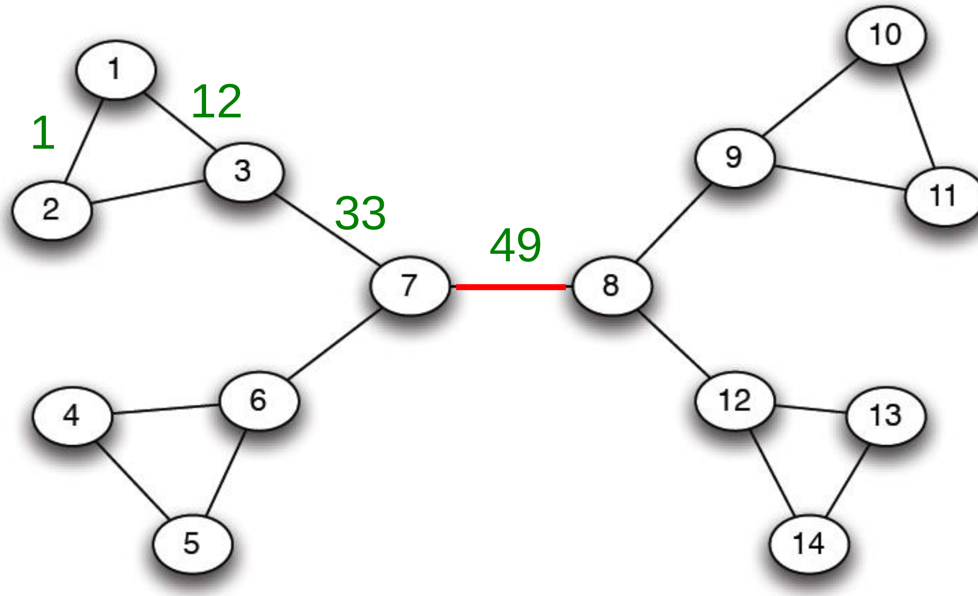


Betweenness(7, 8)= 7x7 = 49

Betweenness(3, 7)=Betweenness(6, 7)=Betweenness(8, 9) = Betweenness(8, 12)= 3x11=33

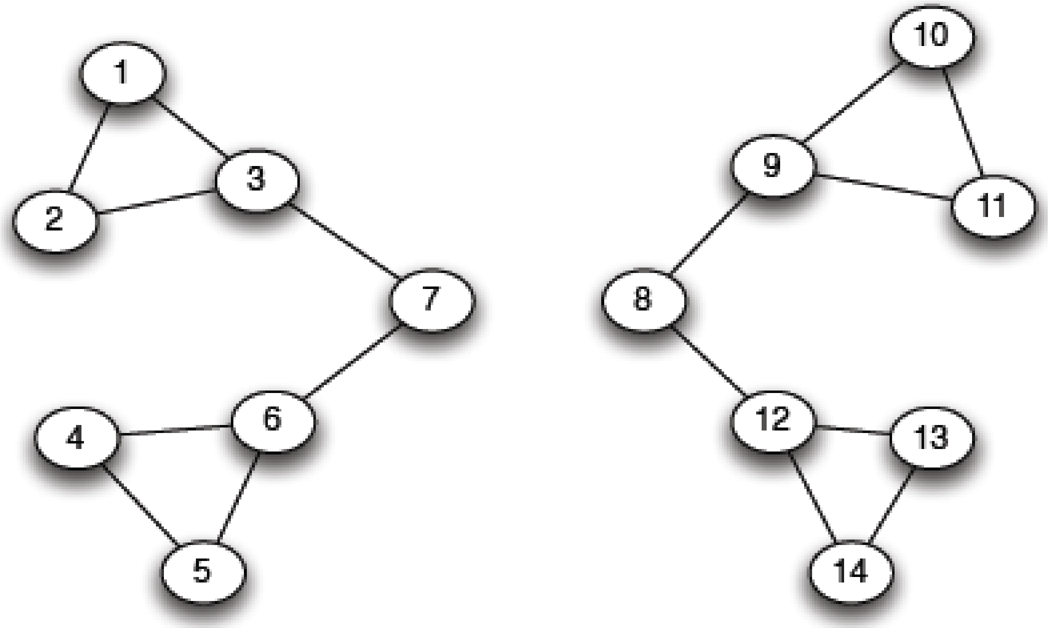
Betweenness(1, 3) = 1x12=12

Girvan-Newman: Example



Need to re-compute betweenness at every step

Girvan Newman method: An example

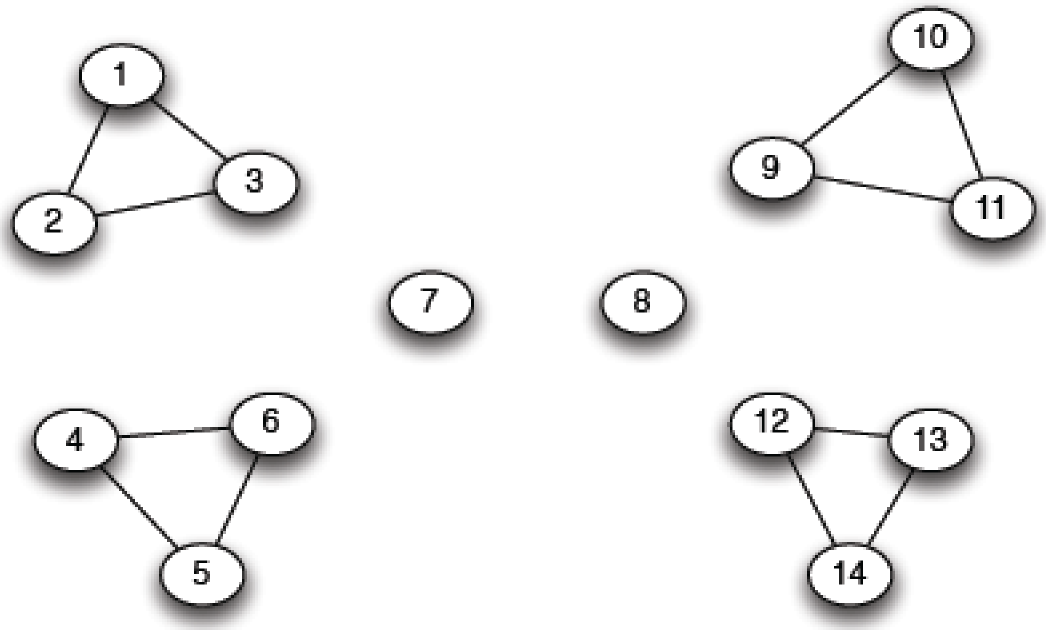


(a) *Step 1*

Betweenness(1, 3) = 1x5=5

Betweenness(3,7)=Betweenness(6,7)=Betweenness(8,9) = Betweenness(8,12)= 3x4=12

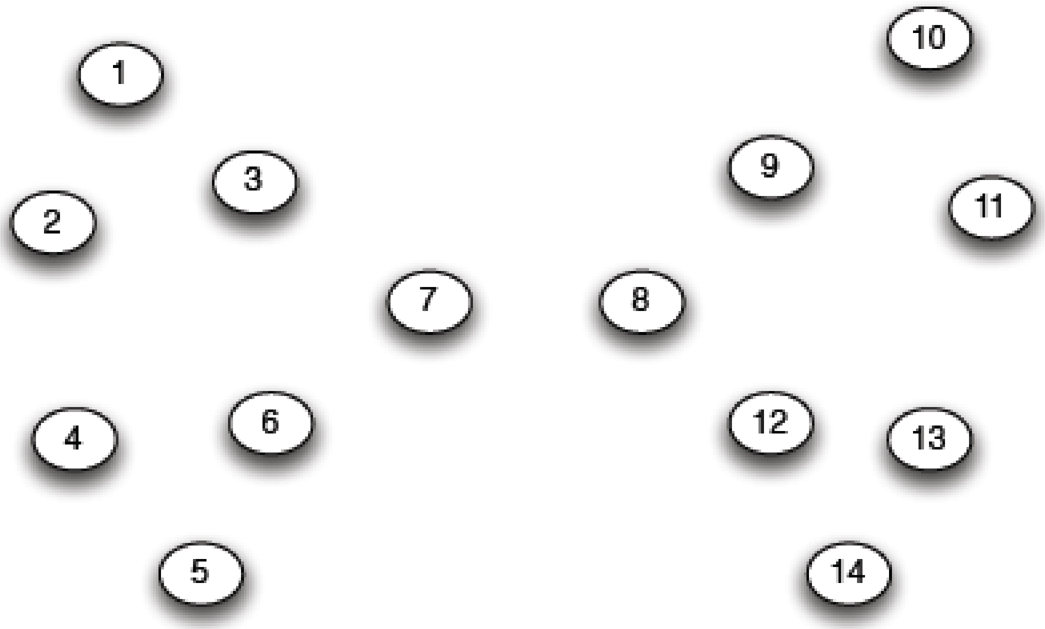
Girvan Newman method: An example



(b) *Step 2*

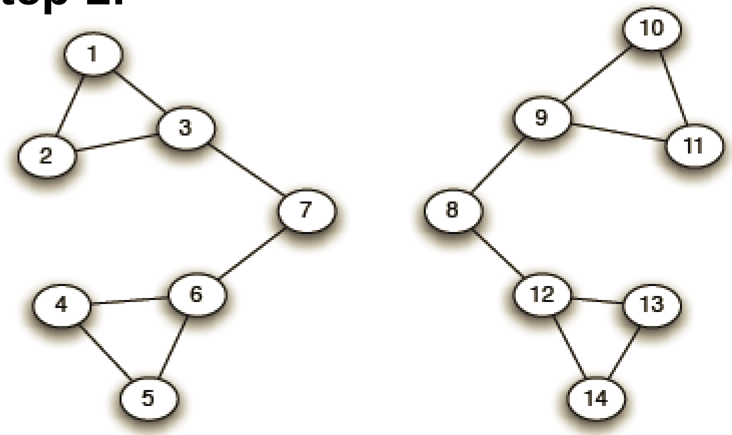
Betweenness of every edge = 1

Girvan Newman method: An example

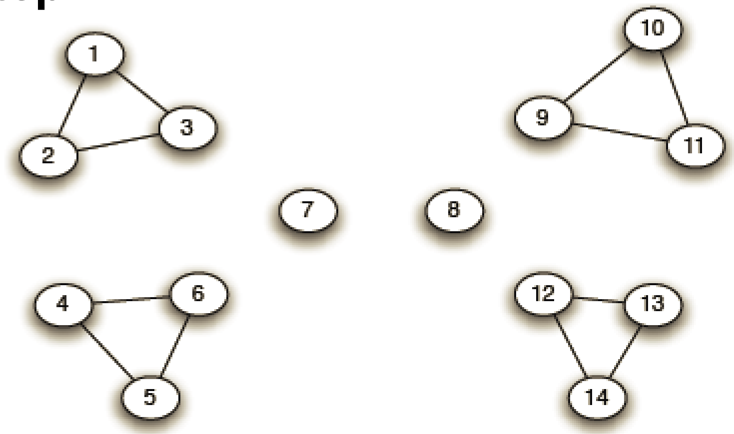


Girvan-Newman: Example

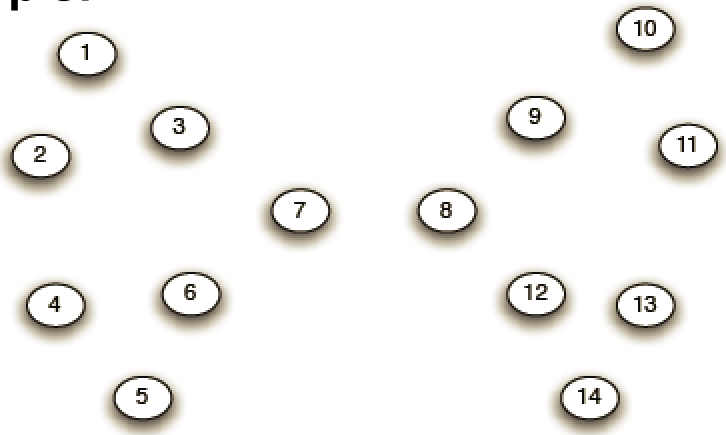
Step 1:



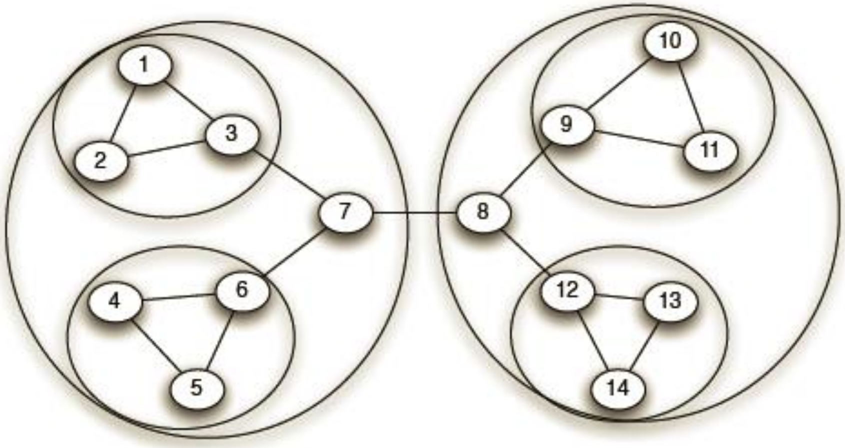
Step 2:



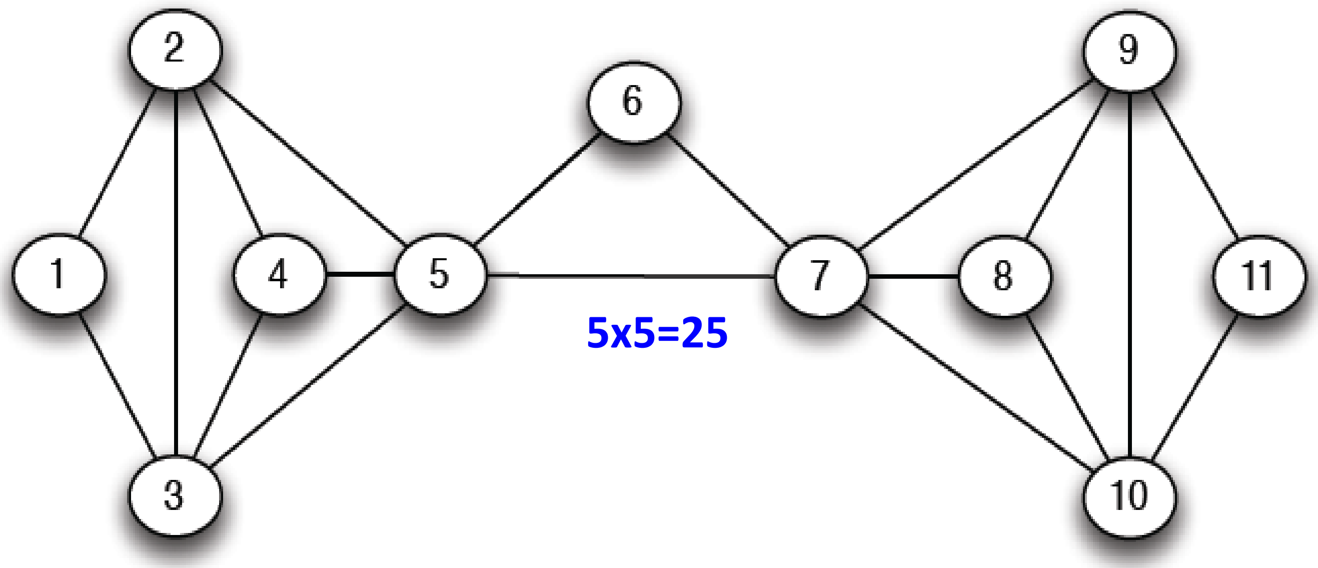
Step 3:



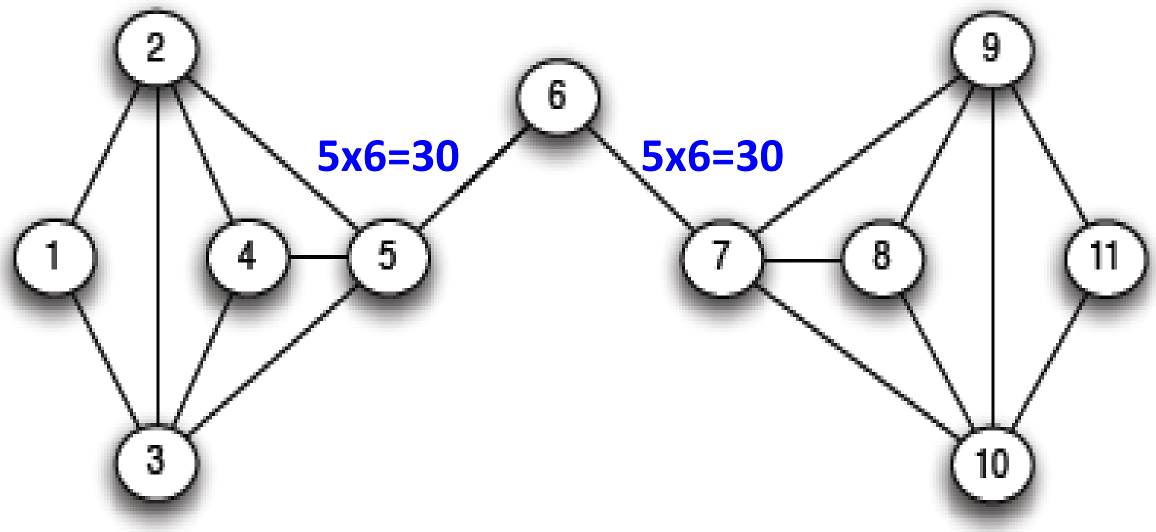
Hierarchical network decomposition:



Another example

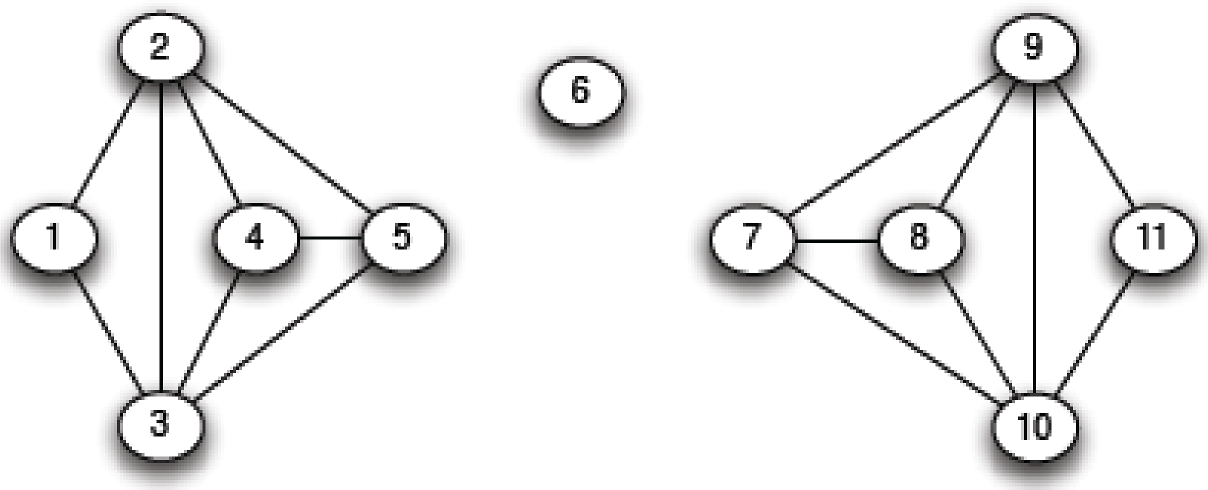


Another example



(a) *Step 1*

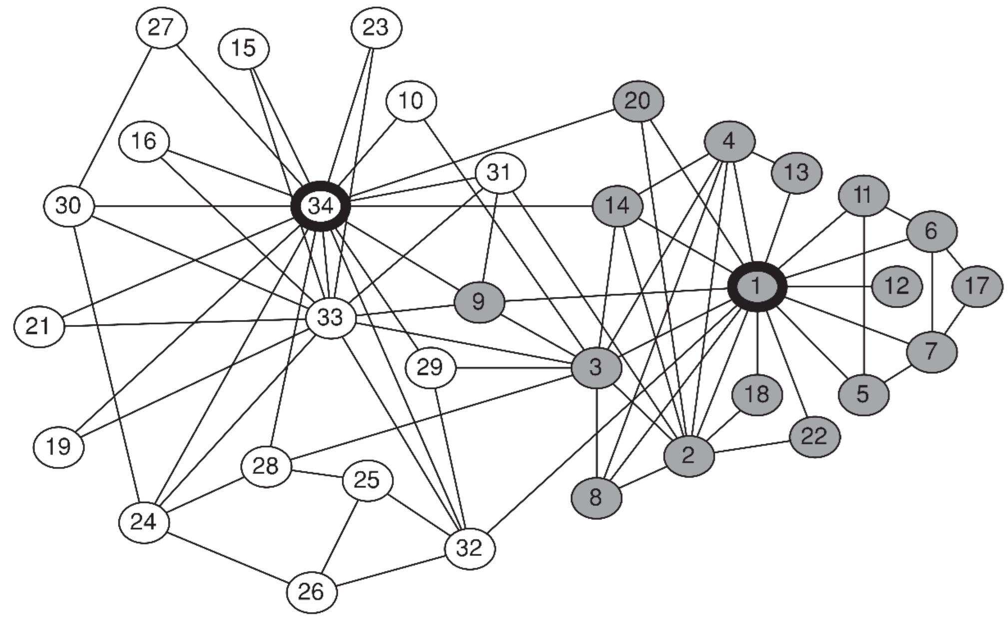
Another example



(b) *Step 2*

Zachary's karate club

Interactions between 34 members of a karate club for over two years

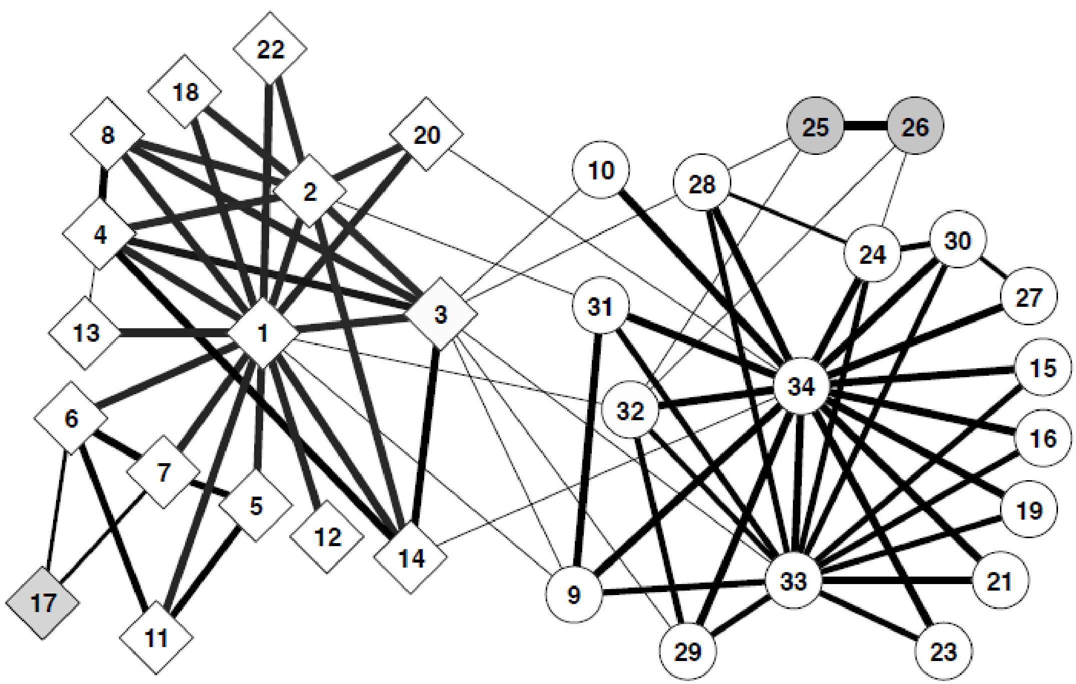


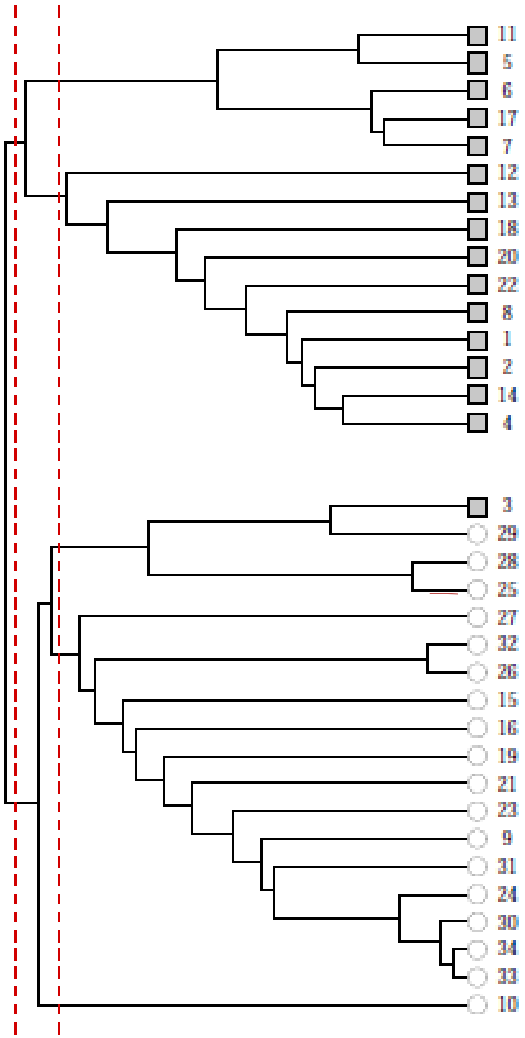
- The club members split into two groups (**gray** and **white**)
- Disagreement between the administrator of the club (node **34**) and the club's instructor of the club(node **1**),
- The members of one group left to start their own club

The same communities can be found using community detection

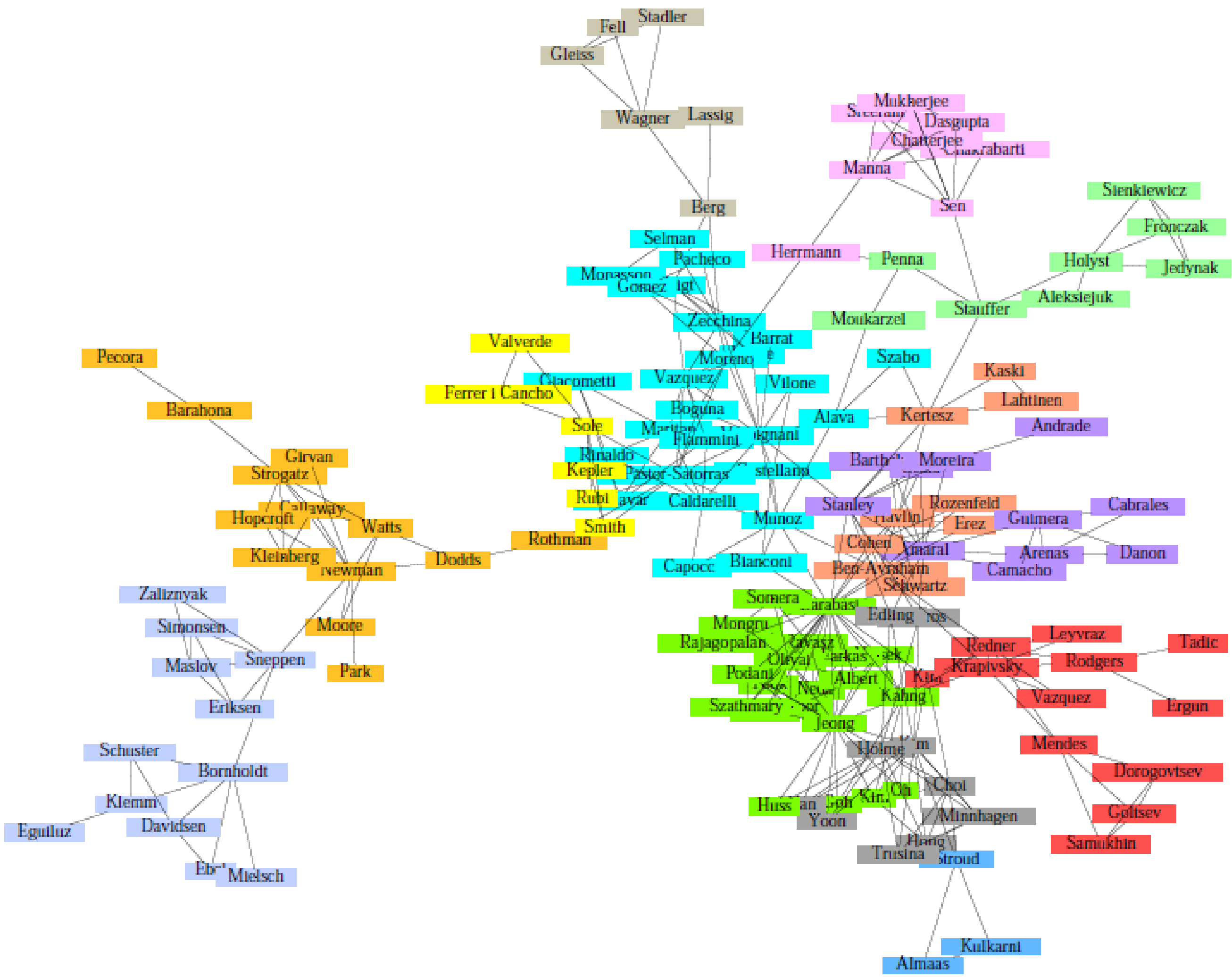
Girvan-Newman: Results

- **Zachary's Karate club:** Hierarchical decomposition





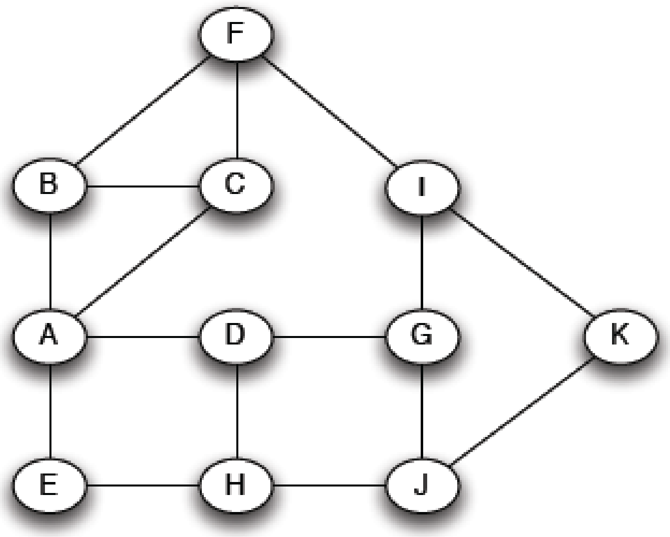
Girvan-Newman: Results



Communities in physics collaborations

How to Compute Betweenness?

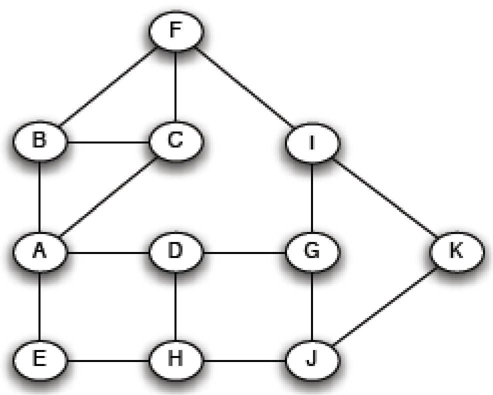
- Want to compute betweenness of paths starting at node A



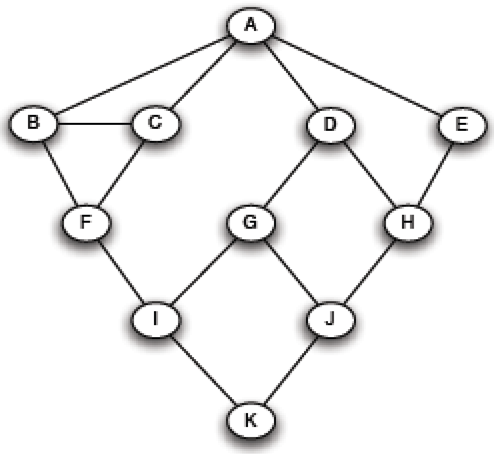
Computing Betweenness

1. Perform a *BFS* starting from A
2. Determine the *number of shortest path* from A to each other *node*
3. Based on these numbers, determine the amount of *flow* from A to all other nodes that uses each edge

Computing Betweenness: step 1



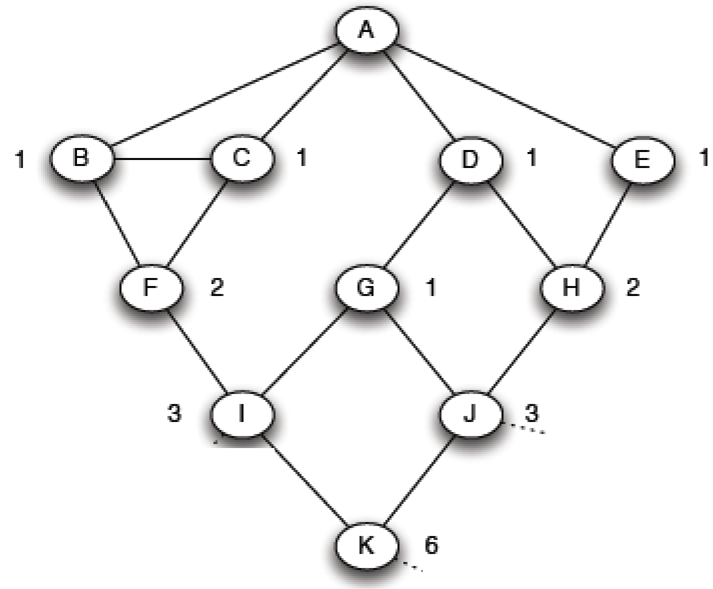
Initial network



BFS on A

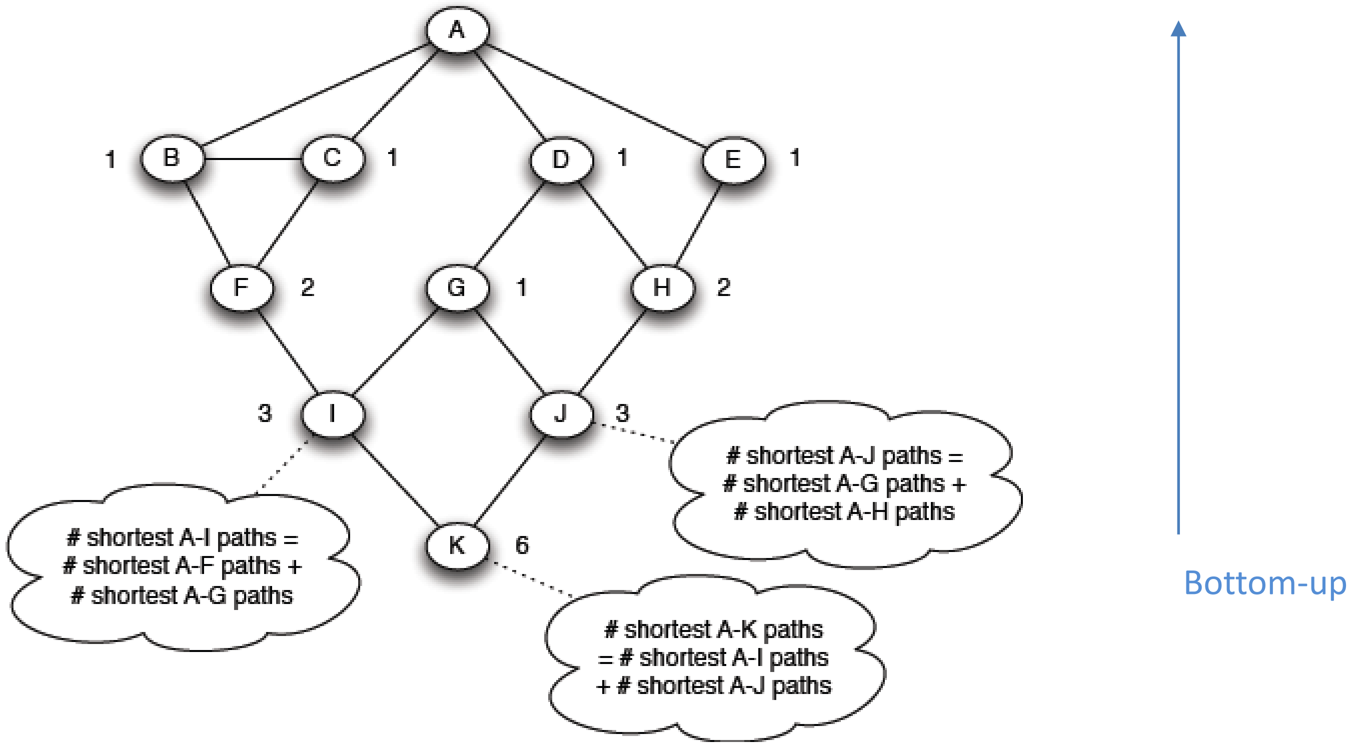
Computing Betweenness: step 2

Count how many shortest paths from A to a specific node



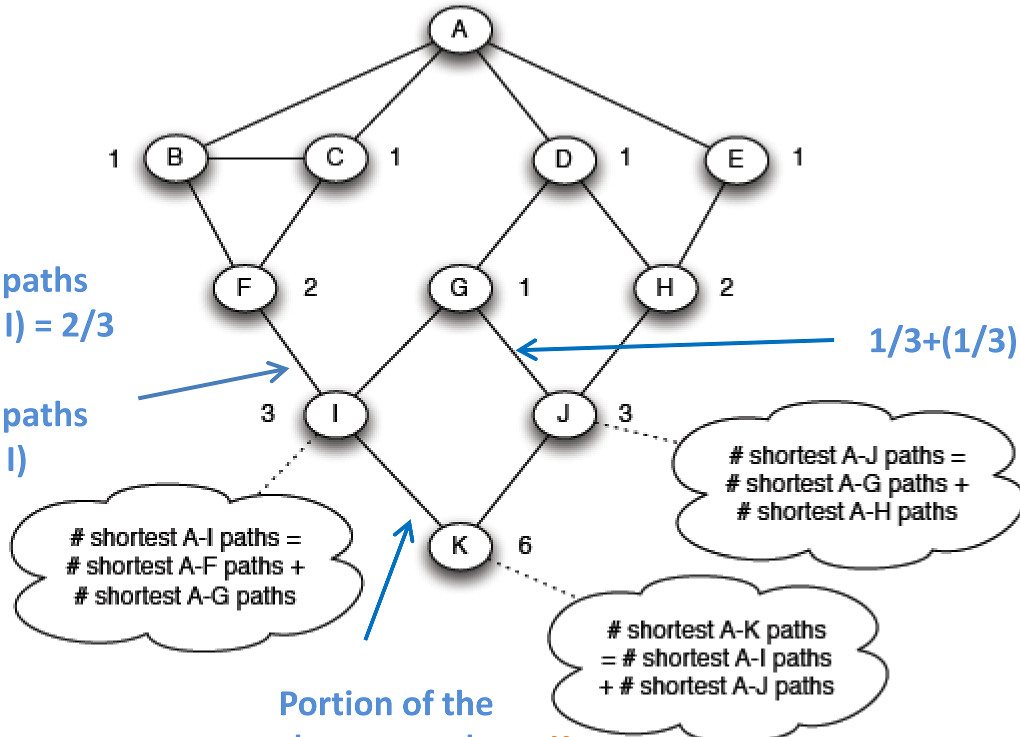
Computing Betweenness: step 3

Compute betweenness by working up the tree: If there are multiple paths count them fractionally



Computing Betweenness: step 3

Count the flow through each edge

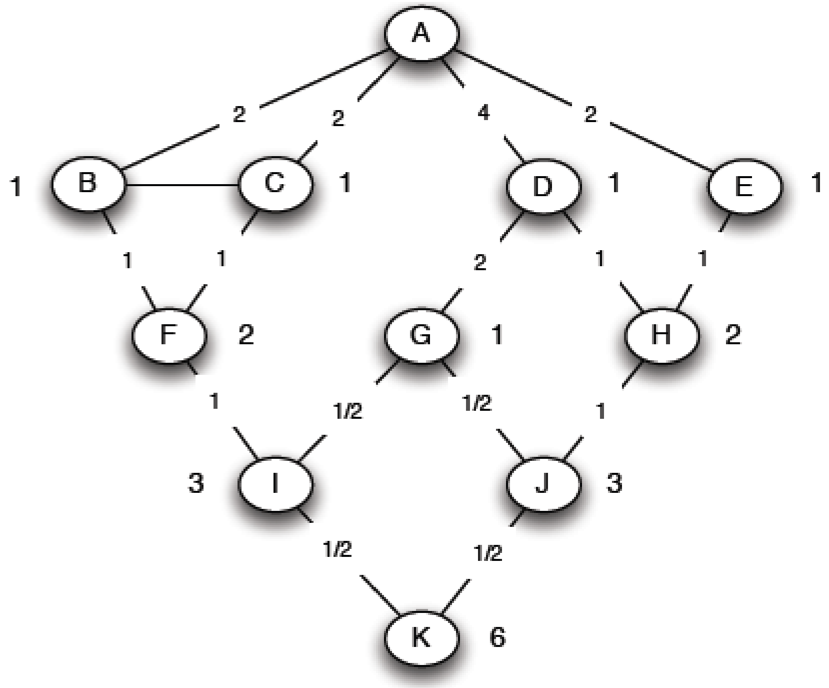


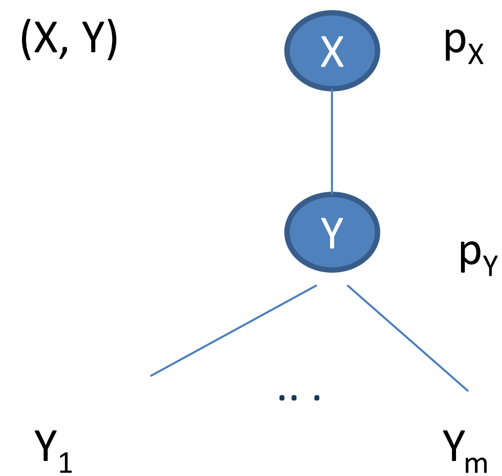
Portion of the shortest paths to **I** that go through (F, I) = 2/3
+
Portion of the shortest paths to **K** that go through (F, I)
(2/3) (1/2) = 1/3
= 1

1/3+(1/3)1/2 = 1/2

Portion of the shortest paths to **K** that go through (I, K)
= 3/6 = 1/2

Computing Betweenness: step 3





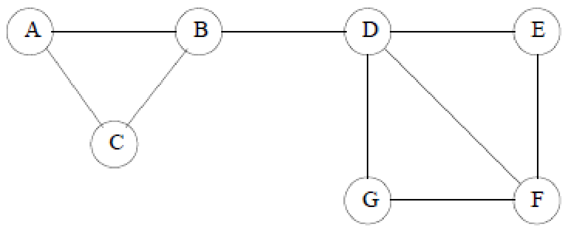
$$flow(X,Y) = \frac{p_X}{p_Y} + \sum_{Y_i\ child\ of\ Y} \frac{p_X}{p_Y}\ flow(Y,Y_i)$$

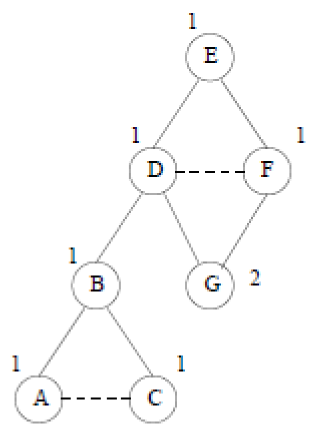
Computing Betweenness

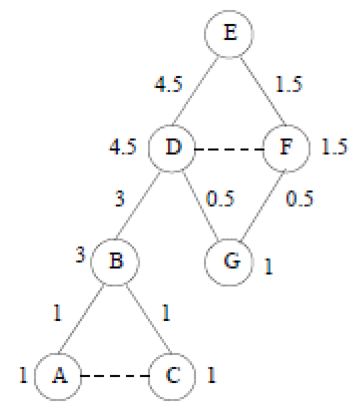
Repeat the process for all nodes

Sum over all BFSs

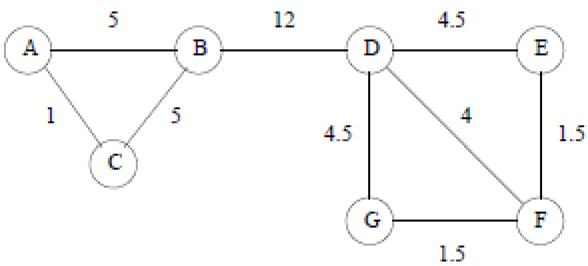
Example

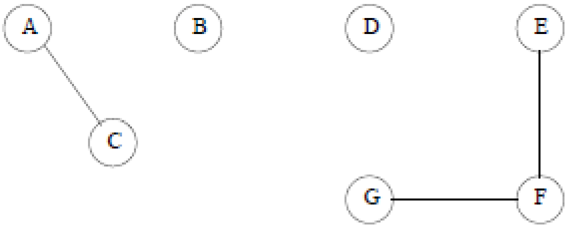






Example





Computing Betweenness

Issues

- Test for connectivity?
- Re-compute all paths, or only those affected
- Parallel computation
- Sampling

Centrality measures

Degree centrality

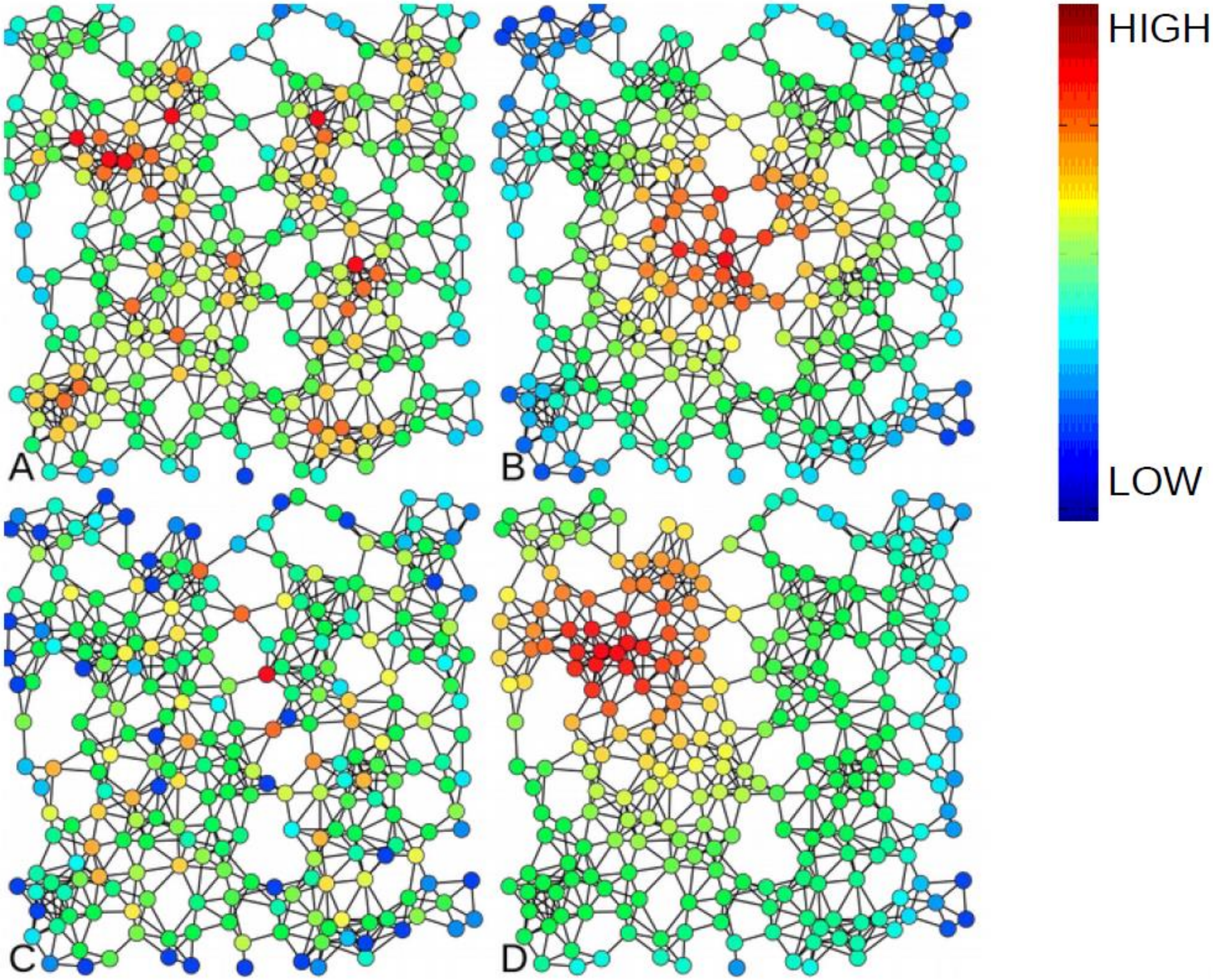
$$\text{closeness}(u) = \frac{1}{\sum_{v \in V, v \neq u} d(u, v)}$$

A: Degree

B: Closeness

C: Betweenness

D: PageRank



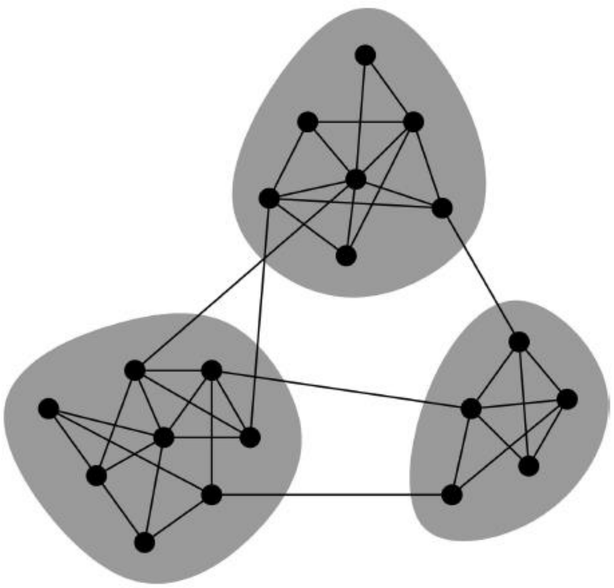
Outline

PART I

1. Introduction: what, why, types?
2. Cliques and vertex similarity
3. Background: Cluster analysis
4. Betweeness centrality
5. Modularity, label propagation
6. How to evaluate

Modularity

- 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 $s \in S$:



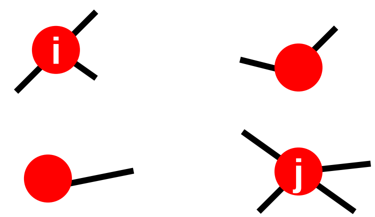
$$Q \propto \sum_{s \in S} [\ (\#\ \text{edges within group } s) - (\text{expected}\ \#\ \text{edges within group } s)\ ]$$

Need a null model!

a copy of the original graph keeping some of its structural properties but without community structure

Null Model: Configuration Model

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



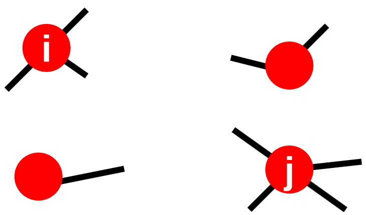
For any edge going out of i randomly, the probability of this edge getting connected to node j is $\frac{\boldsymbol{d_j}}{\boldsymbol{2m}}$

Because the degree for i is d_i*, we have* d_i *number of such edges*

Note:

$$\sum_{u \in N} d_u = 2m$$

Null Model: Configuration Model



- The expected number of edges in (multigraph) **G'**:
  - $= \frac{1}{2}\sum_{i\in N}\sum_{j\in N}\frac{d_i d_j}{2m} = \frac{1}{2}\cdot\frac{1}{2m}\sum_{i\in N} d_i\left(\sum_{j\in N} d_j\right) =$
  - $= \frac{1}{4m}2m\cdot 2m = m$

Modularity

- Given a degree distribution, we know the expected number of edges between any pairs of vertices
- We assume that real-world networks should be far from random.
- The more distant they are from this randomly generated network, the more structural they are.
- Modularity defines this distance and modularity maximization tries to maximize this distance

Consider a partitioning of the data into $\mathrm{S} = (s_1, s_2, s_3, \dots, s_k)$

For partition s_x, this distance can be defined as

$$\sum_{i,j \in s_x} A_{ij} - \frac{d_i d_j}{2m}$$

Modularity

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

Normalizing cost.: -1<Q<1

A_{ij} = 1 if i→j, 0 else

- Modularity values take range [−1, 1]
  - It is positive if the number of edges within groups exceeds the expected number
  - 0.3-0.7 < Q means significant community structure

Modularity

Greedy method of Newman (one of the many ways to use modularity)

Agglomerative hierarchical clustering method

1. Start with a state in which each vertex is the sole member of one of *n* communities
2. Repeatedly join communities together in pairs, choosing at each step the join that results in the greatest increase (or smallest decrease) in Q.

Since the joining of a pair of communities between which there are no edges can never result in an increase in modularity, we ***need only consider those pairs between which there are edges***, of which there will at any time be at most m

Louvain Algorithm

– A greedy modularity optimization method for community detection
  - Invented when all authors affiliated with *Université catholique de Louvain (UCL)*

Louvain Algorithm

The algorithm has multiple passes

Each pass has **two phases**

1. Modularity Optimization
2. Community Aggregation

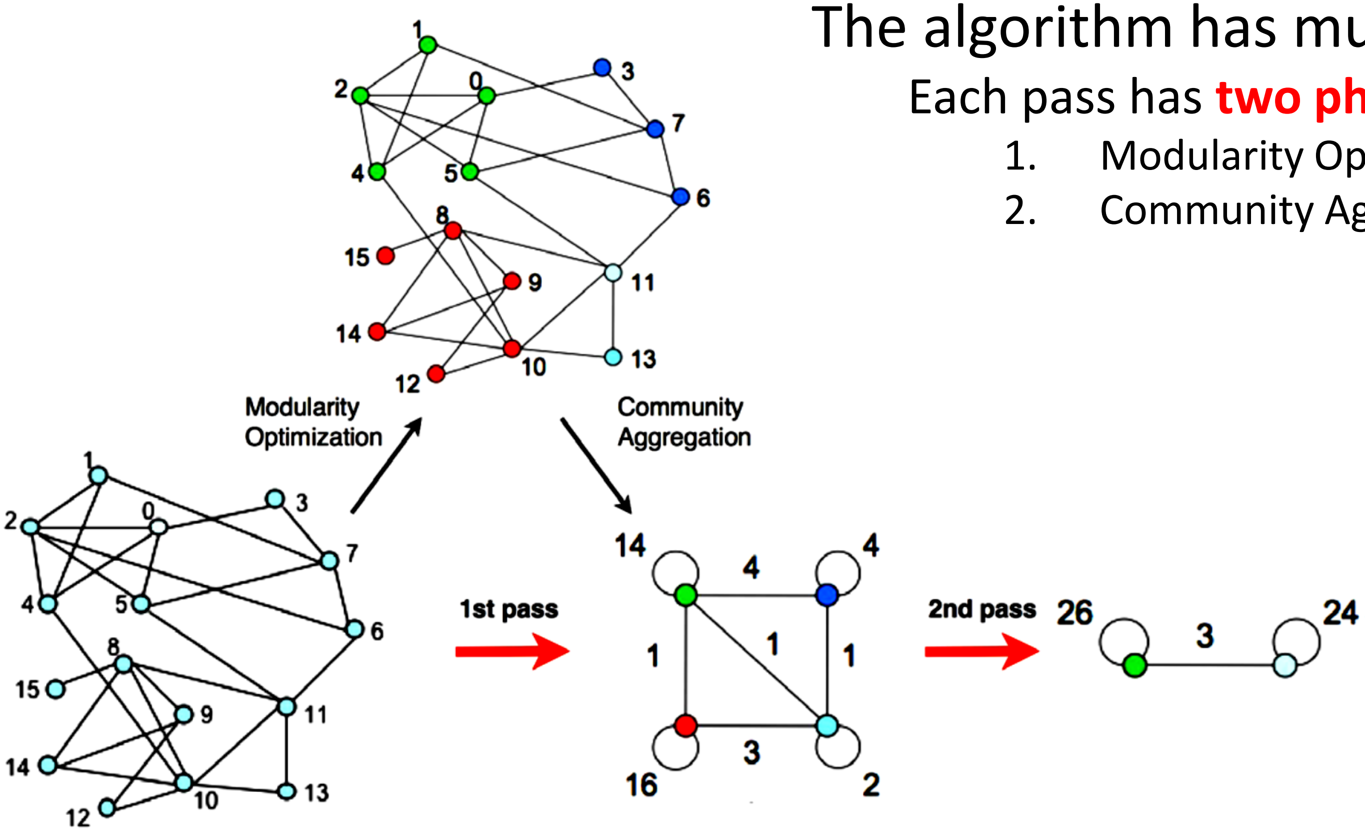


Image from
Blondel, Vincent D., Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. "Fast unfolding of communities in large networks." Journal of statistical mechanics: theory and experiment 2008, no. 10 (2008): P10008.

Louvain Algorithm

Start with a weighted network where all nodes are in their own communities (i.e., *n* communities)

First Phase:

- For each node v_i,
  - For all neighbors $v_j \in N(v_i)$:
    - compute the modularity gain if v_i is removed from its community and placed in the community of v_j.
  - Find the community with the maximum modularity gain
  - If the maximum gain is positive, remove v_i from its community, and place it in that community
  - If no positive gain, do not change communities

- Repeat until no node changes its community

Louvain Algorithm

- A point can be considered multiple times
- A local minima of modularity maximization is achieved in phase I
- Phase I is order dependent
  - The modularity achieved is more or less stable and is less dependent on the initial order
  - The computation time depends on the initial order.

Louvain Algorithm

Second Phase:

- Build a new network
  - Nodes are communities
  - Edges are the edges between nodes in the corresponding communities (weights are sum of the weights)
  - Self-loops represent edges within the community

- The algorithm creates hierarchies of communities
- It usually ends in less than 10 passes

Modularity

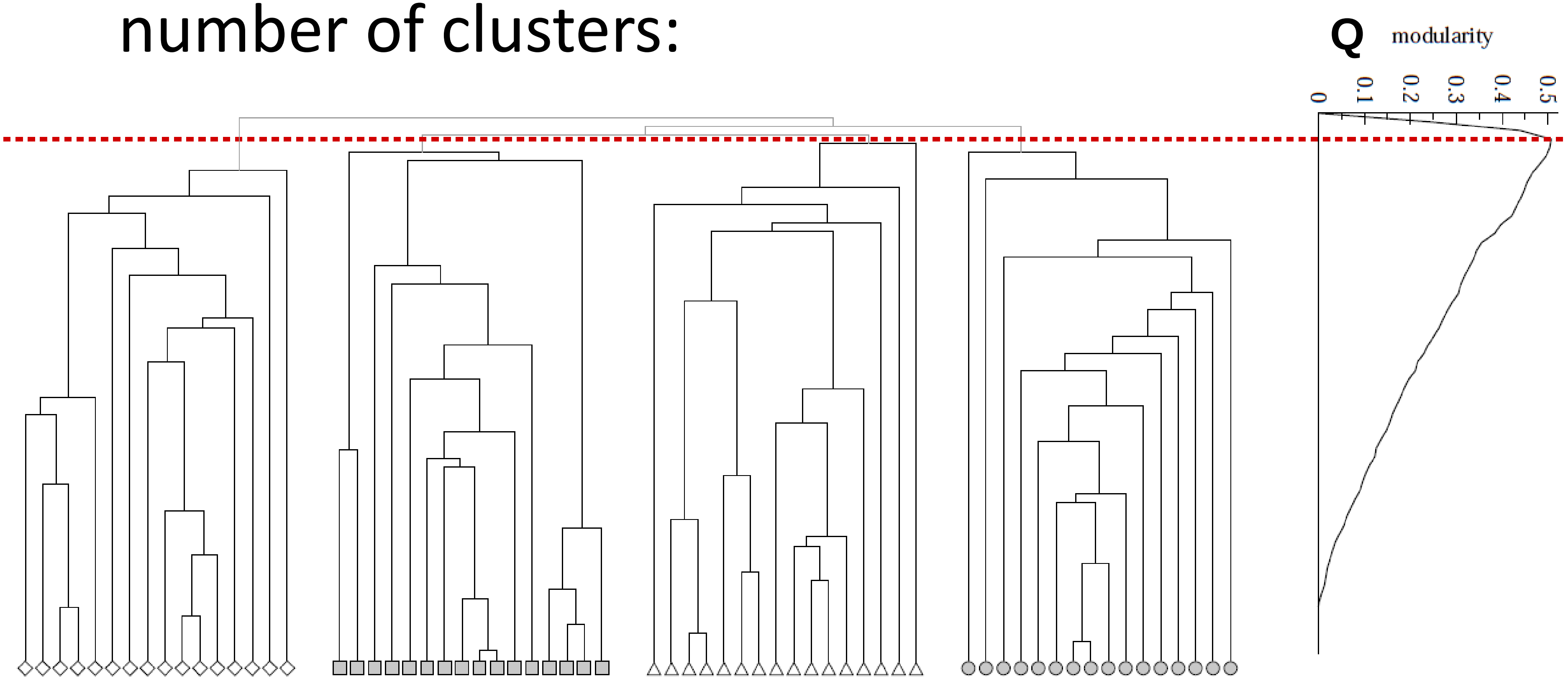
- Modularity of partitioning S of graph G:

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

$$\sum_{i \in S} \sum_{j \in S} \left(A_{ij} - \frac{d_i\, d_j}{2m} \right) = \sum_{i \in S} \sum_{j \in S} A_{ij} - \sum_{i \in S} \sum_{j \in S} \frac{d_i\, d_j}{2m} = L_{in} - \frac{(sum_{degree})^2}{2m}$$

Modularity: Number of clusters

- Modularity is useful for selecting the number of clusters:



Modularity: Cluster quality

When a given clustering is “good”?

Also, it is both a local (per individual cluster) and global measure

Outline

PART I

1. Introduction: what, why, types?
2. Cliques and vertex similarity
3. Background: Cluster analysis
4. Betweeness centrality
5. Modularity, label propagation

Label propagation

Vertices are initially given unique labels (e.g., their vertex labels).

At each iteration,

sweep over all vertices, in random sequential order:

each vertex takes the label shared by the majority of its neighbors.

If no unique majority, one of the majority label is picked at random.

Stop (convergence) when each vertex has the majority label of its neighbors

Communities: groups of vertices having identical labels at convergence

Label propagation

- *Labels propagate across the graph*: most labels will disappear, others will dominate.
- By construction, each vertex has more neighbors in its community than in any other community.
- Due to many possible ties, different partitions
  - Perform *many propagations* from the same initial condition, with different random seeds
  - *Aggregate partition label* each vertex with the set of all labels it has in different partitions → overlapping communities

Basic References

- Jure Leskovec, Anand Rajaraman, Jeff Ullman, Mining of Massive Datasets, Chapter 10, http://www.mmds.org/
- Reza Zafarani, Mohammad Ali Abbasi, Huan Liu, Social Media Mining: An Introduction, Chapter 6, http://www.socialmediamining.info/
- Santo Fortunato: Community detection in graphs. CoRR abs/0906.0612v2 (2010)
- Pang-Ning Tan, Michael Steinbach, Vipin Kumar, Introduction to Data Mining, Chapter 8, http://www.users.cs.umn.edu/~kumar/dmbook/index.php
- Albert-László Barabasi, Network Science, Chapter 9, http://networksciencebook.com/

Questions?