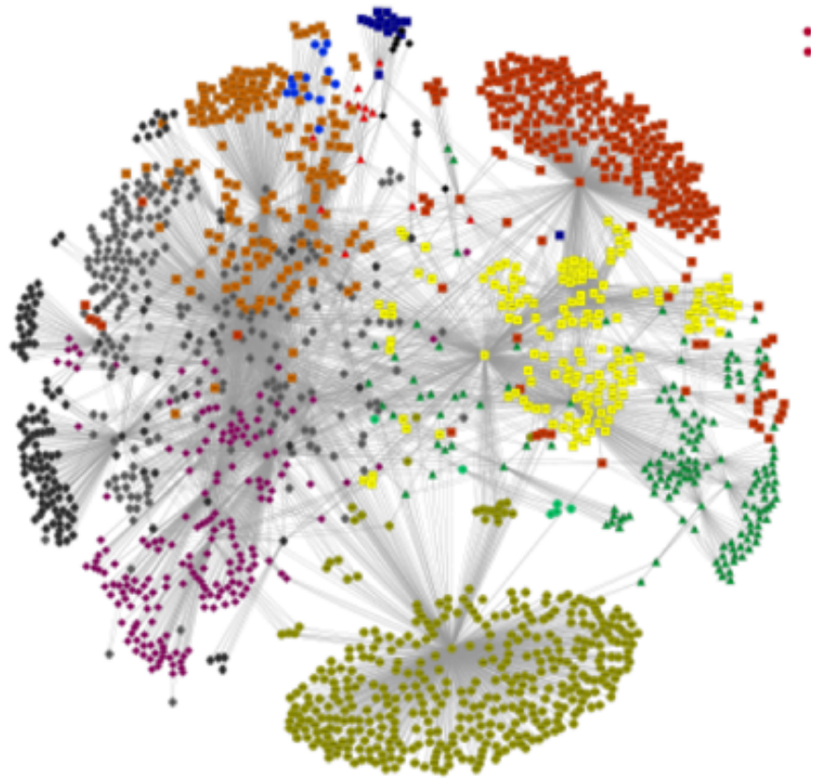
Graphs-1

Graph Properties, Centrality, Shortest Paths, Clustering

Why Graphs ?

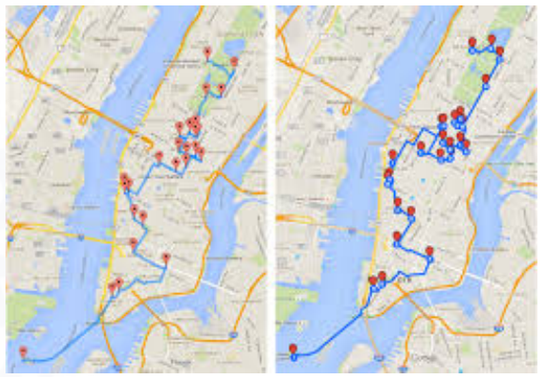
- Many Real world datasets are graphs
  - Social Networks
  - Web graphs
  - XML parse trees
  - Protein-protein interactions
  - Road Networks
  - ….

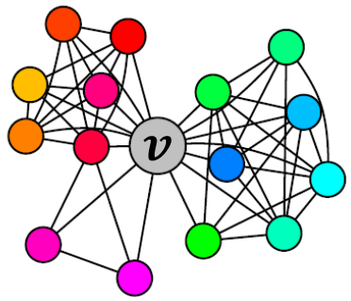


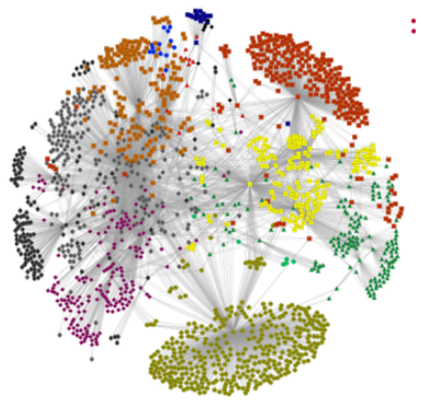
- **Many of these Graphs are massive and need automatic methods to understand them**

Whats possible with Graphs ?

- Finding shortest paths
  - Road networks, Social Networks
  - Internet routing
- Centrality
  - Importance of nodes, edges
- Discovering communities

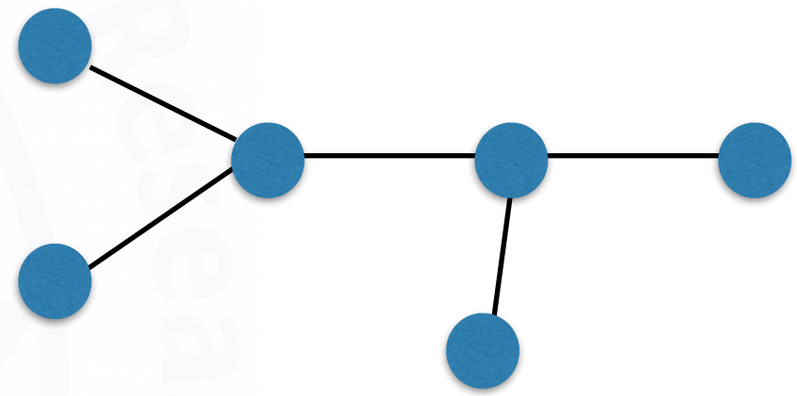






What are Graphs ?

- A graph is composed of vertices and edges $(V, E \subseteq V^2)$
  - Elements in V are **vertices** or **nodes**
  - Pairs (v,u) in E are edges of the graph
    - Pairs can be ordered or unordered
  - directed and undirected graphs
  - Graphs can be labelled
    - Vertices can have a labelling L(v)
    - Edges can have a labelling L(u,v)
- A tree is a rooted, connected and acyclic graph
- Graphs can be represented using Adjacency matrices
  - |V|X|V| matrix **A** with A(i,j) = 1 if $(v_i, v_j) \in E$



|V| = n

|E| = m

Eccentricity, Radius & Diameter

- The **distance** d(u,v) between two vertices is the (weighted) length of the shortest path between them

- The **eccentricity** of a vertex v, $e(v)$, is its maximum distance to any other vertex

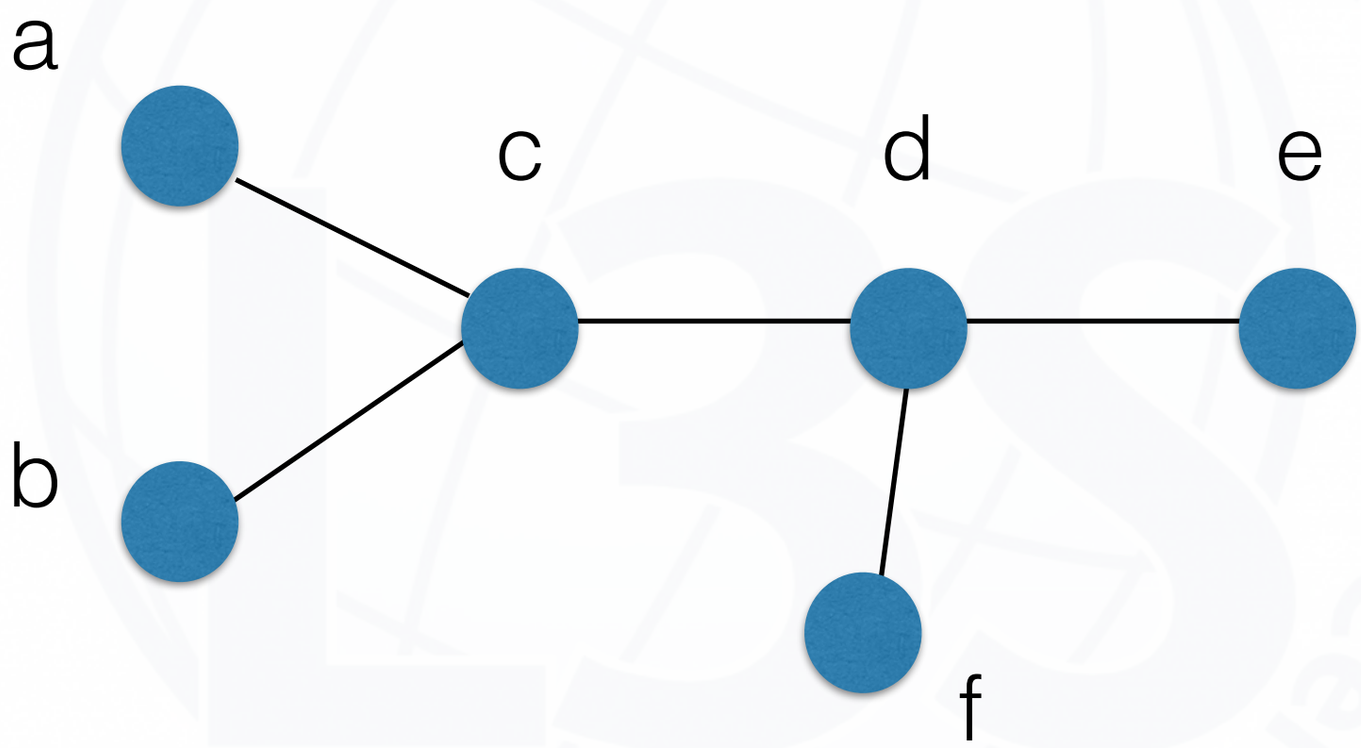
- The **radius** of a connected graph, $r(G)$, is the minimum eccentricity of any vertex

- The **diameter** of a connected graph, $d(G)$, is the maximum eccentricity of any vertex

- The **effective diameter** of a graph is smallest number that is larger than the eccentricity of a large fraction of the vertices in the graph

Eccentricity, Radius & Diameter

- What is the sum of outdegrees of all nodes in a graph G (n,m)?



- Find the eccentricity (of nodes a,c), radius and diameter of graph

Centrality

- The more central a node the faster it can reach other notes
  - “Every actor is related to Kevin Bacon by no more than 6 hops ”

- Measures for centrality
  - Degree and Eccentricity centrality
  - Closeness centrality
  - Betweenness centrality

Degree and Eccentricity centrality

Centrality is a function c: $V \rightarrow$ R that induces a total order in V

– The higher the centrality of a vertex, the more important it is

- In **degree centrality** $c(v_i) = d(v_i)$, the degree of the vertex

- In **eccentricity centrality** the least eccentric vertex is the most central one, $c(v_i) = 1/e(v_i)$
  – The least eccentric vertex is *central*
  – The most eccentric vertex is *peripheral*

Closeness centrality

- In **closeness centrality** the vertex with least distance to *all other* vertices is the centre

closeness centrality $c(x)$ of a vertex x

$$c(x) = \frac{1}{\sum_{y \neq x \in V} d(y,x).}$$

- In eccentricity centrality we aim to minimize the maximum distance

- In closeness centrality we aim to minimize the average distance
  – This is the distance used to measure the centre of Hollywood

Betweenness centrality

- The **node betweenness centrality or betweenness** measures the number of shortest paths that travel through a node/vertex x

- Can also be defined for edges — **edge betweenness**

- Edges with high edge betweenness are called weak ties

betweenness centrality $b(x)$ of a vertex x:

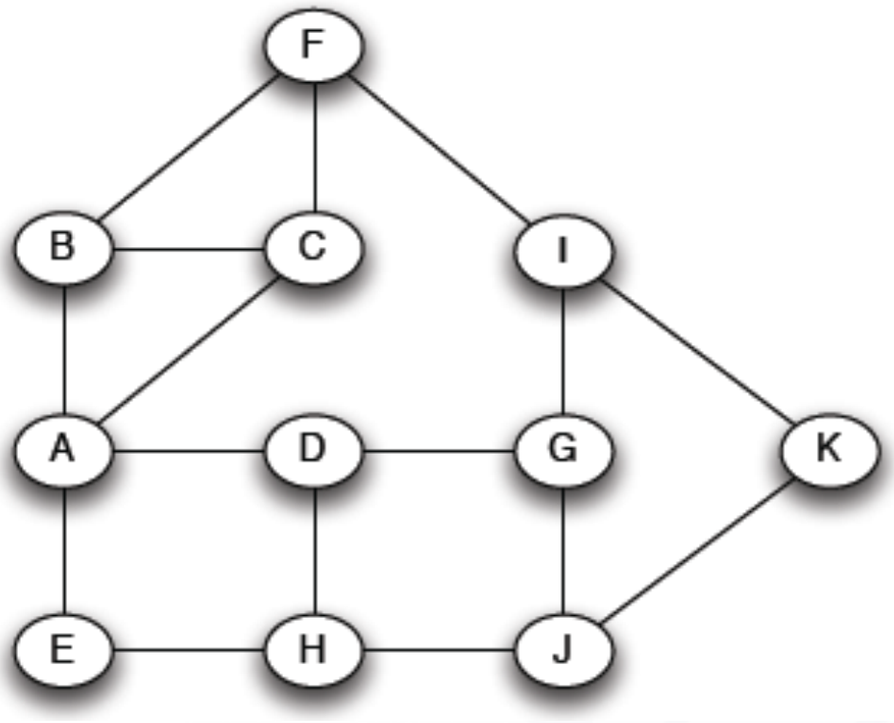
$$b(x) = \sum_{\substack{s \neq x \neq t \in V \\ s \neq t}} \frac{\sigma_{st}(x)}{\sigma_{st}}$$

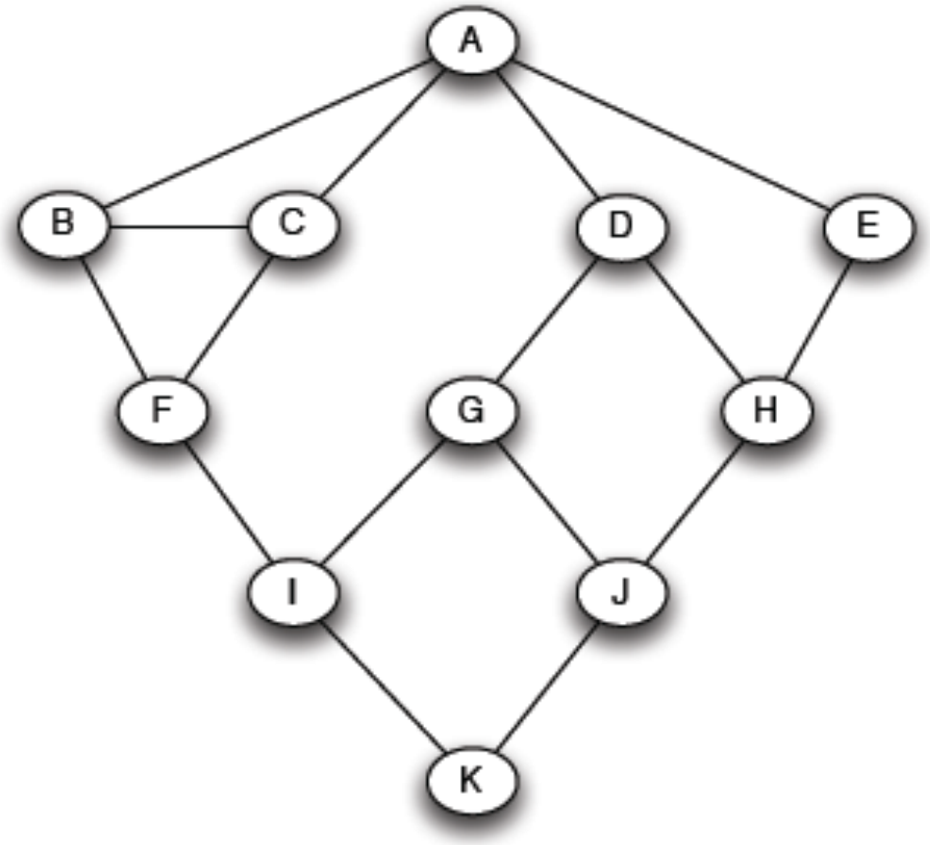
σ_{st}: number of SPs from s to t

$\sigma_{st}(x)$: how many of them pass through x

How to Compute Betweenness?

Start BFS from A if you need to compute betweenness of paths starting from A





0

1

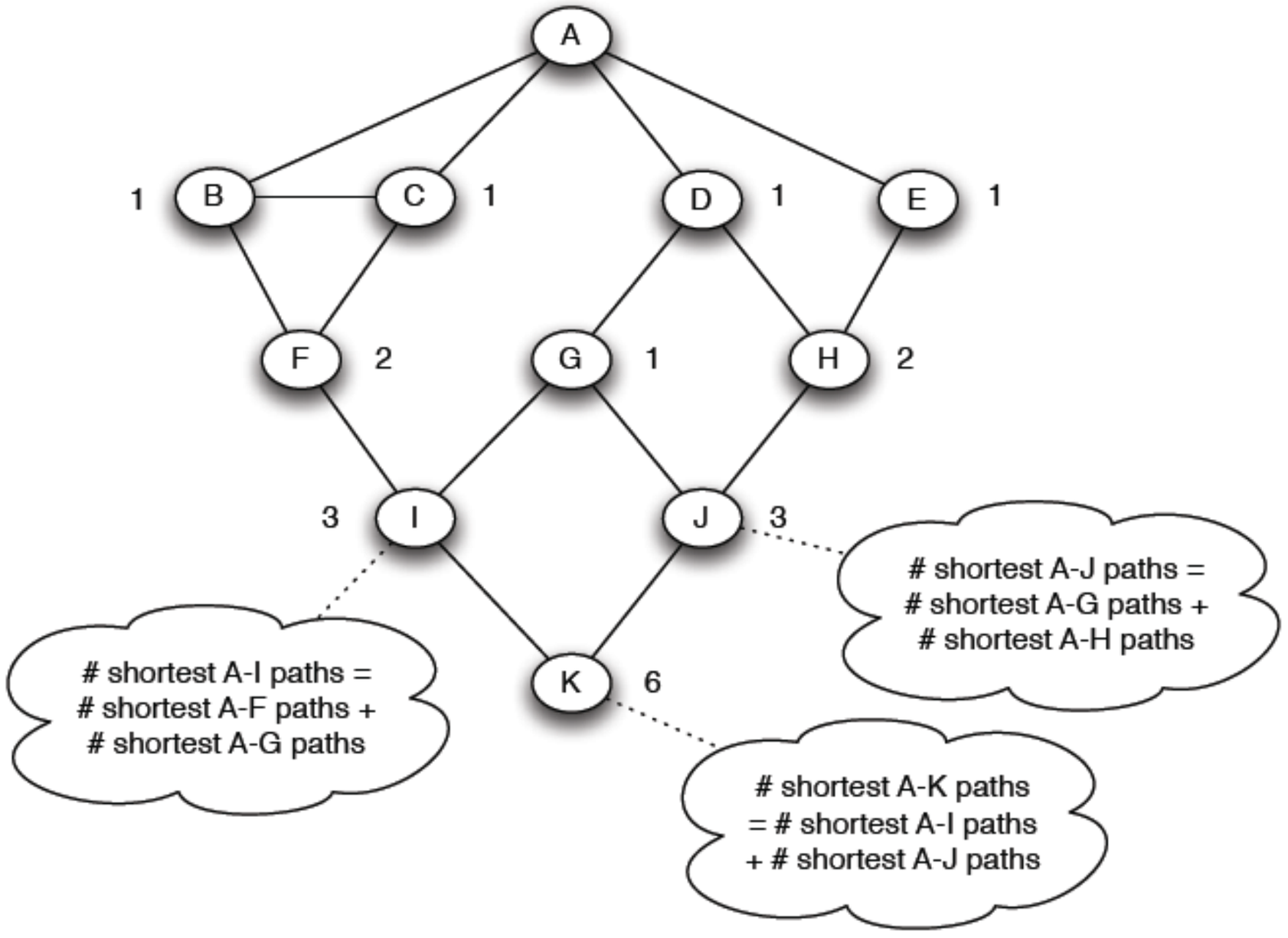
2

3

4

How to Compute Betweenness?

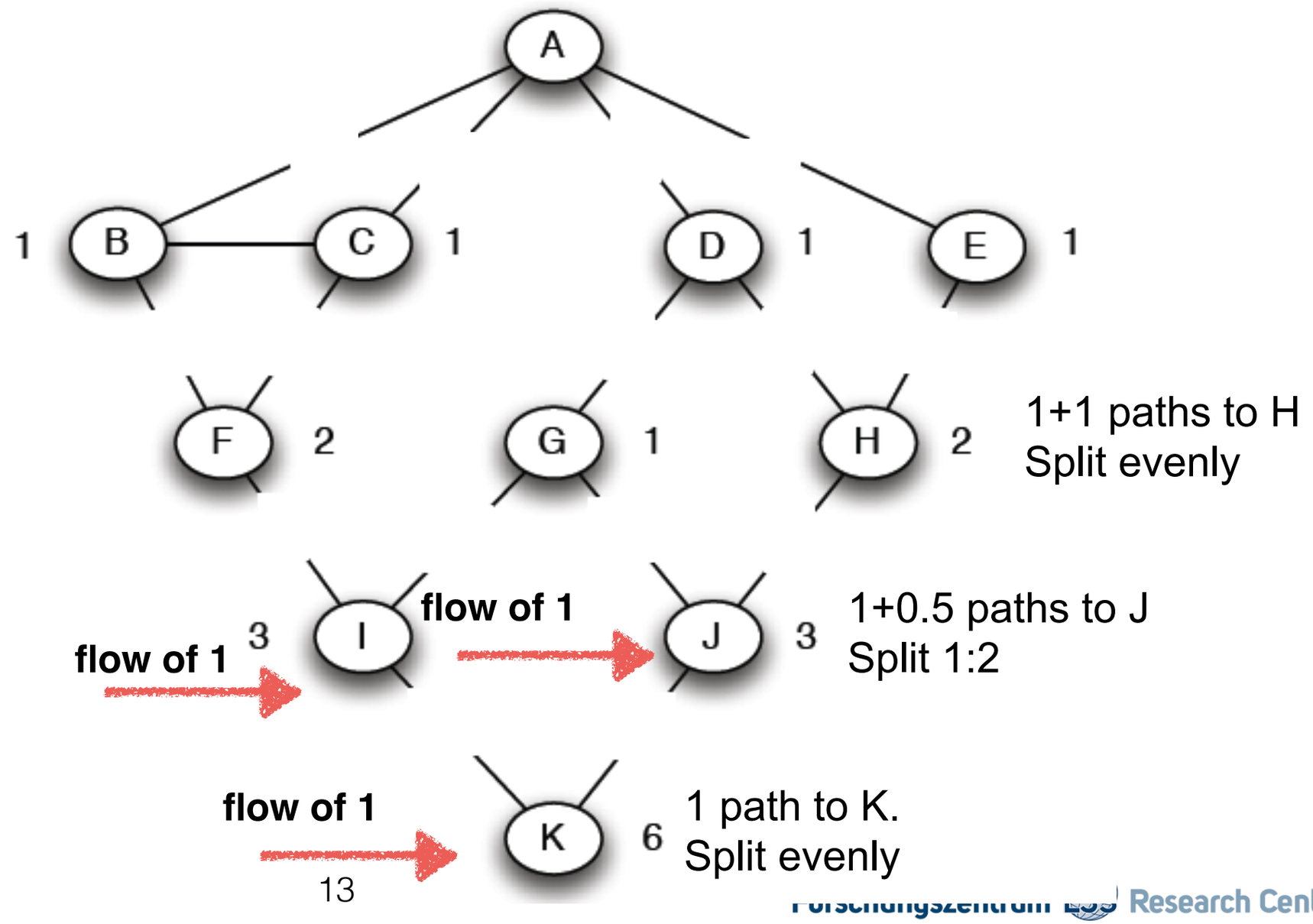
Count the number of shortest paths from A to all other nodes in the network



How to Compute Betweenness?

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

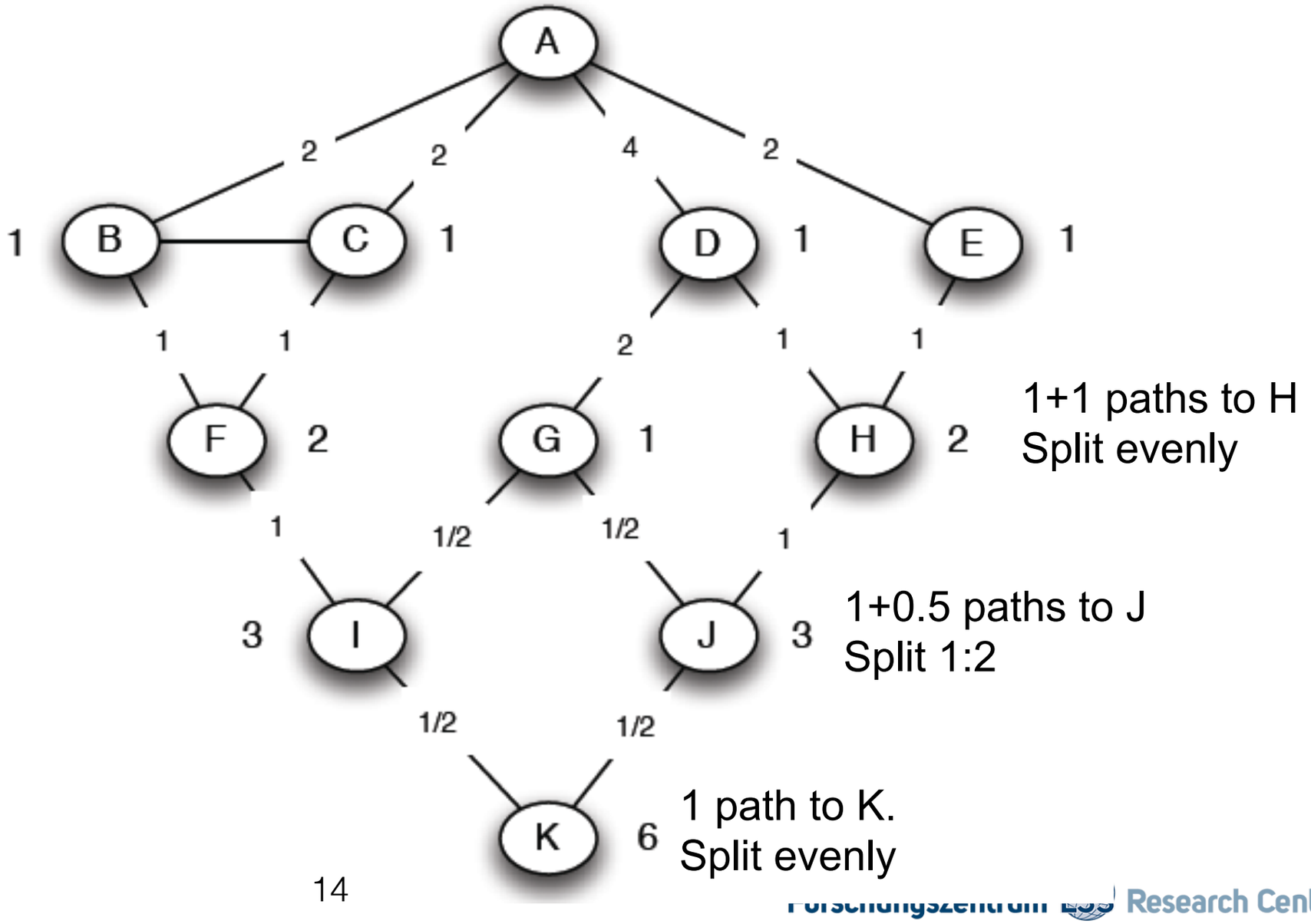
The algorithm:
- Add edge **flows**:
  -- node flow = 1+∑child edges
  -- split the flow up based on the parent value
- Repeat the BFS procedure for each starting node U



How to Compute Betweenness?

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

The algorithm:
- Add edge **flows**:
  -- node flow = 1+∑child edges
  -- split the flow up based on the parent value
- Repeat the BFS procedure for each starting node U



Graph Properties

- Several real-world graphs exhibit certain characteristics
  - Studying what these are and explaining why they appear is an important area of network research
- As data miners, we need to understand the consequences of these characteristics
  - Finding a result explained merely by one of these characteristics is not interesting
- We also want to *model* graphs with these characteristics

Small World Property

- A graph G is said to exhibit a **small-world property** if its average path length scales logarithmically
  - The six degrees of Kevin Bacon is based on this property
  - How far a mathematician is from Hungarian combinatorist Paul Erdős
  - A radius of a large, connected mathematical co-authorship network (268K authors) is 12 and diameter 23

Scale Free Property

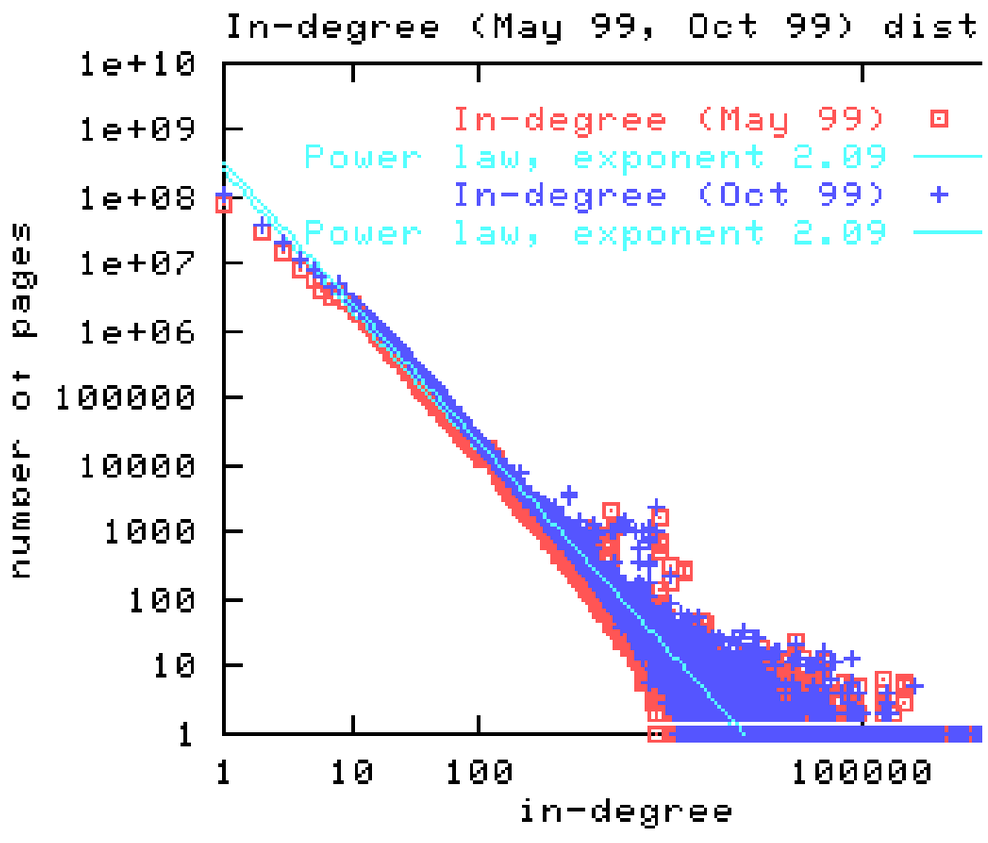
- The **degree distribution** of a graph is the distribution of its vertex degrees
  - How many vertices with degree 1, how many with degree 2, etc.
  - $f(k)$ is the number of vertices with degree k

- A graph is said to exhibit **scale-free property** if $f(k) \propto k^{-\gamma}$
  - So-called power-law distribution
  - Majority of vertices have small degrees, few have very high degrees

- Scale-free: $f(ck) = \alpha(ck)^{-\gamma} = (\alpha c^{-\gamma})k^{-\gamma} \propto k^{-\gamma}$

Example- WWW

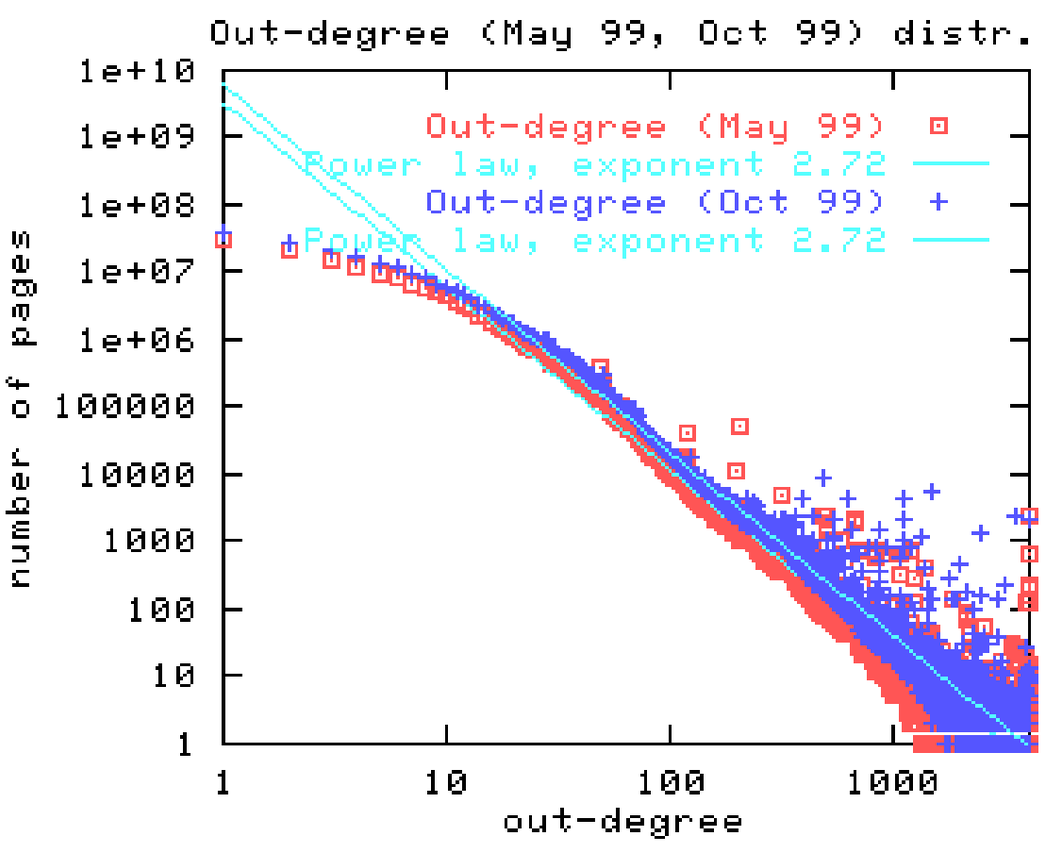
In-degree



Broder et al. *Graph structure in the web*. WWW'00

$s = 2.09$

Out-degree

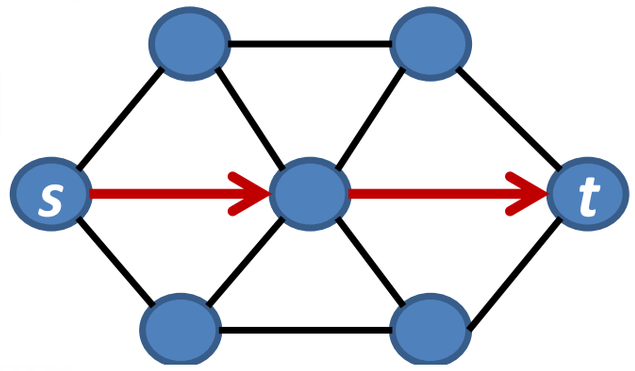


$s = 2.72$

Shortest Paths on large graphs

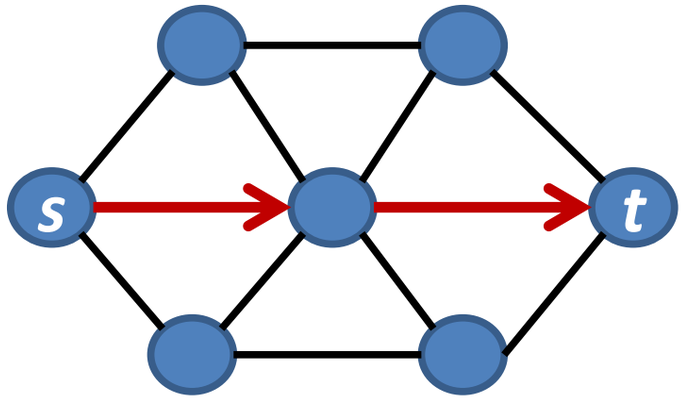
Shortest Paths on Graphs

- Find the shortest path/distance between a source node **s** and a destination node **d**
- Assuming we have a unweighted undirected graph G
  - What is complexity if the entire graph is in memory ?
  - What if the graph does not fit in memory ?
  - What if an application wants to support multiple queries



- **We need methods to precompute distances to support multiple queries later**

BFS vs Transitive Closure

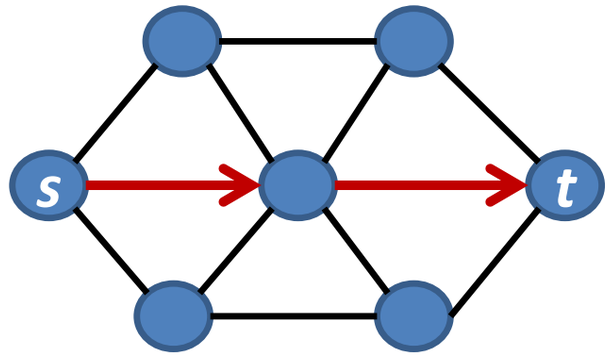


- BFS for Shortest path computation (u)
  - Maintain a queue **q**
  - Update distance for the head of the queue **h**
    - For **h** visit all **unvisited** neighbors and add them to the queue
    - Iterate until the queue is empty
- **Transitive Closure:** Find all pairs shortest paths
  - needs quadratic space and computations

Indexing for distances

Given a graph $G = (V, E)$

1. construct an index to
2. answer distance $d_G(s, t)$



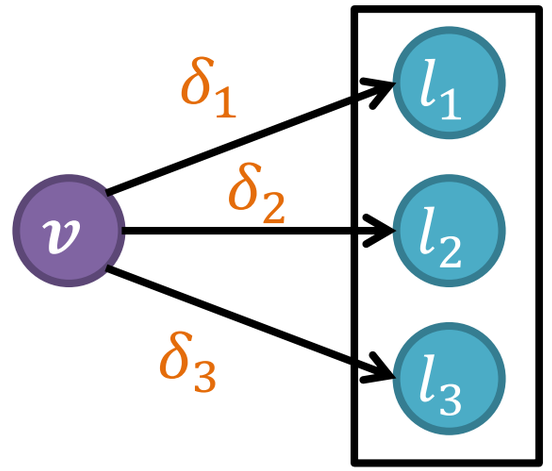
Goal: Good trade-off

Scalability
Indexing time
Index size

Query Performance
Query time
Precision

2-Hop Labelling

- Index: label $L(v) = \{(l_1, \delta_1),\ (l_2, \delta_2\ ),\ \ldots\}$
  - $l_i \in V,\ \delta_i = d_G(v, l_i)$



Example

$L(1)$:

Vertex	1	4	5	7	10
Distance	0	3	2	4	5

$L(2)$:

Vertex	2	4	6	12
Distance	0	1	5	3

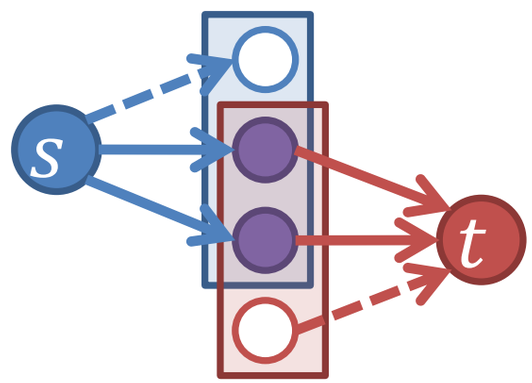
$L(3)$:

Vertex	2	3	4	6	7
Distance	5	0	4	7	2

$d_G(1,10) = 5$

2-Hop Labelling - Querying

- Query: $d_G(s,t) = \min_{l \in L(s) \cap L(t)} d_G(s,l) + d_G(l,t)$
  - Paths through common vertices



Example

$L(1)$:

Vertex	1	4	5	7	10
Distance	0	3	2	4	5

$L(3)$:

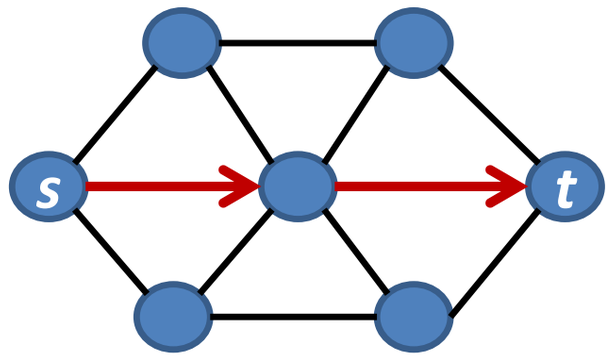
Vertex	2	3	4	6	7
Distance	5	0	4	7	2

Distance between vertex **1** and **3**：

- **1 –…– 4 –…– 3** ： 3 + 4 = 7
- **1 –…– 7 –…– 3** ： 4 + 2 = 6

} Answer min{6, 7} = **6**

Challenges



- How do we compute the labels for each node ?
  - *correctness* : each pair should be covered
  - *index size*: size of the labels or index size should be reasonable small
  - *scalability and efficiency*: the indexing time and the Query processing time should be small
- Idea: Use Landmark Labelling

Landmark Labelling (naiive)

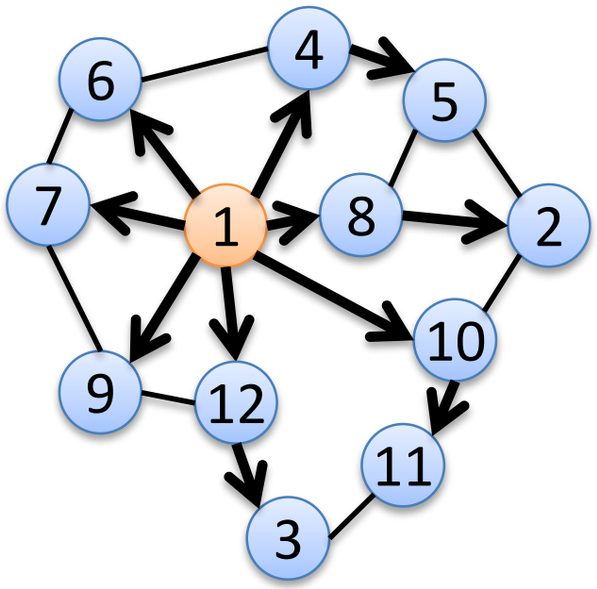
1. $L_0 \leftarrow$ an empty index
2. For each vertex $v_1, v_2, \ldots, v_n$
   - Conduct a BFS from v_i
   - Label all the visited vertices
     - $L_i(u) = L_{i-1}(u) \cup \left(v_i, d_G(u, v_i)\right)$

|V| = n

|E| = m

$O(nm)$ indexing & space, $O(n)$ query

Landmark Labelling



After a BFS from 1

$L_1(1)$:

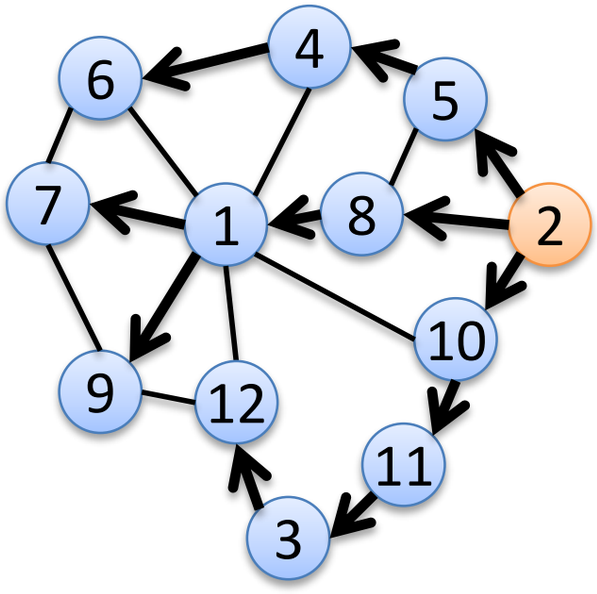
Vertex	1
Distance	0

$L_1(2)$:

Vertex	1
Distance	2

$L_1(3)$:

Vertex	1
Distance	2



After a BFS from 2

$L_2(1)$:

Vertex	1	2
Distance	0	2

$L_2(2)$:

Vertex	1	2
Distance	2	0

$L_2(3)$:

Vertex	1	2
Distance	2	3

Landmark Labelling (naiive)

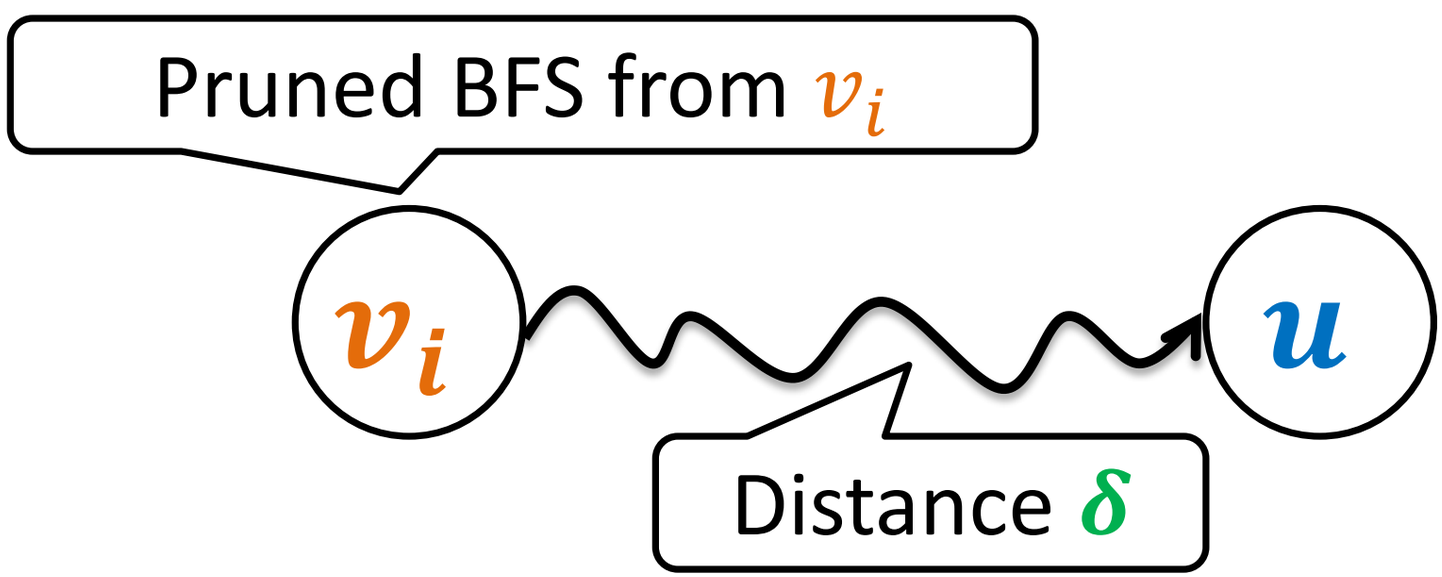
- Both indexing and QP is expensive
- We do not need to maintain all reachable nodes from a given node but
  - partial information in nodes should be complete
  - overall labels should be small

Pruned Landmark Labelling

1. $L'_0 \leftarrow$ an empty index
2. For each vertex $v_1, v_2, \ldots, v_n$
   - Conduct a pruned BFS from v_i
   - Label all the visited vertices
     - $L'_i(u) = L'_{i-1}(u) \cup \left(v_i, d_G(u, v_i)\right)$

[Akiba+ ’10]

Pruned BFS

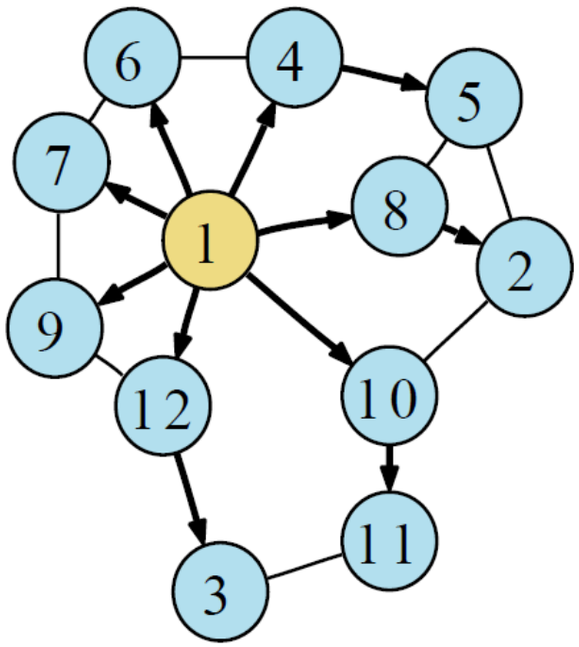


Current incomplete index

If QUERY$(v_i, u, L'_{i-1}) \leq \delta \rightarrow$ **Prune** u

- We do not label u this time
- We do not traverse edges from u

Pruned BFS - Example



First BFS from vertex 1

$L'_1(1)$:

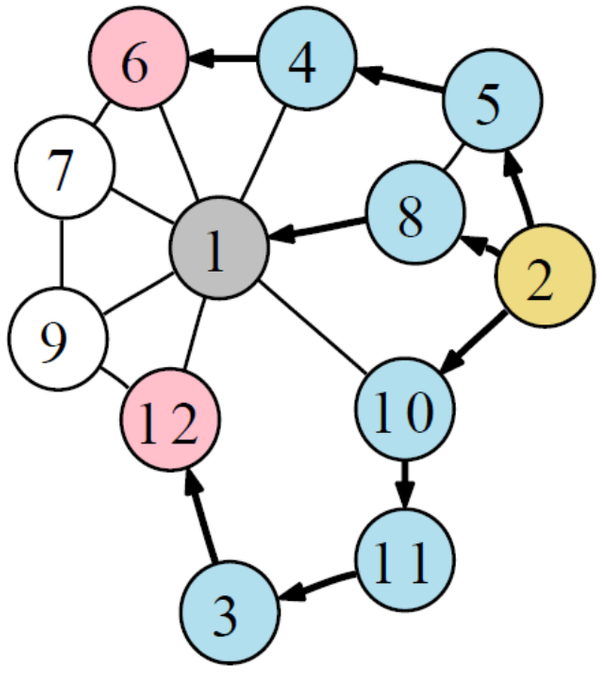
Vertex	1
Distance	0

$L'_1(2)$:

Vertex	1
Distance	2

$L'_1(6)$:

Vertex	1
Distance	1

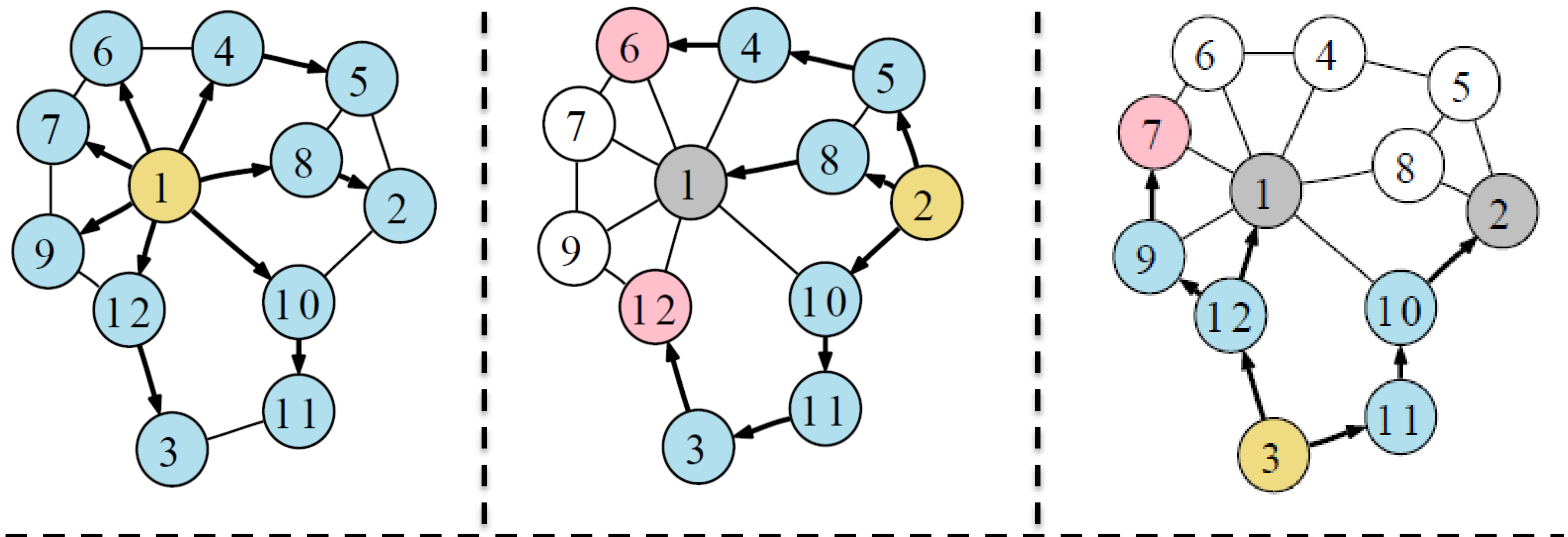


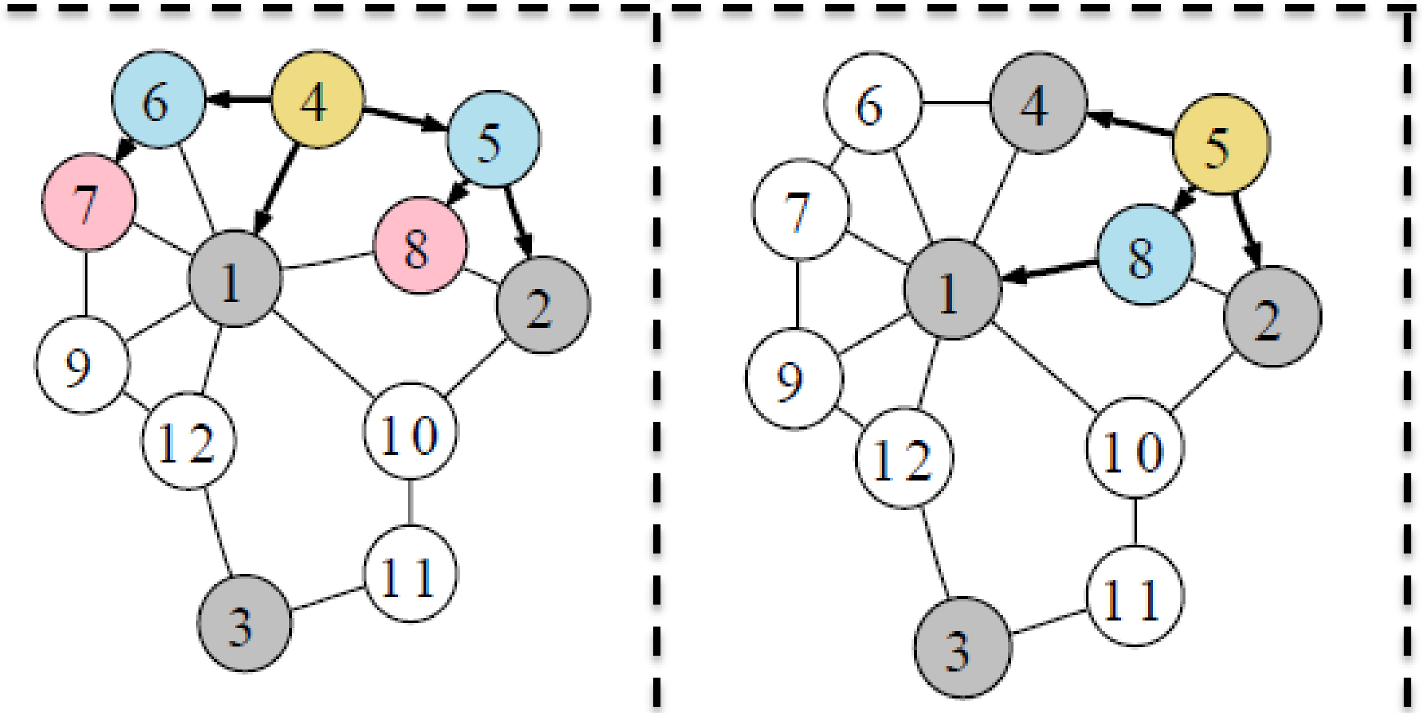
Second BFS from vertex 2

$\text{QUERY}(2, 6, L'_1) = 2 + 1 = 3 = d(2,6)$

→ Vertex 6 is pruned.

Pruned BFS - Example



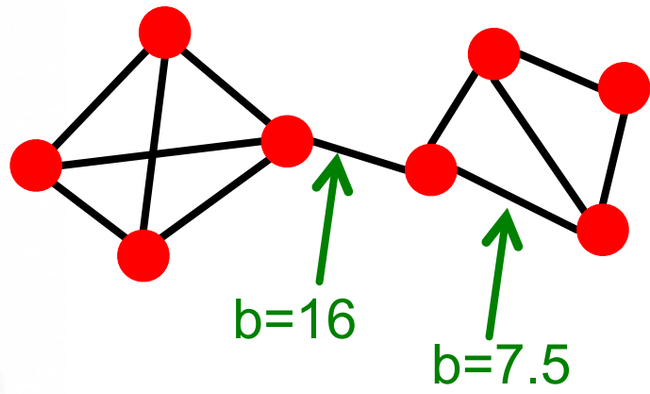


The search space gets smaller and smaller

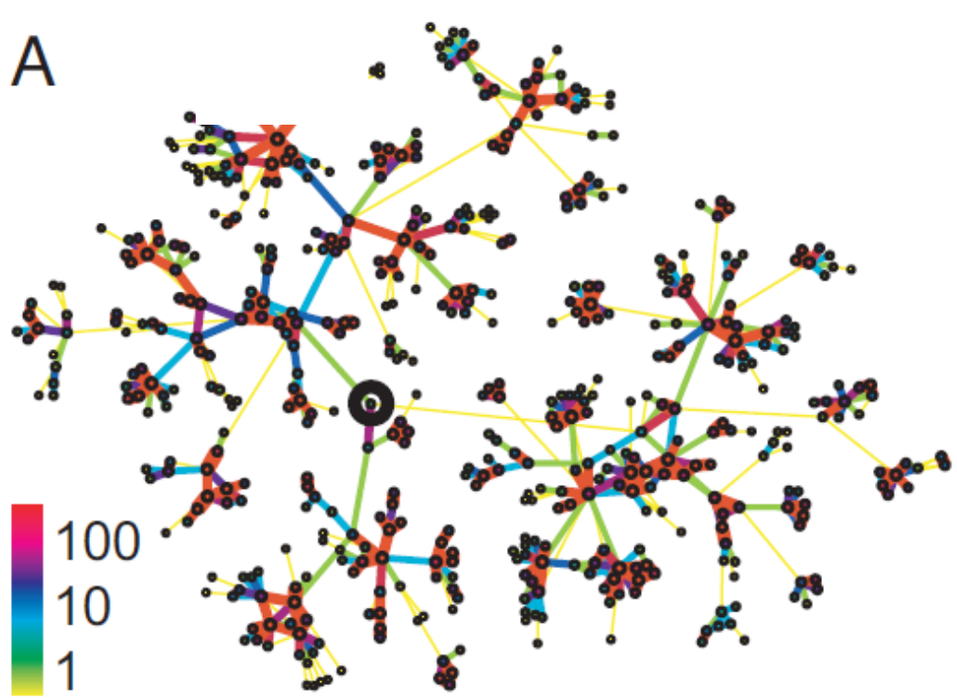
Community Detection by Graph Clustering

Method 1: Strength of Weak Ties

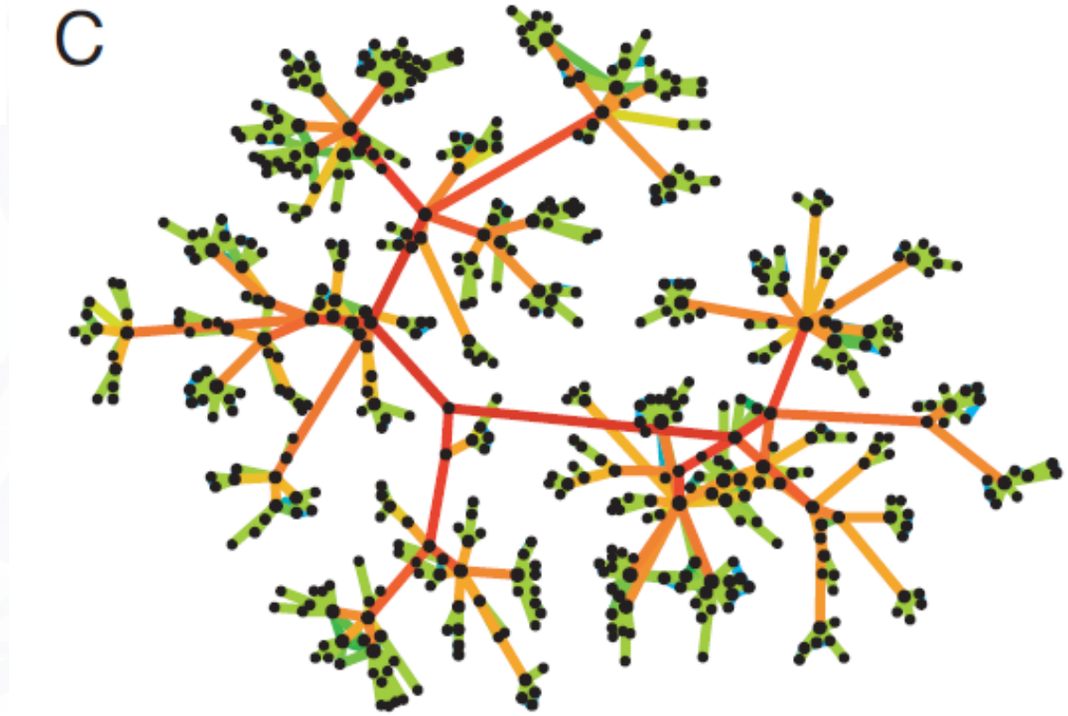
- **Edge betweenness: Number of shortest paths passing over the edge**



- **Intuition:**



Edge strengths (call volume) in a real network



Edge betweenness in a real network

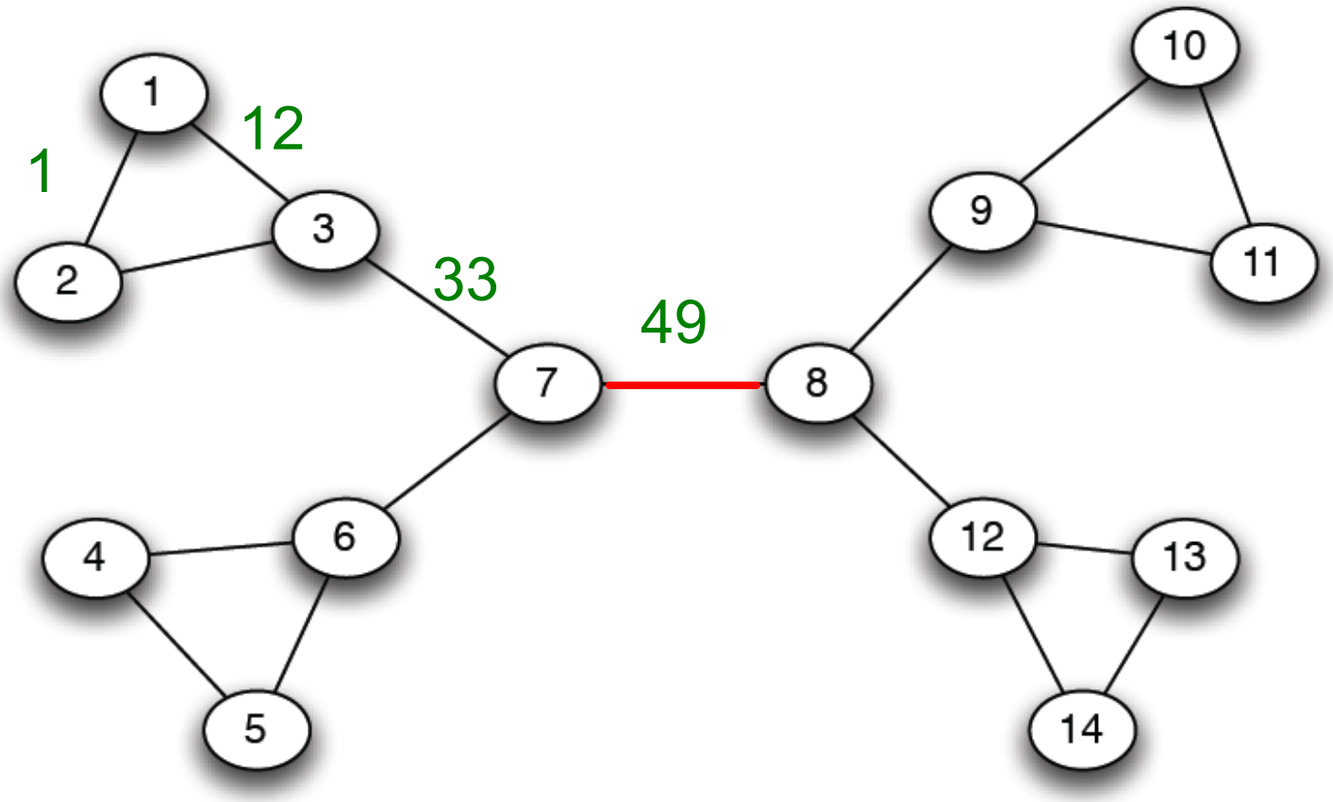
[Girvan-Newman '02]

Method 1: Girvan-Newman

- Divisive hierarchical clustering based on the notion of edge **betweenness**:

- **Girvan-Newman Algorithm:**
  - **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

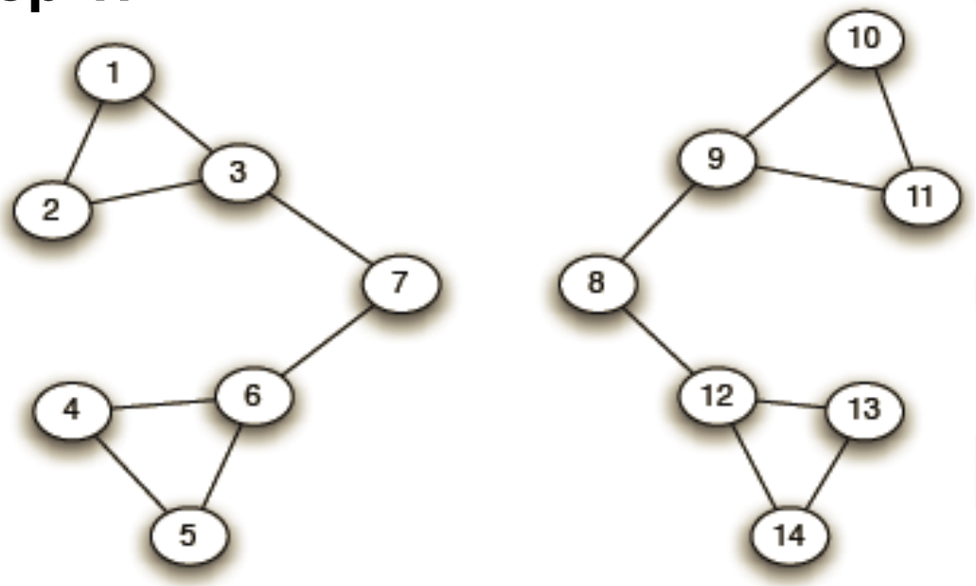
Girvan-Newman: Example



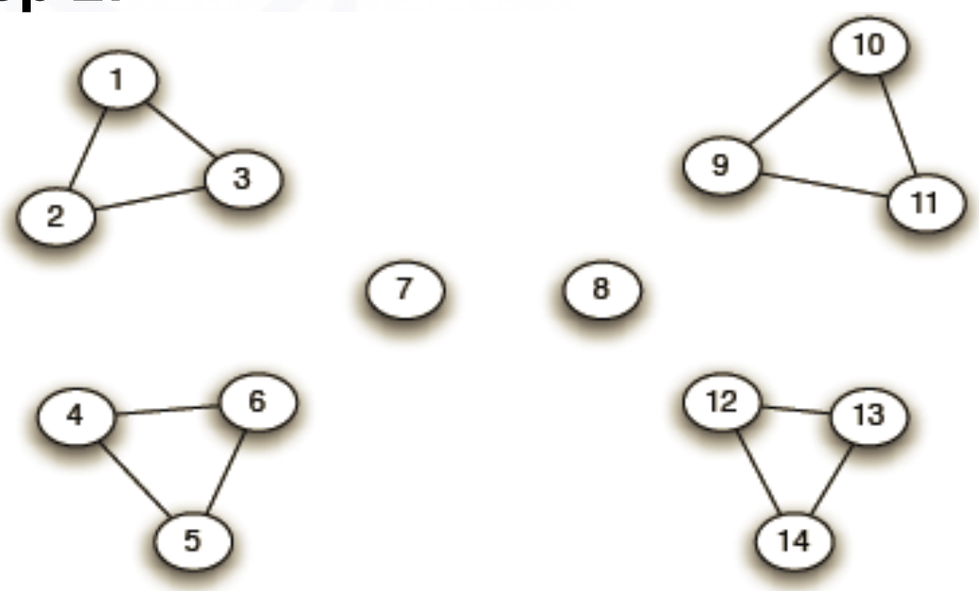
Need to re-compute betweenness at every step

Girvan-Newman: Example

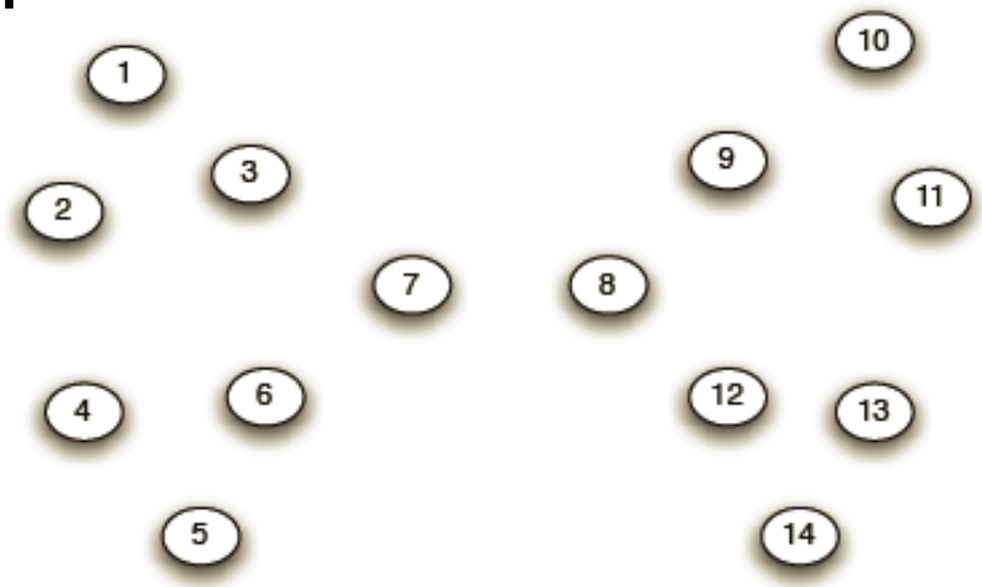
Step 1:



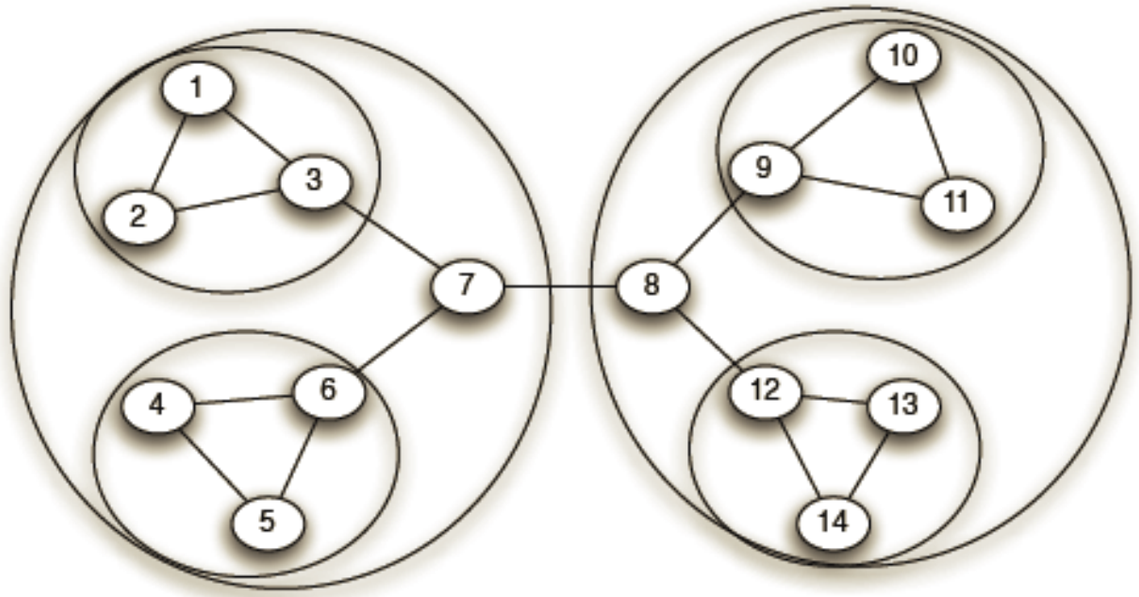
Step 2:



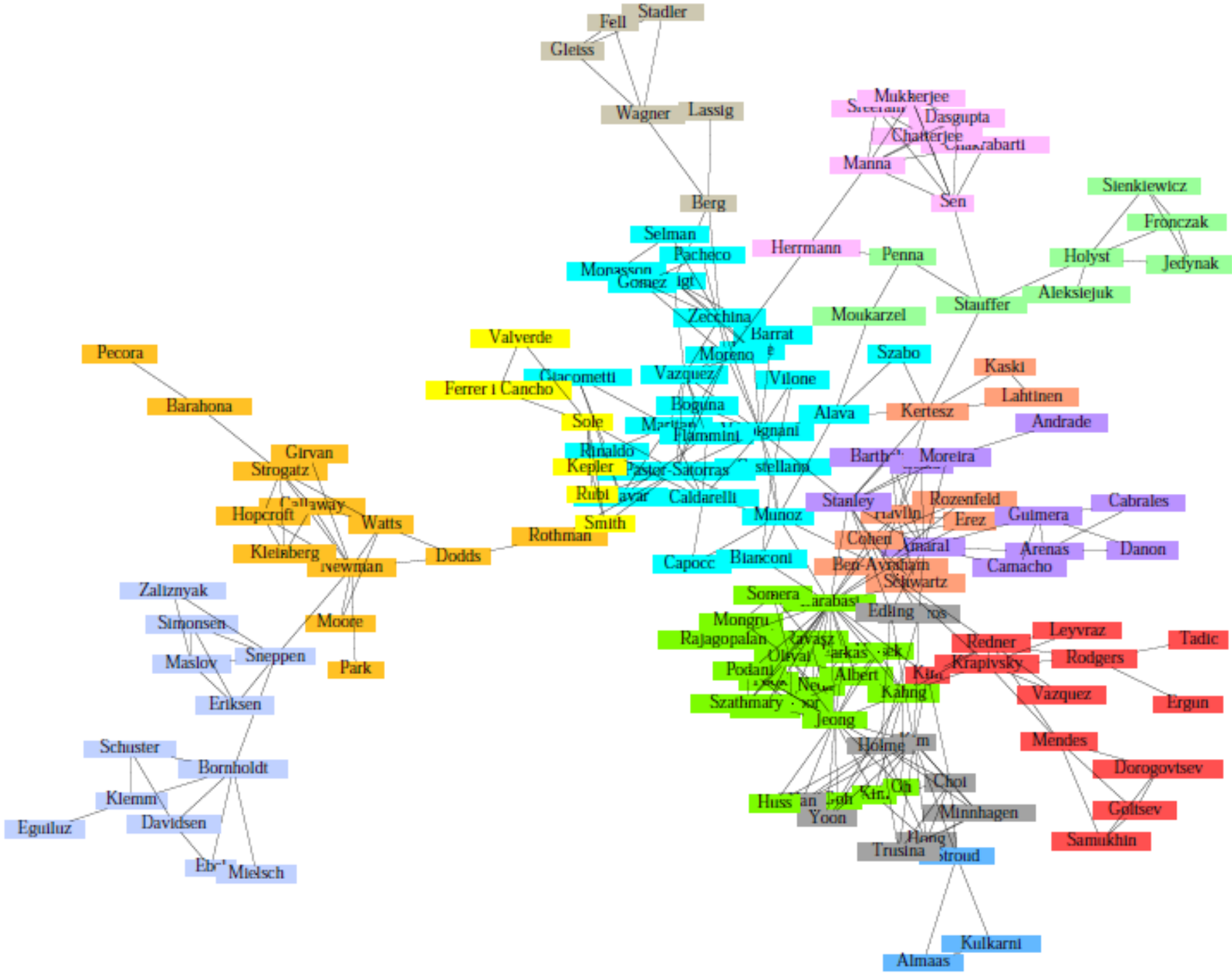
Step 3:



Hierarchical network decomposition:

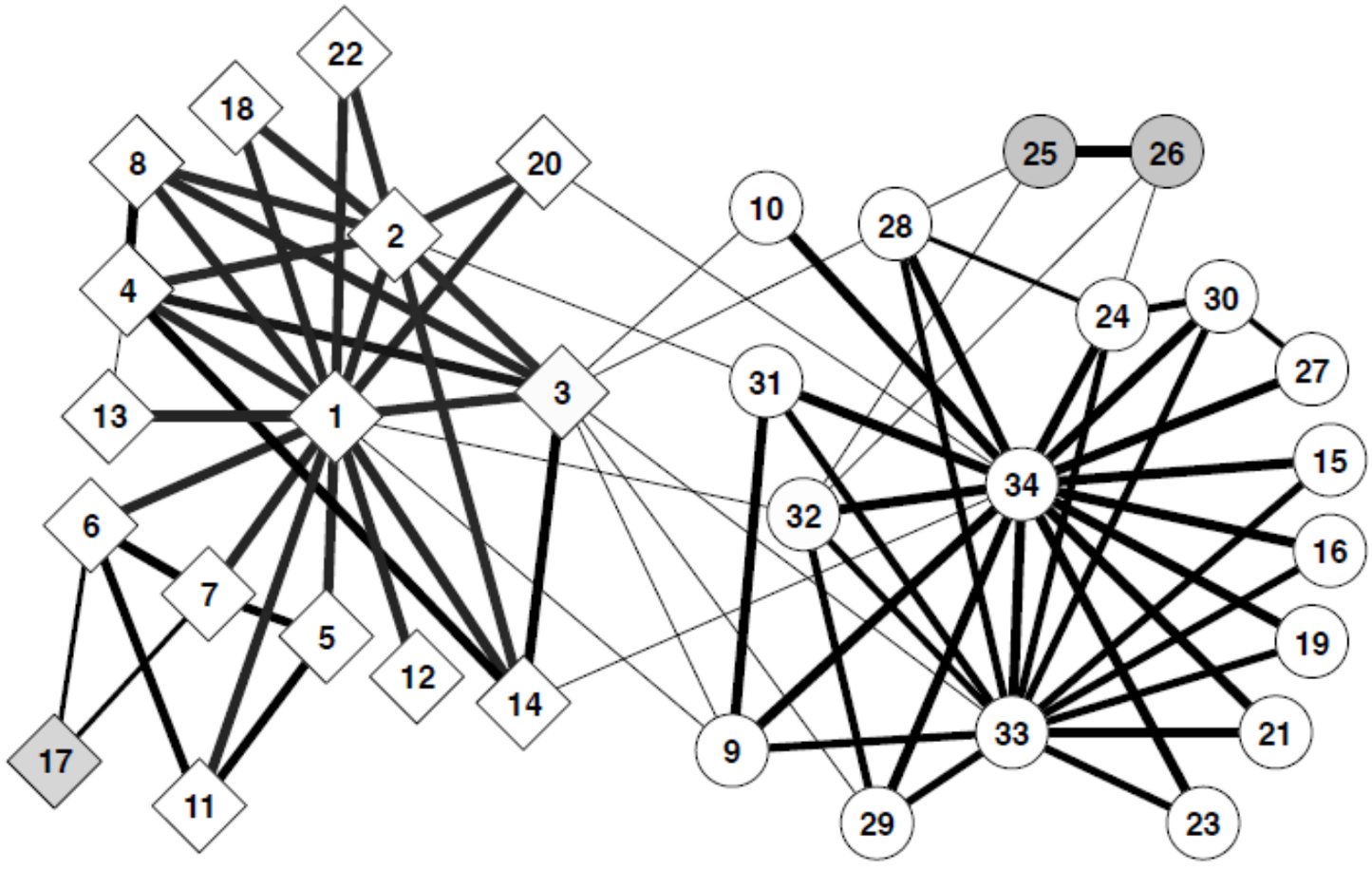


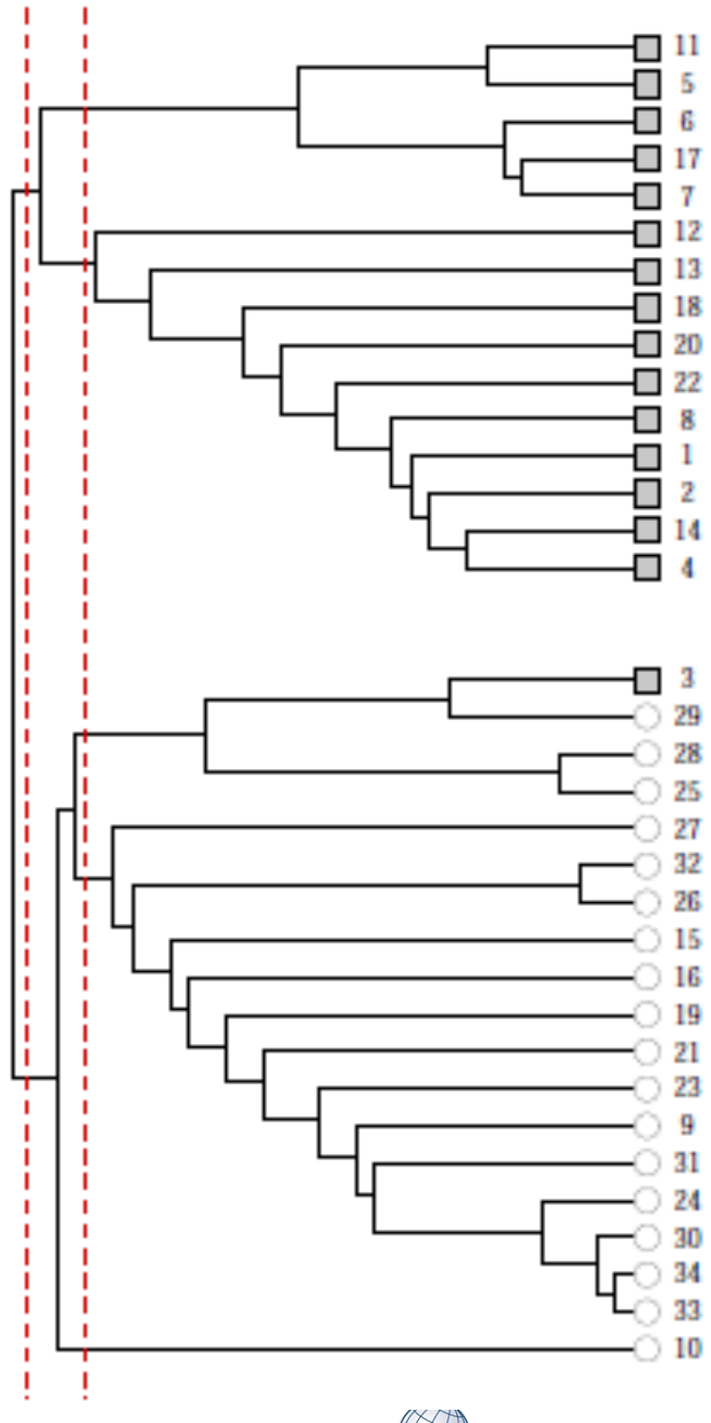
Girvan-Newman: Results



Communities in physics collaborations

Girvan-Newman: Results





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

Girvan-Newman

- How do we compute Betweenness efficiently ?

we did this earlier

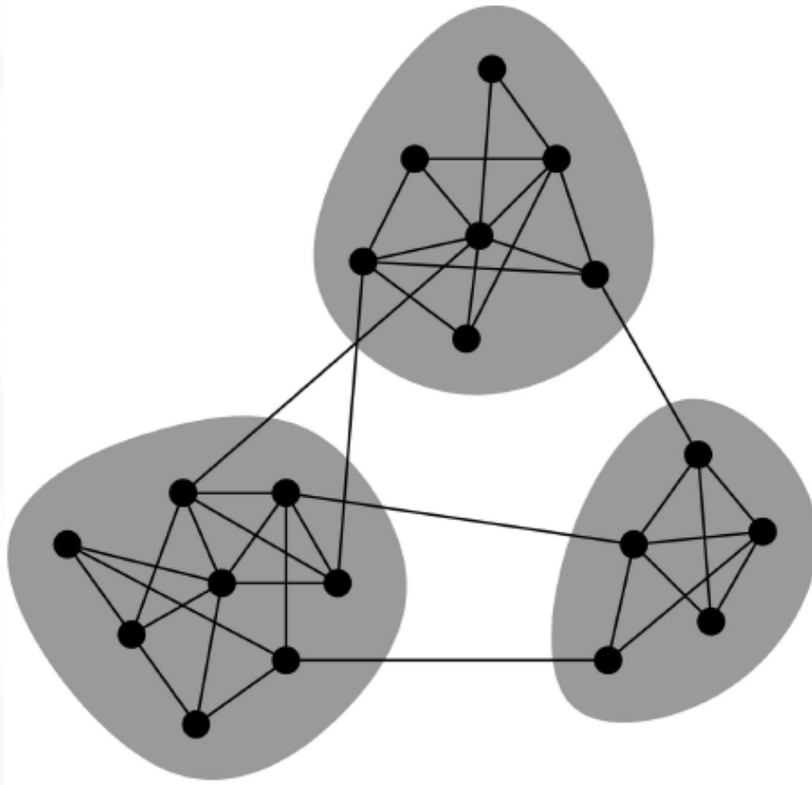
- When do we stop aggregating ?

modularity

Network Communities

- Communities: sets of tightly connected nodes

- Modularity : A measure of how well a network is partitioned into communities



Given a partitioning of the network into groups $s \in S$:

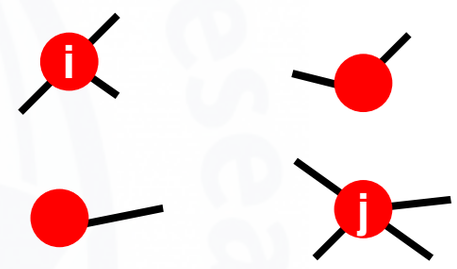
$$Q \propto \sum_{s \in S} [\ (\#\ \text{edges within group } s) - (\text{expected } \#\ \text{edges within group } s)\ ]$$

Need a null model!

Null Model: Configuration Model

Given real G on n nodes and m edges, construct rewired network G'

- Same degree distribution but random connections
- Consider G' as a multigraph



What is the probability that a node with outdegree (o.d) **k** is connected with another node (a) with o.d **1** (b) with o.d **p** ?

What is the expected number of edges between them ?

Null Model: Configuration Model

- **The expected number of edges between nodes i and j** of degrees k_i and k_j equals to: $k_i \cdot \frac{k_j}{2m} = \frac{k_i k_j}{2m}$
  - The expected number of edges in (multigraph) **G'**:
    - $= \frac{1}{2} \sum_{i \in N} \sum_{j \in N} \frac{k_i k_j}{2m} = \frac{1}{2} \cdot \frac{1}{2m} \sum_{i \in N} k_i \left(\sum_{j \in N} k_j \right) =$
    - $= \frac{1}{4m} 2m \cdot 2m = m$

Note:
$$\sum_{u \in N} k_u = 2m$$

Modularity

Modularity of partitioning S of graph G:

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

Normalizing cost.: -1<Q<1

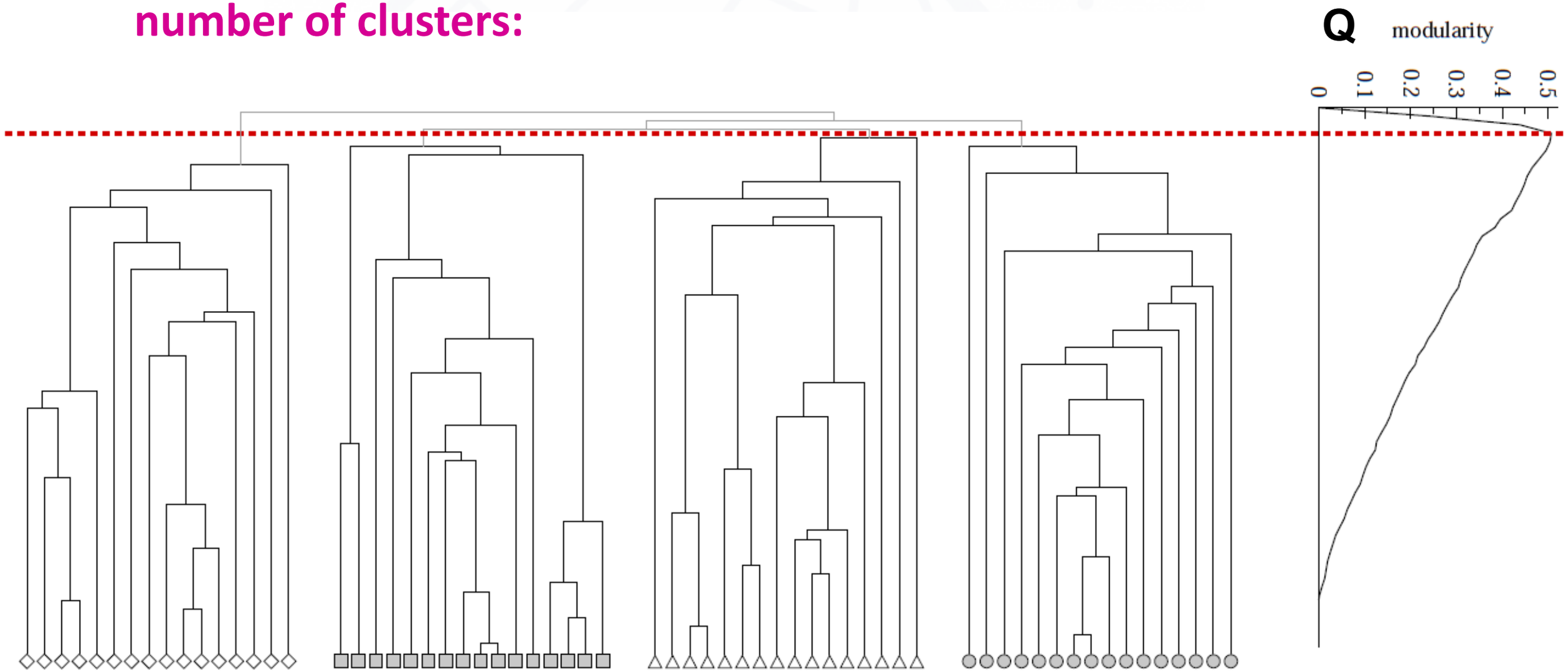
A_{ij} = 1 if i→j,
0 else

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: Number of clusters

- **Modularity is useful for selecting the number of clusters:**



Summary

- **Graphs are everywhere and large**
  - **Common properties used to study and model real world graphs**
- **Distance queries: Graph indexing based**
  - **2-Hop Labelling**
  - **landmark labelling**
- **Community detection based on hierarchical graph clustering**