Objectives

Graph Partitioning

- Revisiting Betweenness Centrality
- Edge Betweenness

Graph Partitioning

- Girvan-Newman Method:
  - Divisive method Proposed by Girvan and Newman in 2002
  - Uses edge betweenness to identify edges to remove
  - Edge betweenness: Total amount of "flow" an edge carries between all pairs of nodes where a single unit of flow between two nodes divides itself evenly among all shortest paths between the nodes ($1/k$ units flow along each of k shortest paths)

Revisiting Betweenness Centrality

Basics

- This measures the number of times a given vertex u lies on a (shortest length) path between other vertices v_1 and v_2 , and gives a higher score the more times u appears.

Betweenness Centrality: Formula

$$c(v) = \sum_{s \neq v \neq t} \frac{\#\{shortest\ st - paths\ containing\ v\}}{\#\{shortest\ stpaths\}}$$

Edge Betweeness

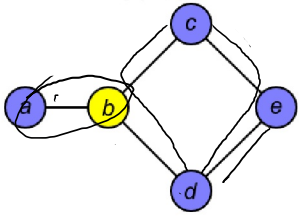
Betweeness Centrality

Betweeness Centrality of node b

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\#\{\text{shortest } st-\text{paths containing } v\}}{\#\{\text{shortest } st\text{paths}\}}$$

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

$$C_B(n) = \frac{\sum_{s \neq n \neq t} \frac{\sigma_{st}(n)}{\sigma_{st}}}{\binom{n-1}{2}} \quad // \text{ normalizing}$$



Node betweenness?

$$C_B(b) = \frac{\sigma_{ac}(b)}{\sigma_{ac}} + \frac{\sigma_{ad}(b)}{\sigma_{ad}} + \frac{\sigma_{ae}(b)}{\sigma_{ae}} + \frac{\sigma_{cd}(b)}{\sigma_{cd}} + \frac{\sigma_{ce}(b)}{\sigma_{ce}} + \frac{\sigma_{de}(b)}{\sigma_{de}}$$

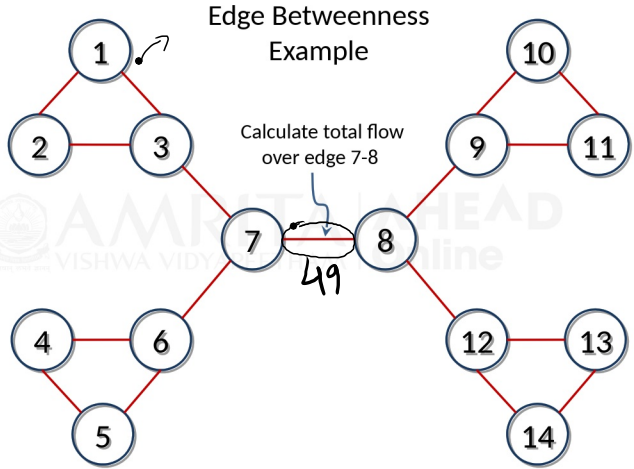
$$= \frac{1}{1} + \frac{1}{1} + \frac{2}{2} + \frac{1}{2} + \frac{0}{1} + \frac{0}{1} = 3.5$$

$$= \frac{3.5}{\binom{4}{2}} = \frac{3.5}{6}$$

Edge Betweeness as "flow"

- Assume that each node generates **1 unit** of traffic to every other node
  - c → a - 1 Unit
  - d → a - 1 Unit
  - e → a - 1 Unit
  - c → d - 0.5 Unit
- Total traffic flow through the edge $a-b$ is 3.5

EDGE-BETWEENNESS



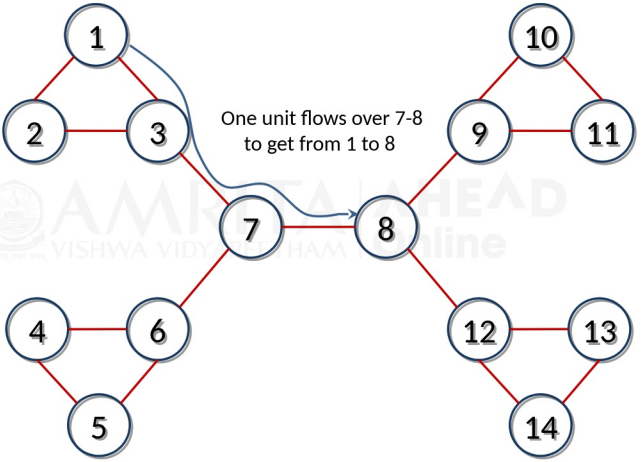
1

(1,8)
(1,9)

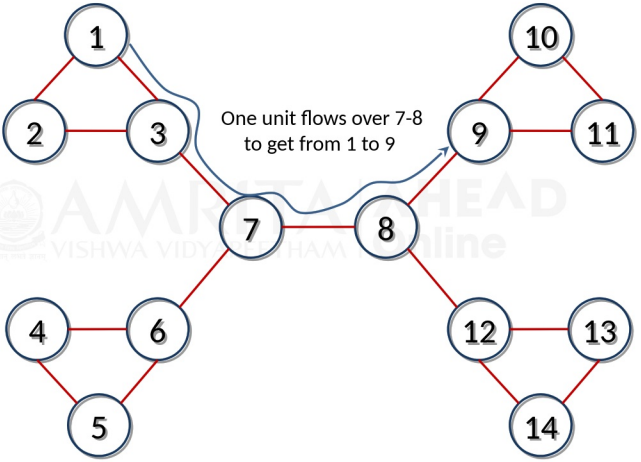
Node 1 → 7
↳ times
edge (7,8) is
in betⁿ.

Node 2 →
3
4
5
8

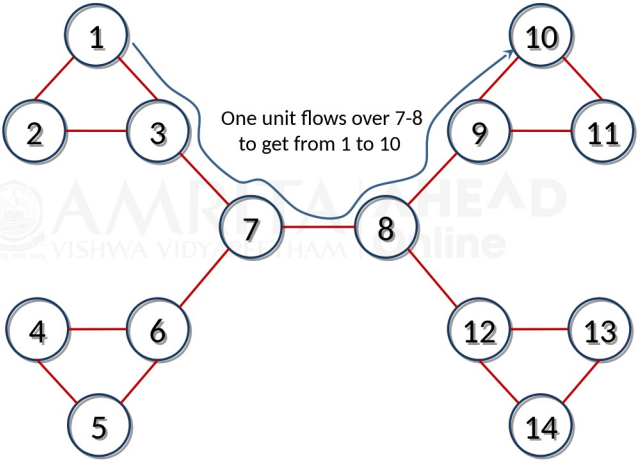
EDGE-BETWEENNESS ..



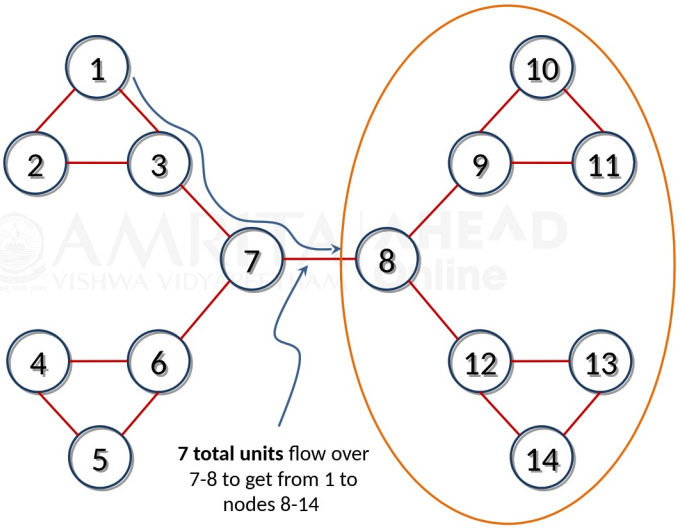
EDGE-BETWEENNESS ..



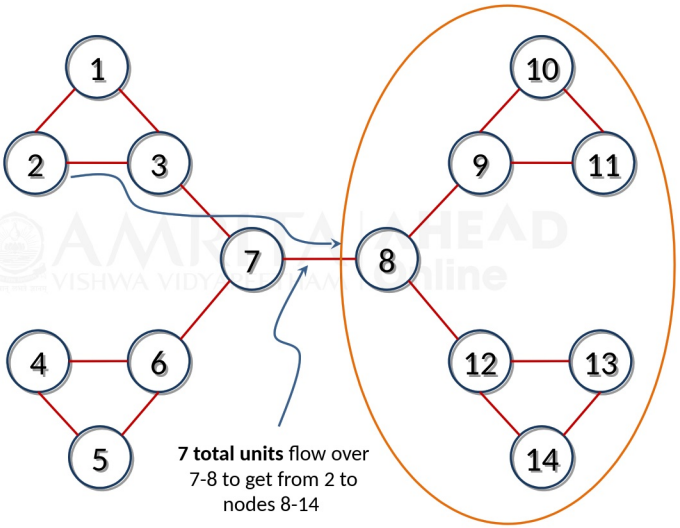
EDGE-BETWEENNESS ..



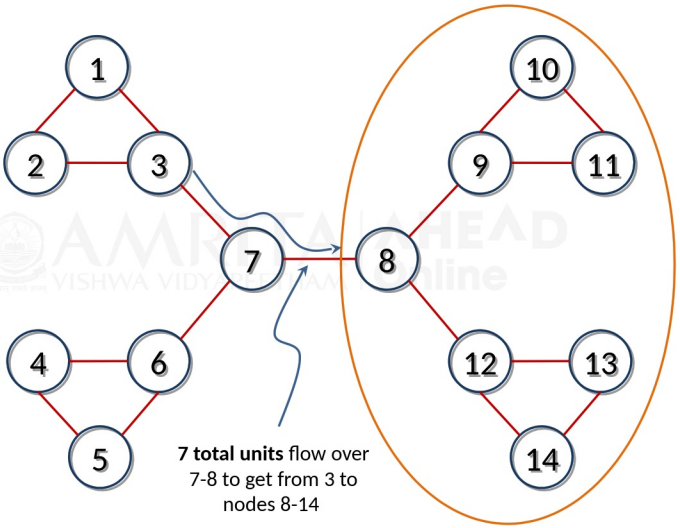
EDGE-BETWEENNESS ..



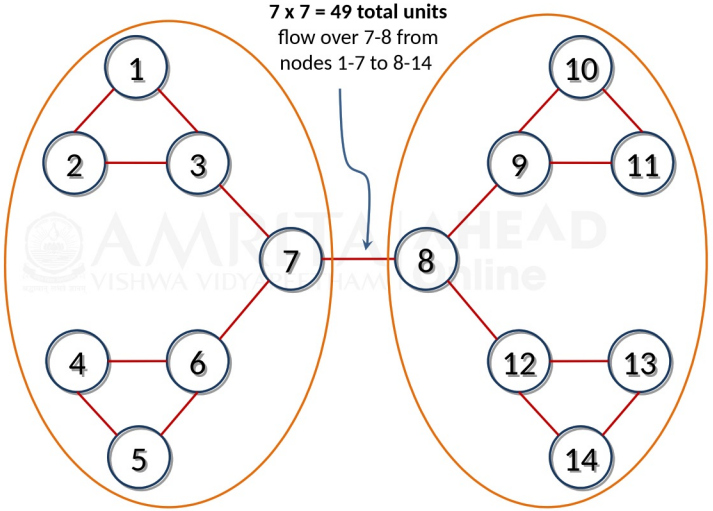
EDGE-BETWEENNESS ..



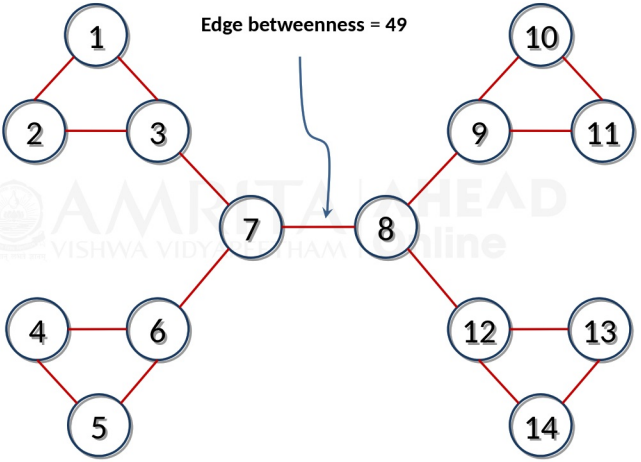
EDGE-BETWEENNESS ..



EDGE-BETWEENNESS ..

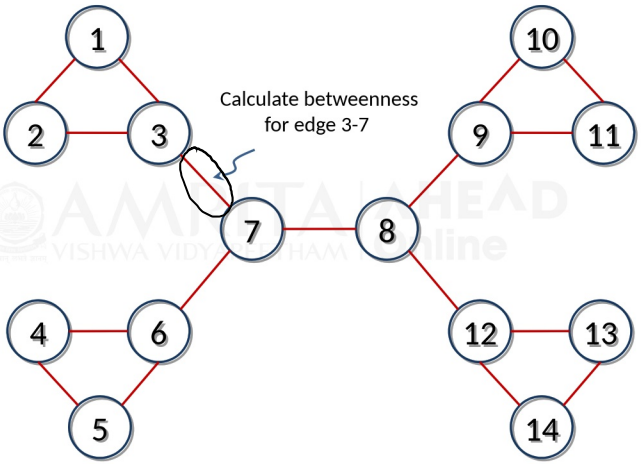


EDGE-BETWEENNESS ..

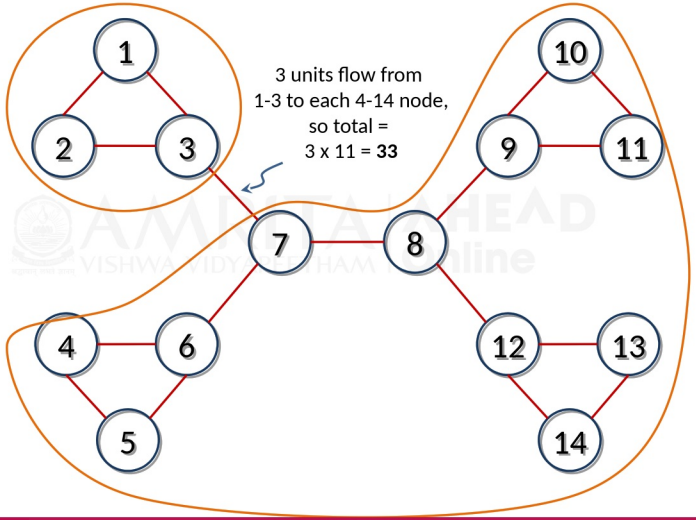


EDGE-BETWEENNESS ..

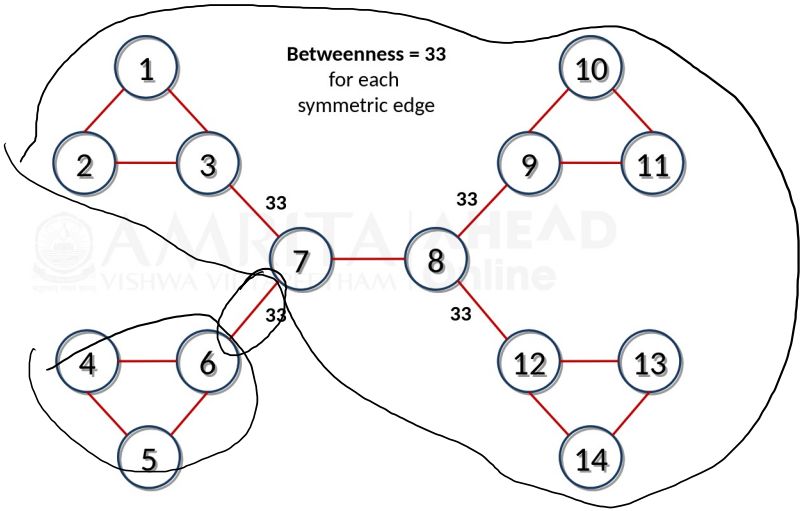
(1, 7)
(1, 6)
(1, 5)
(1, 4)
(1, 8)



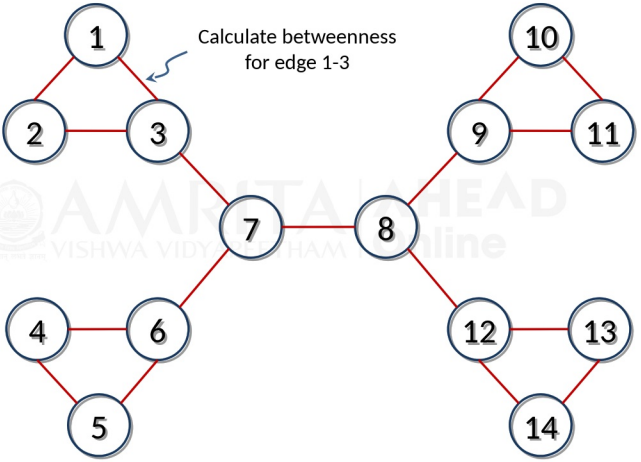
EDGE-BETWEENNESS ..



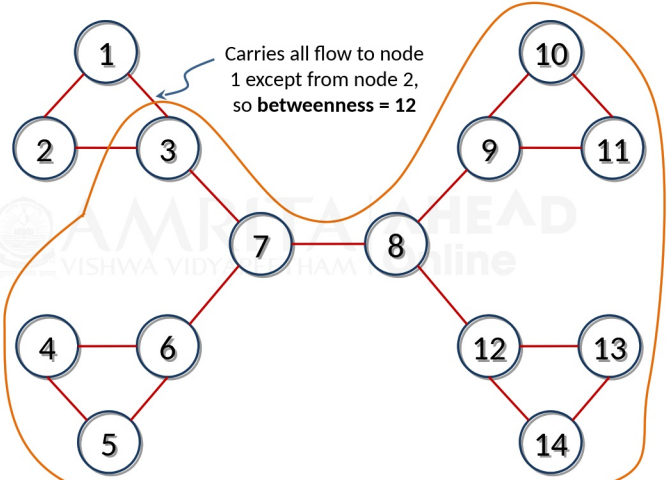
EDGE-BETWEENNESS ..



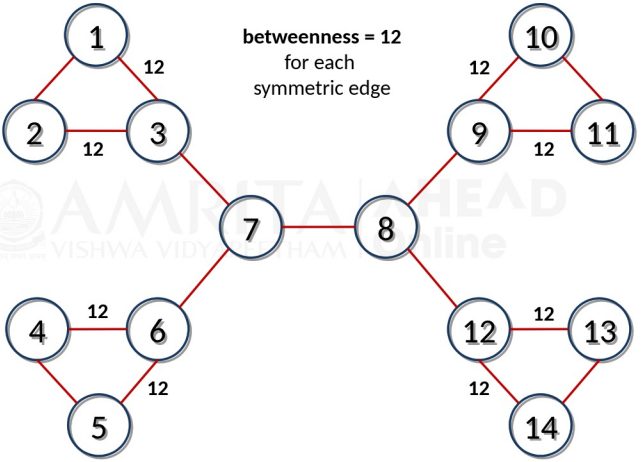
EDGE-BETWEENNESS ..



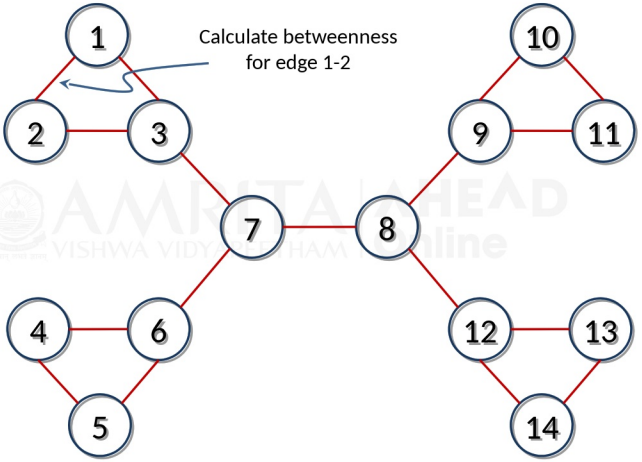
EDGE-BETWEENNESS ..



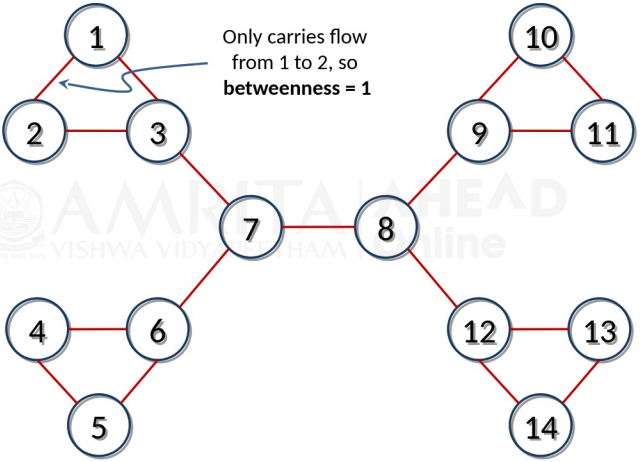
EDGE-BETWEENNESS ..



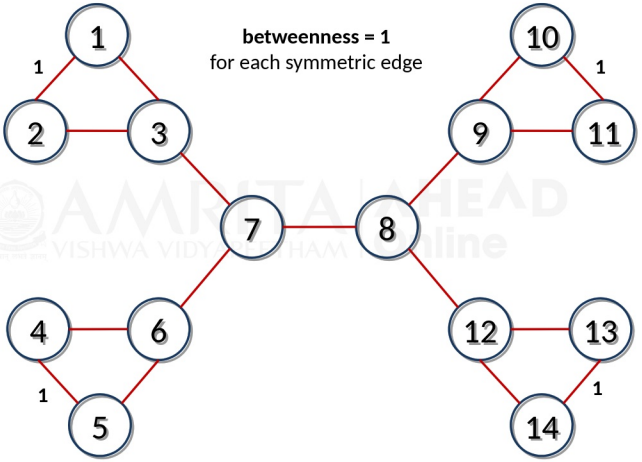
EDGE-BETWEENNESS ..



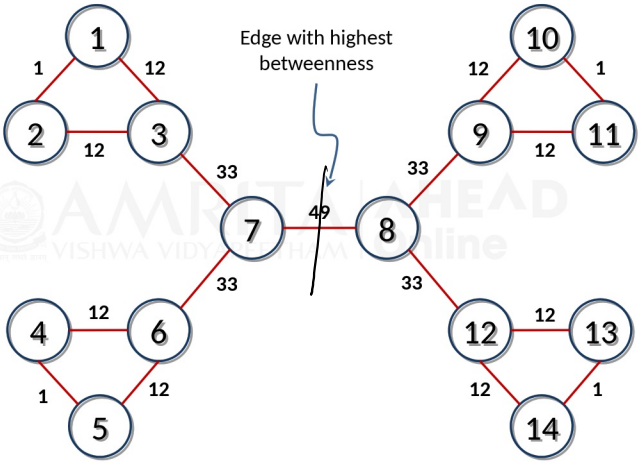
EDGE-BETWEENNESS ..



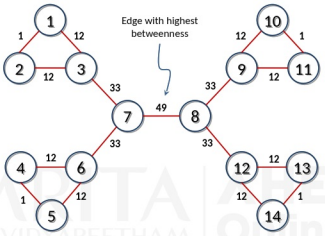
EDGE-BETWEENNESS ..



EDGE-BETWEENNESS ..



EDGE-BETWEENNESS ..



- Node Betweenness:
  - Total amount of "flow" a node carries when a unit of flow between each pair of nodes is divided up evenly over shortest paths
- Nodes and edges of high betweenness perform critical roles in the network structure

Objectives

Graph Partitioning

- Girvan-Newman Algorithm

GIRVAN-NEWMAN

ALGORITHM

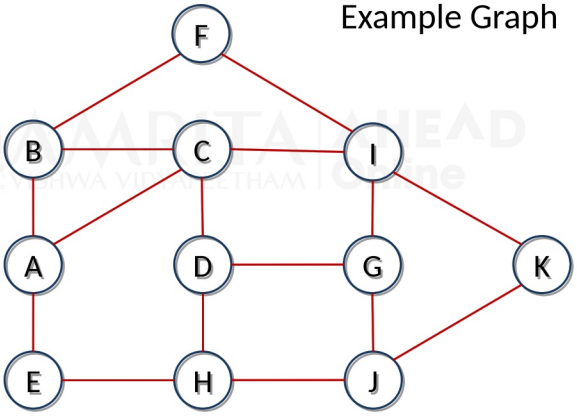
```
i.   Calculate betweenness of all edges
ii.  Remove the edge(s) with highest betweenness
iii. Repeat steps i and ii until graph is partitioned into
     as many regions as desired
```

COMPUTING EDGE BETWEENNESS EFFICIENTLY

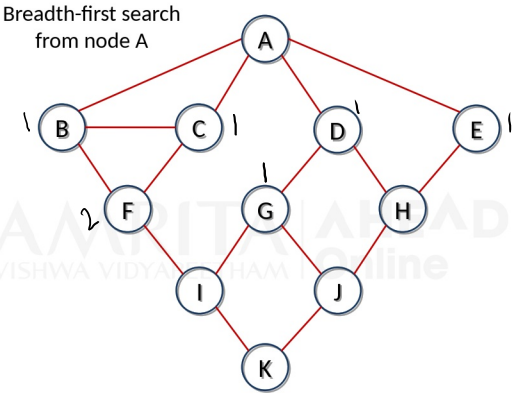
```
     For each node N in the graph
i.   Perform breadth-first search of graph starting at node N
ii.  Determine the number of shortest paths from N to every other node
iii. Based on these numbers, determine the amount of flow from N
     to all other nodes that use each edge
iv.  Divide sum of flow of all edges by 2
```

Girvan-Newman: Example

Efficient Computation

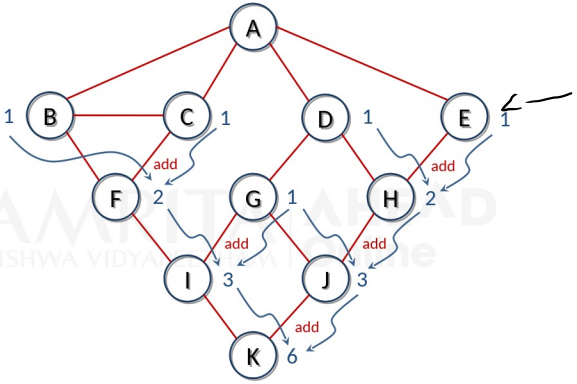


GIRVAN-NEWMAN: EXAMPLE



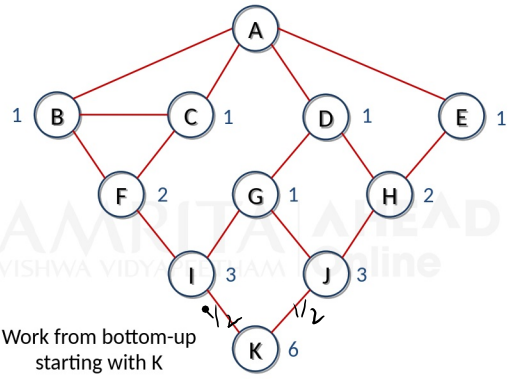
Perform breadth-first search of graph starting at node N

Girvan-Newman: Example



Determine the number of shortest paths from N to every other node

GIRVAN-NEWMAN: EXAMPLE



Edge credit

$(K,I) = (1+0)\cdot\frac{3}{6}$

$= 1/2$

$(K,J) = 1/2$

$(I,F) = (1+1/2)\cdot 2/3 = 1$

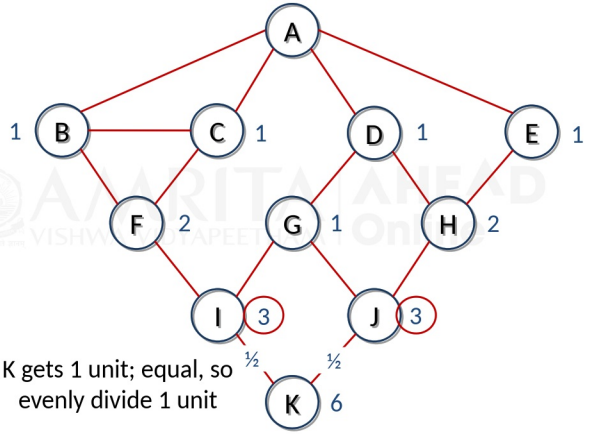
$(I,G) = 1/2$

(J,G) ⋮

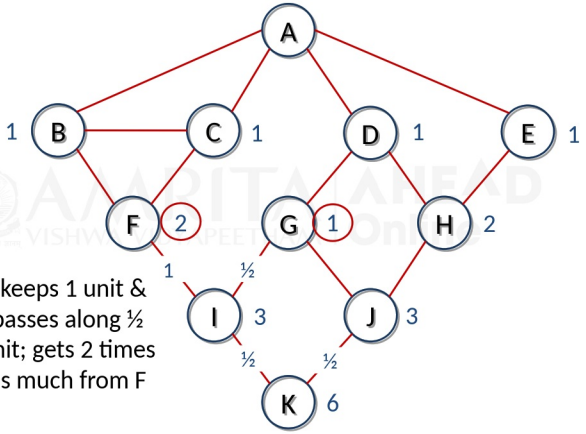
(J,H) ⋮

Based on these numbers, determine the amount of flow from N to all other nodes that use each edge

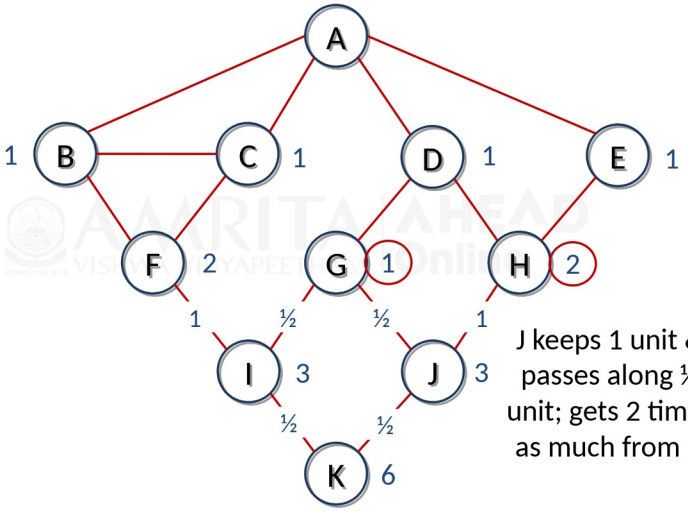
GIRVAN-NEWMAN: EXAMPLE



GIRVAN-NEWMAN: EXAMPLE

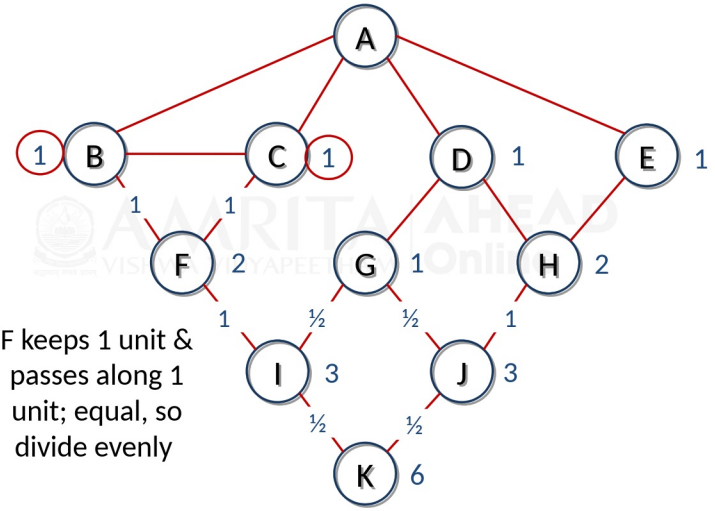


GIRVAN-NEWMAN: EXAMPLE



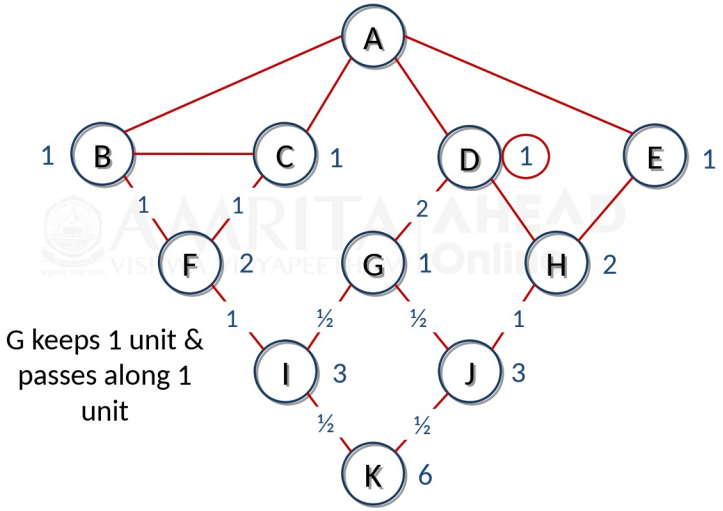
J keeps 1 unit & passes along ½ unit; gets 2 times as much from H

GIRVAN-NEWMAN: EXAMPLE

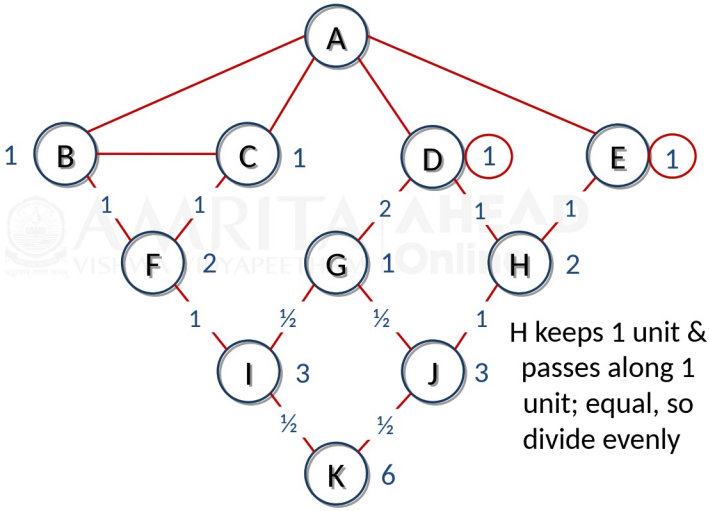


F keeps 1 unit & passes along 1 unit; equal, so divide evenly

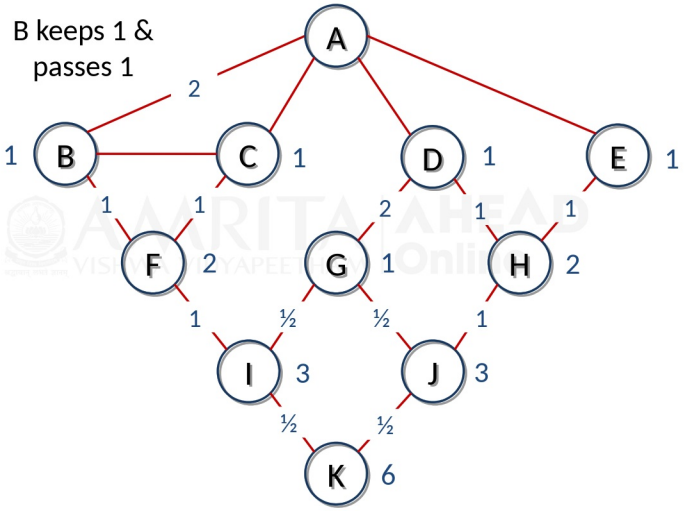
GIRVAN-NEWMAN: EXAMPLE



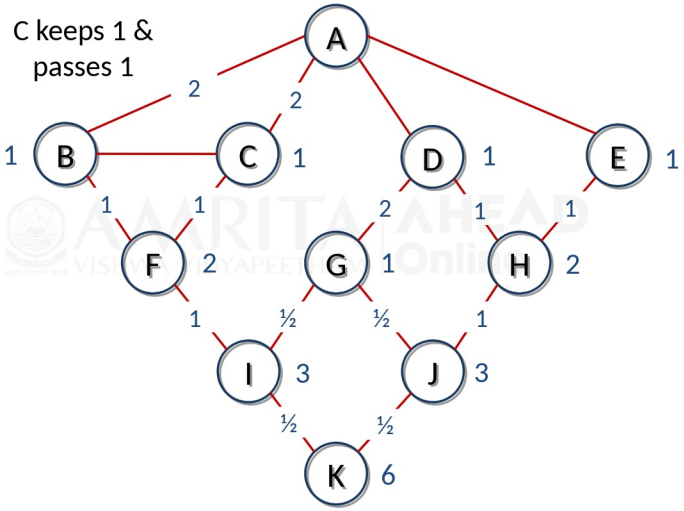
GIRVAN-NEWMAN: EXAMPLE



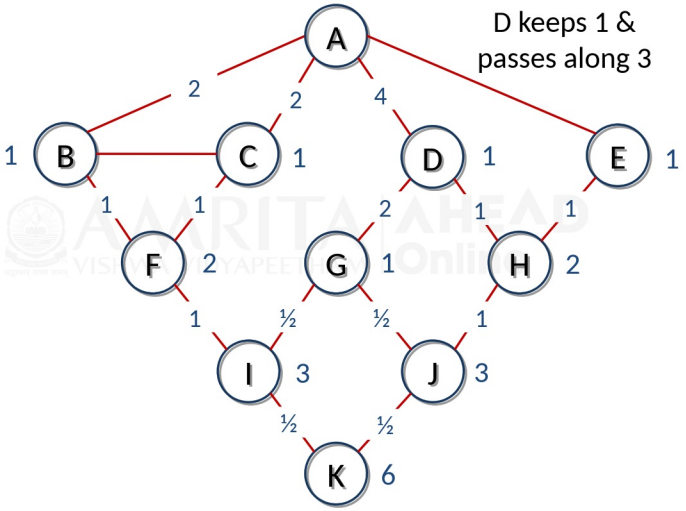
GIRVAN-NEWMAN: EXAMPLE



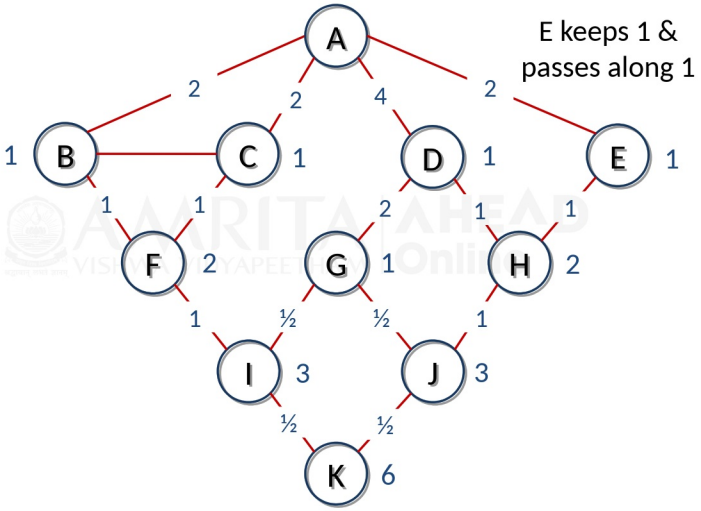
GIRVAN-NEWMAN: EXAMPLE



GIRVAN-NEWMAN: EXAMPLE



GIRVAN-NEWMAN: EXAMPLE



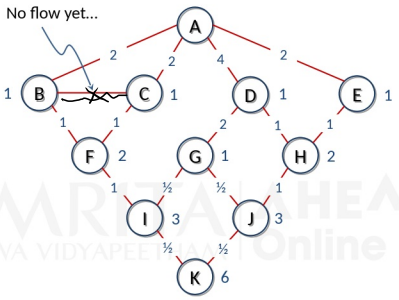
GIRVAN-NEWMAN: EXAMPLE

BFS-A BFS-B BFS

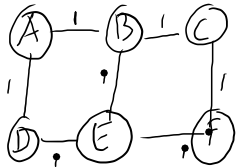
n Graphs

for a graph

w' n vertices

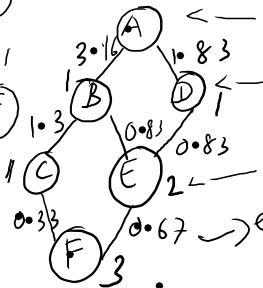


- Divide sum of flow of all edges by 2
  - Since includes flow from A to B and B to A, etc.

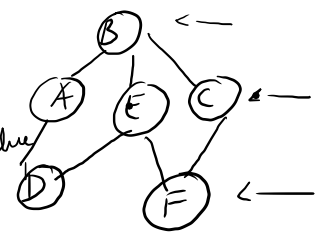


BFS

BFS_A



BFS_B



BFS_C

BFS_D

E

F

Edge credit score

$$\left(1 + \sum iec\right) \frac{\text{dep. score}}{\text{src score}}$$

iec → incoming edge credit

$(F,E) = (1+0) * 1/3 = 0.33$

$(F,E) = (1+0) * 2/3 = 0.67$

$(C,B) = (1+0.33) * 1/1 = 1.33$

$(E,B) = (1+0.67) \cdot 1/2 = 0.83$

$(E,D) = (1+0.67) \cdot 1/2 = 0.83$

$(B,A) = (1+1.33+0.83) \cdot 1/1 = 3.16$

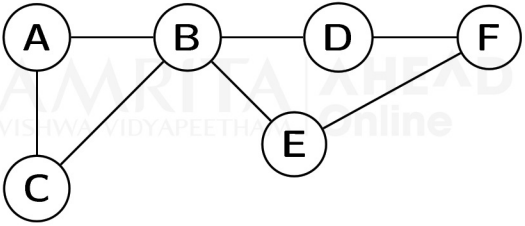
Objectives

Girvan-Newman Algorithm

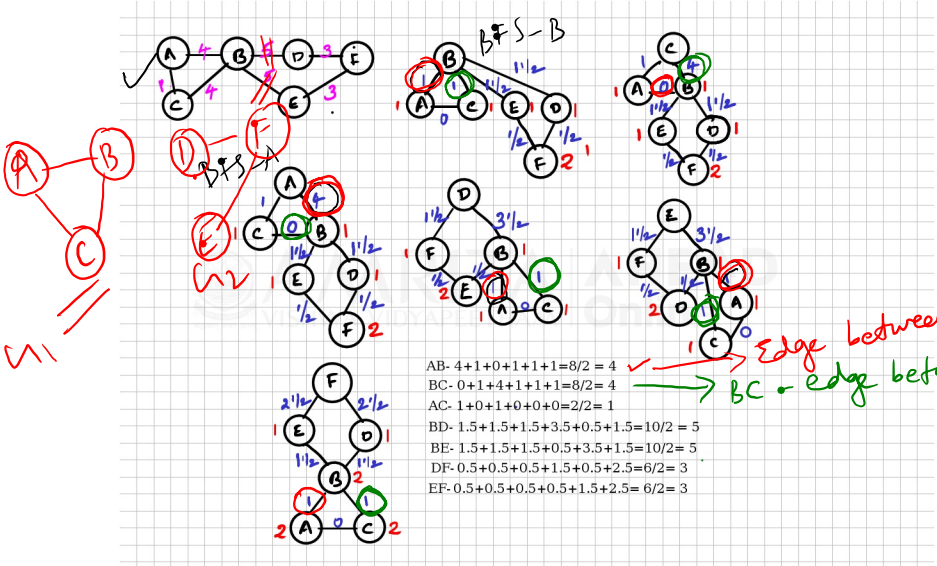
- Example
- In NetworkX

GIRVAN-NEWMAN: EXAMPLE

EDGE BETWEENNESS COMPUTATION

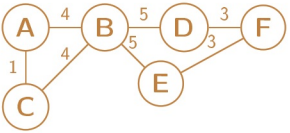


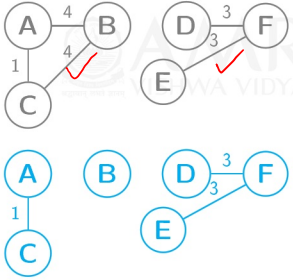
GIRVAN-NEWMAN: BFS TREES

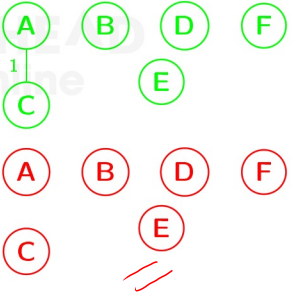


GIRVAN-NEWMAN: EXAMPLE ..

PARTITIONING THE GRAPH

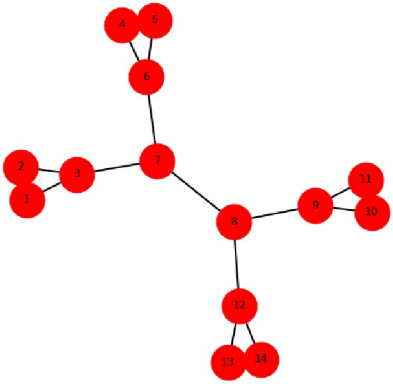






GIRVAN-NEWMAN IN NETWOKX (1/3)

```
import networkx as nx
import matplotlib.pyplot as plt
edge_widths = [2]
G2 = nx.Graph([(1, 2),(2,3),..
(7,8),(8,9),(8,12)])
color_map = []
pos=nx.springer_layout(G2)
fig, ax = plt.subplots(figsize=(8,8))
nx.draw (G2 , ax=ax,pos=pos
node_size=1500, node_color="red",
width=edge_widths)
plt.savefig("girvan_ex_1.png")
```



GIRVAN-NEWMAN IN NETWOKX (2/3)

```
import networkx as nx
from networkx.algorithms import community
communities_generator = community.girvan_newman(G2)
top_level_communities = next(communities_generator)
sorted(map(sorted, top_level_communities))
[[1, 2, 3, 4, 5, 6, 7], [8, 9, 10, 11, 12, 13, 14]]
```

breaklines

```
next1_level_communities = next(communities_generator)
sorted(map(sorted, next1_level_communities))
[[1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11, 12, 13, 14]]
```

breaklines

```
next2_level_communities = next(communities_generator)
sorted(map(sorted, next2_level_communities))
[[1, 2, 3], [4, 5, 6, 7], [8, 12, 13, 14], [9, 10, 11]]
```

GIRVAN-NEWMAN IN NETWOKX (3/3)

```
next3_level_communities = next(communities_generator)
sorted(map(sorted, next3_level_communities))
[[1, 2, 3], [4, 5, 6], [7], [8, 12, 13, 14], [9, 10, 11]]
```

```
next4_level_communities = next(communities_generator)
sorted(map(sorted, next4_level_communities))
[[1, 2, 3], [4, 5, 6], [7], [8], [9, 10, 11], [12, 13, 14]]
```

Objectives

Graph Partitioning

- Introduction
- Applications

Graph Partitioning

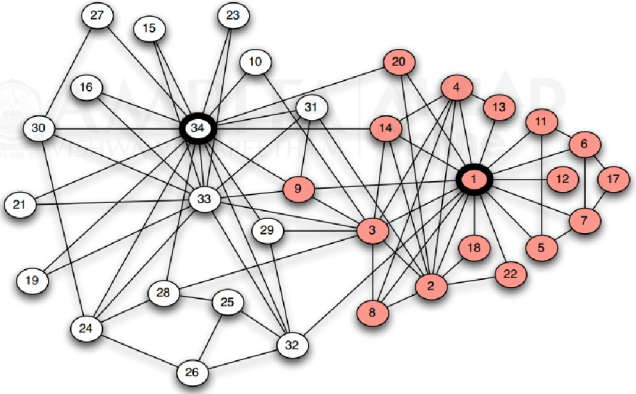
- Objectives:
  - Define densely connected regions of a network
- Graph partitioning:
  - Algorithm to identify densely connected regions
  - Breaking a network into a set of nodes densely connected with each other with edges
  - Sparser interconnections between the regions

Graph Partitioning

Applications

Karate Club splits after a dispute

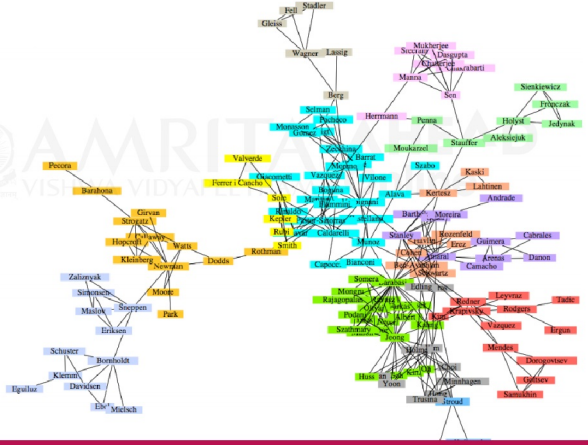
Can new clubs be identified based on network structure?



GRAPH PARTITIONING ..

APPLICATIONS

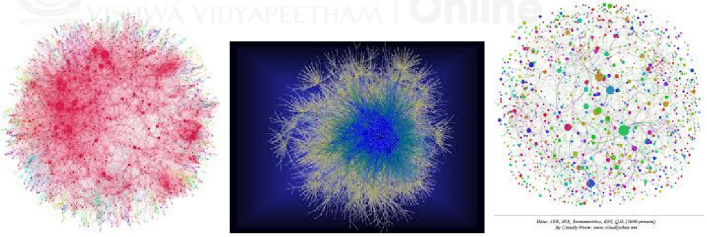
Co-authorship network: How can the tightly clustered groups be identified?



Graph Partitioning

Large Graphs Applications

- Web search
- Identify communities
- Locate hot spots
- Trace targets
- Trust Analysis
- Combat link spam



Graph Partitioning Approaches

- Many general approaches
  - Divisive methods: Repeatedly identify and remove edges connecting densely connected regions
  - Agglomerative methods: Repeatedly identify and merge nodes that likely belong in the same region

Graph Partitioning Approaches ..

- Divisive methods: breaking first at the 7-8 edge, and then the nodes into nodes 7 and 8
- Agglomerative methods: merge the 4 triangles and then pairs of triangles (via nodes 7 and 8)