Mining Social-Network Graphs

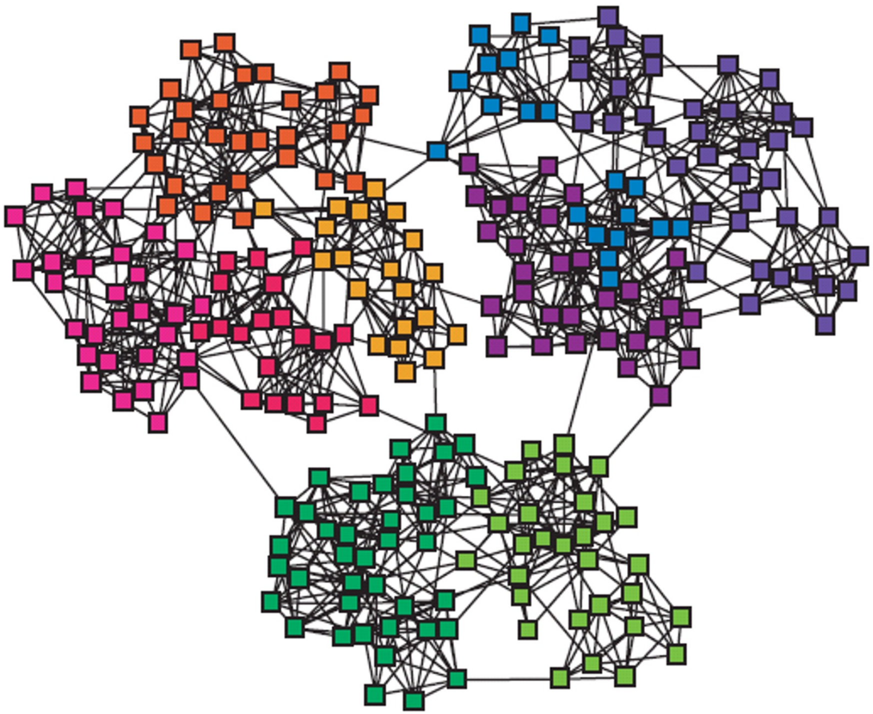
Analysis of Large Graphs: Community Detection

Professor Wei-Min Shen
University of Southern California

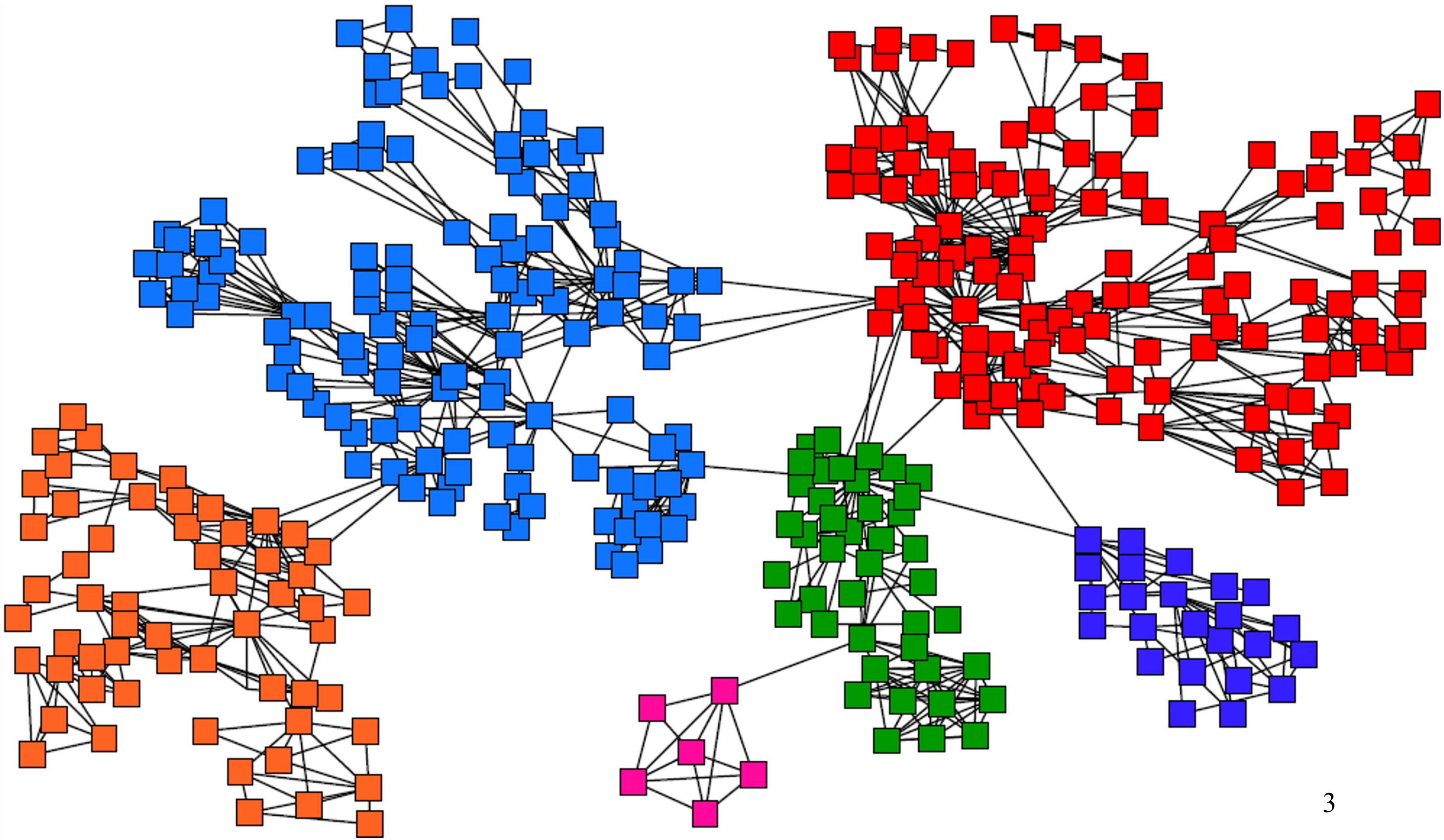
Thanks for source slides and material to:
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets
http://www.mmds.org

Networks & Communities

- We often think of networks being organized into modules, cluster, communities:



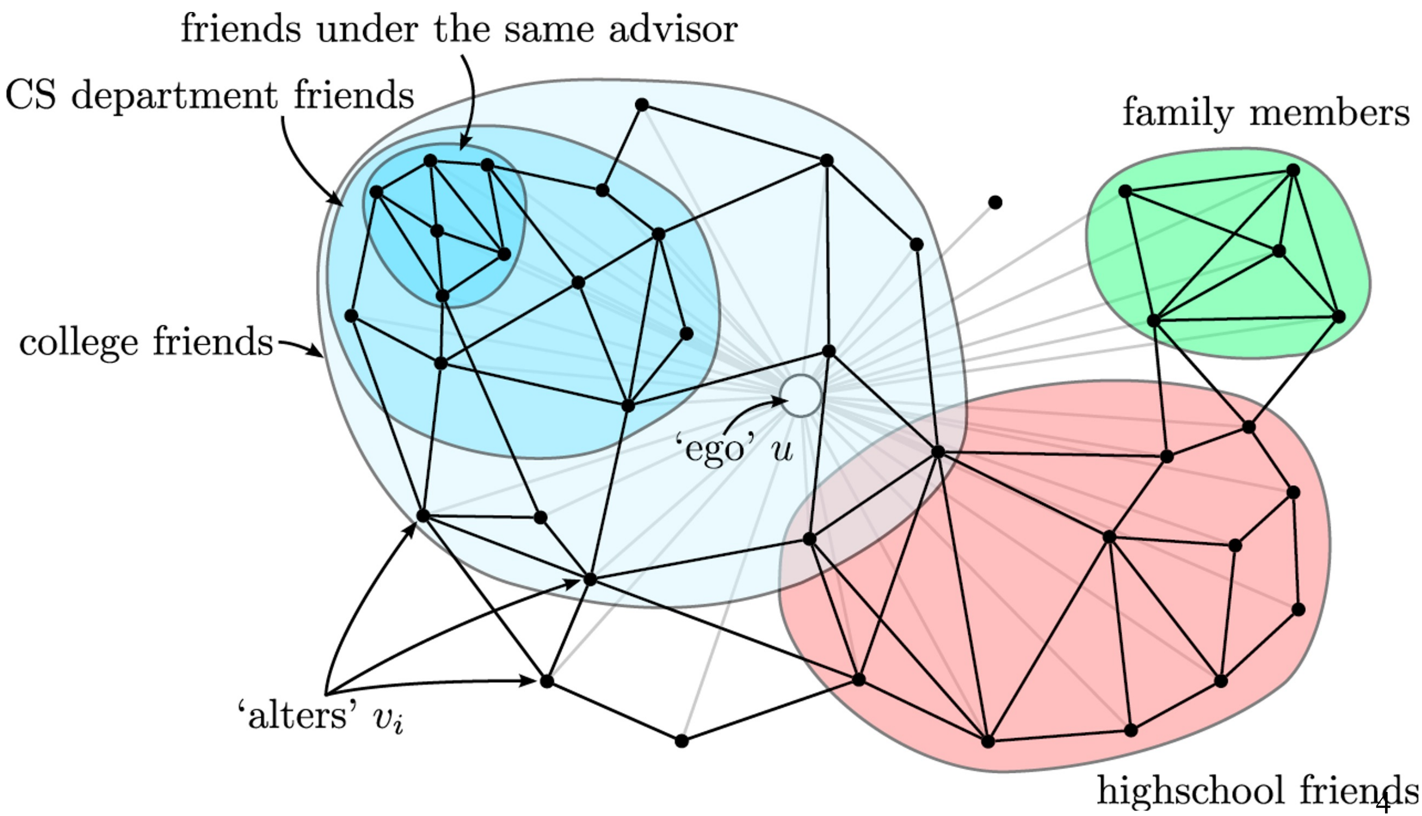
Goal: Find Densely Linked Clusters



eg:

Twitter & Facebook

- Discovering social circles, circles of trust:

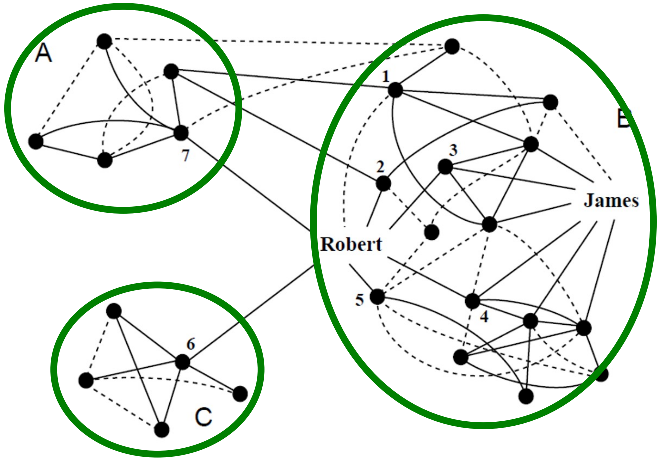


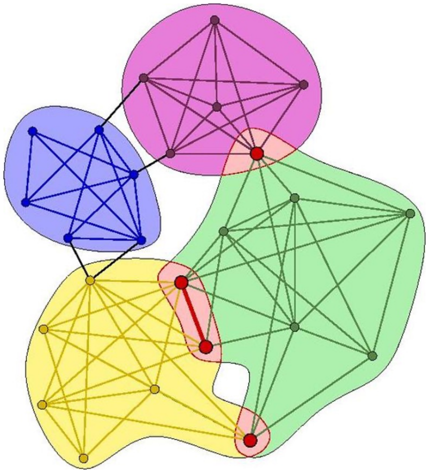
[McAuley, Leskovec: Discovering social circles in ego networks, 2012]

COMMUNITY DETECTION (GRAPH BASICS)

without overlapping

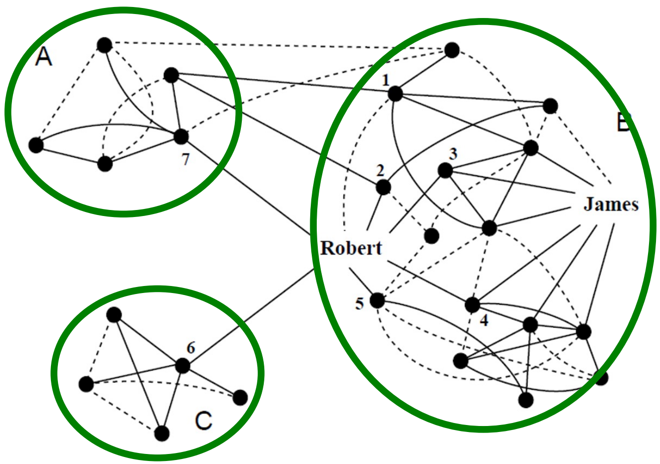
How to find communities?

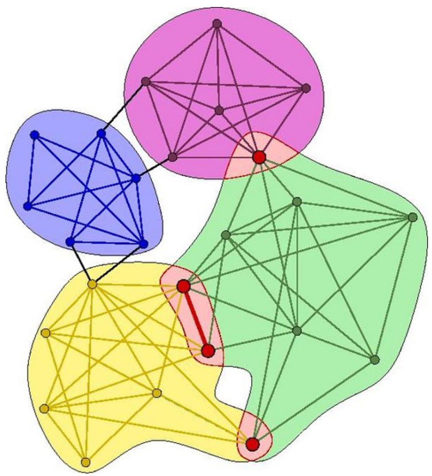




COMMUNITY DETECTION (ALGORITHMS AND METHODS)

How to find communities?





We will work with **undirected** (unweighted) networks

Approaches

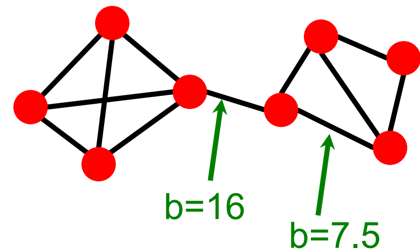
1. Clustering by edge "betweenness"
   - Cut edges that have highest "betweenness"
2. Spectral Clustering
   - Minimizing the cut size by the 2^{nd} Eigenvector
3. Direct Discovery
   - Thawing
   - Frequent itemsets

-. clustering by edge betweenness.

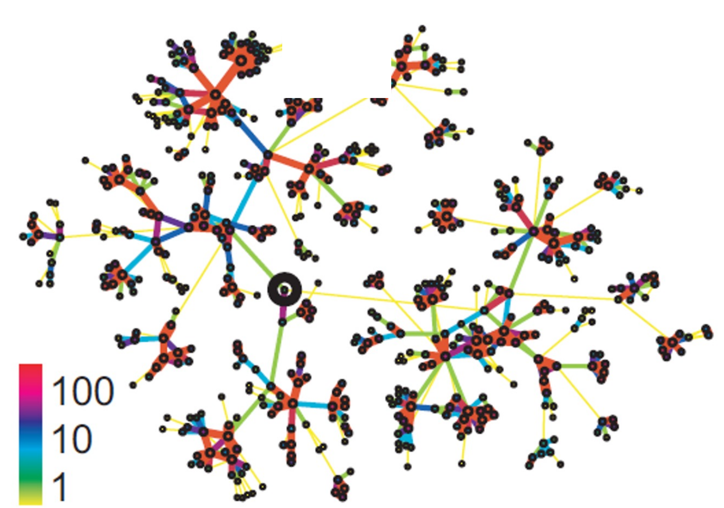
Betweenness Concept

1.

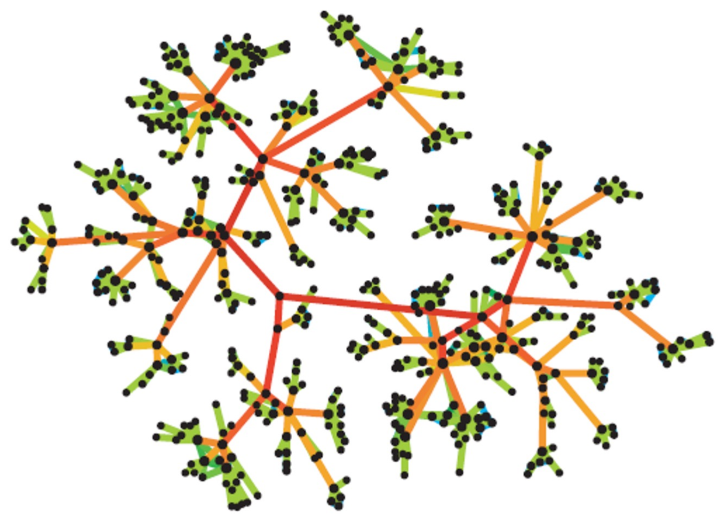
- ❑ **Edge betweenness:** Number of shortest paths passing over the edge



- ❑ **Intuition:**



Edge strengths (call volume) in a real network



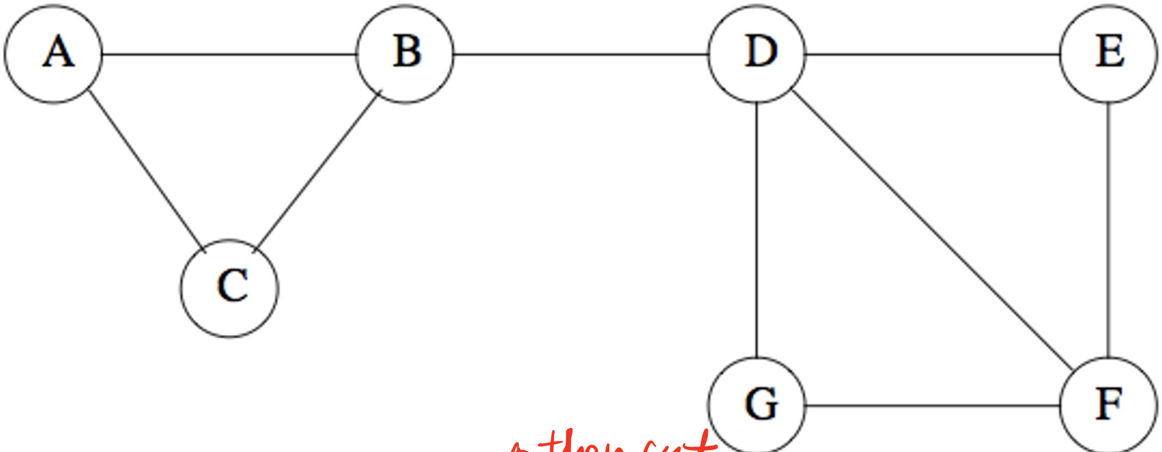
Edge betweenness in a real network

Betweenness Concept (Cont'd)

- Find edges in a social network graph that are least likely to be inside a community
- Betweenness of edge (a, b):
  - number of pairs of nodes x and y -> x, y ∈ C
  - **edge (a,b) lies on the shortest path between x and y**
- If there are several shortest paths between x and y, edge (a,b) is credited with the fraction of those shortest paths that include edge (a,b)
- **A high score is "bad"**: suggests that edge (a,b) runs between two different communities

eg:

Betweenness Example



then cut
⇒ get ≥ clusters

- Expect that edge (B,D) has highest betweenness
- (B,D) is on every shortest path from {A,B,C} to {D,E,F,G}
- Betweenness of (B,D) = 3x4 = 12
- (D,F) is on every shortest path from {A,B,C,D} to {F}
- Betweenness of (D,F) = 4x1 = 4
- **Natural communities: {A,B,C} and {D,E,F,G}**

(一) 、How to compute betweenness?

1、

compute betweeness efficiently

The Girvan-Newman Algorithm

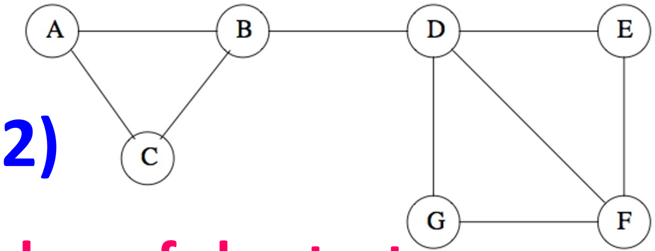
- Want to **discover communities using divisive hierarchical clustering**
  - Start with one cluster (the social network) and recursively split it
- **Will do this** based on the notion of edge **betweenness**:
  Number of shortest paths passing through the edge
- **Girvan-Newman Algorithm:**
  - Visits each node X once
  - Computes the number of shortest paths from X to each of the other nodes that go through each of the edges
- **Repeat:**
  - Calculate betweenness of edges
    1. Thresholding to remove high betweeness edges, or
    2. Remove edges with highest betweenness: **between** communities
- **Connected components are communities**
- Gives a hierarchical decomposition of the network

2. Step

Girvan-Newman Algorithm (1)

- Visit each node X once and **compute the number of shortest paths from X to each of the other nodes that go through each of the edges**
- **1) Perform a breadth-first search (BFS) of the graph, starting at node X**
  - The depth level of each node in BFS is length of the shortest path from X to that node
  - So edges that go between nodes on the same depth level can never be part of a shortest path from X
  - **Edges between depth levels are called DAG edges** (DAG = Directed Acyclic Graph)
  - **Each DAG edge is part of at least one shortest path from root X**

Girvan-Newman Algorithm (2)



- (2) Label each node by the number of shortest paths that reach it from the root node
  - Example: BFS starting from node E, labels assigned

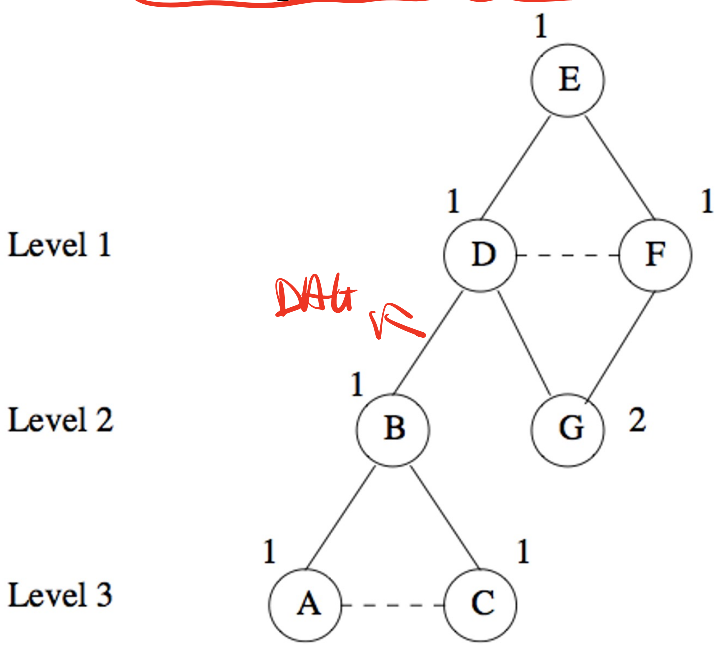


Figure 10.4: Step 1 of the Girvan-Newman Algorithm

Girvan-Newman Algorithm (3)

- 3) Calculate for each edge *e*, the sum over all nodes Y (of the fraction) of the shortest paths from the root X to Y that go through edge *e*
  - **Compute this sum for nodes and edges, starting from the bottom of the graph**
  - **Each node** other than the root node **is given a credit of 1**
  - **Each leaf node in the DAG gets a credit of 1**
  - Each **node** that is not a leaf gets credit = **1 + sum of credits of the DAG edges from that node to level below**
  - A DAG **edge** e entering node Z (from the level above) is **given a share of the credit of Z proportional to the fraction of shortest paths from the root to Z that go through e**

Girvan-Newman Algorithm (4)

- Assign node and edge values starting from bottom

leaf node: A, C, G

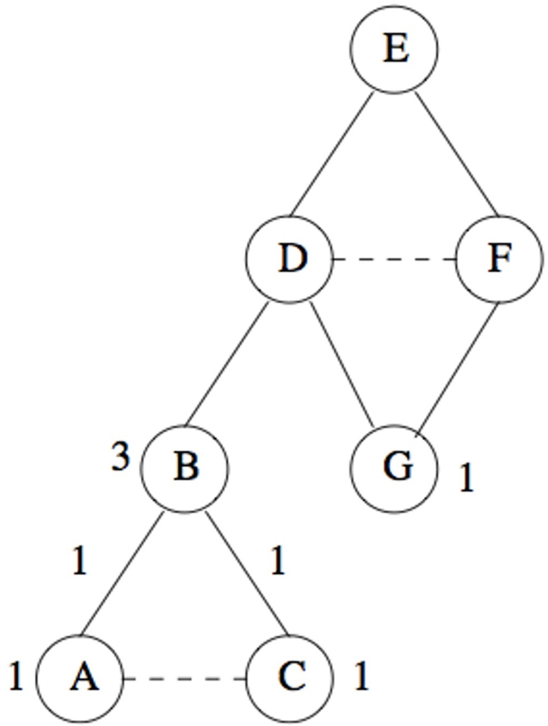
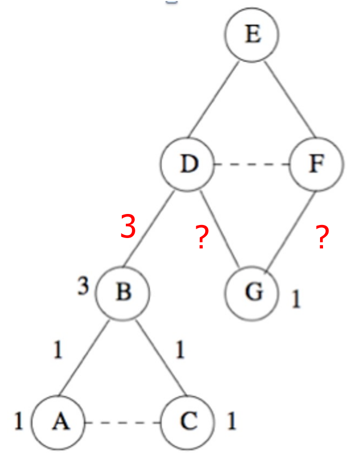


Figure 10.5: Final step of the Girvan-Newman Algorithm – levels 3 and 2

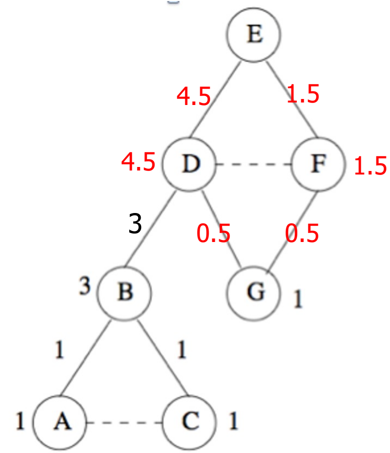
Girvan-Newman Algorithm (5)



Assigning credits:

- A and C are **leaves**: get credit = 1
- Each of these nodes **has only one parent**, so their credit=1 is **given to edges** (B,A) and (B,C)
- At level 2, G is a **leaf**: gets credit = 1
- B gets credit **1 + credit of DAG edges entering from below** = 1 + 1 +1 = 3
- B has only **one parent**, so edge (D,B) **gets entire credit of node** B = 3
- **Node G has 2 parents (D and F): how do we divide credit of G between the edges?**

Girvan-Newman Algorithm (6)



- In this case, **both D and F have just one path from E to each of those nodes**
  - So, **give half credit of node** G **to each of those edge**
  - Credit = 1/(1 + 1) = 0.5
- **In general, how we distribute credit of a node to its edges depends on number of shortest paths**
  - **Say there were 5 shortest paths to D and only 3 to F**
  - **Then credit of edge (D,G) = 5/8 and credit of edge (F,G) = 3/8**
- Node D gets credit = **1 + credits of edges below it** =
  1 + 3 + 0.5 = 4.5
- Node F gets credit = 1 + 0.5 = 1.5
- D has **only one parent**, so Edge (E,D) gets credit = 4.5 from D
- Likewise for F: Edge (E,F) gets credit = 1.5 from F

Girvan-Newman Algorithm (7): Completion of Credit Calculation starting at node E

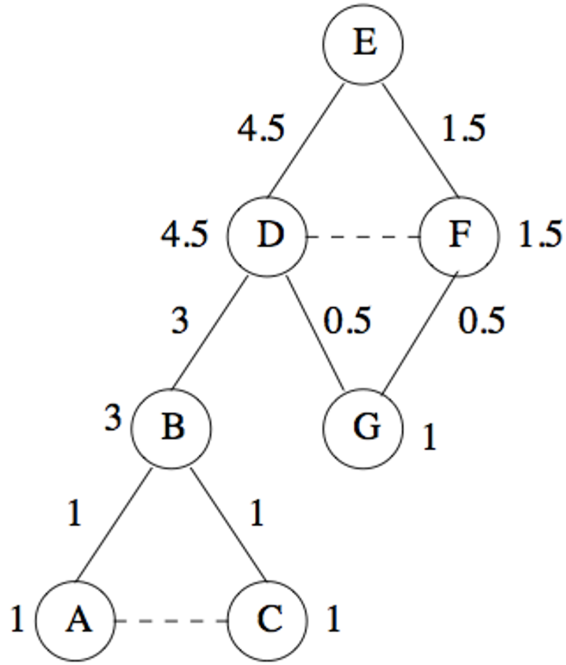


Figure 10.6: Final step of the Girvan-Newman Algorithm – completing the credit calculation

Girvan-Newman Algorithm (8): Overall Betweenness Calculation

3、

- To complete betweenness calculation, must:
  - **Repeat this for every node as root**
  - **Sum the contributions** on each edge
  - **Divide by 2** to get true betweenness
    - Since every shortest path will be counted twice (up and down), once for each of its endpoints

4. compute within community if want to cut more

Using Betweenness to Find Communities: Clustering

- **Betweenness scores for edges of a graph behave something like a distance metric**
  - Not a true distance metric
- **Could cluster by taking edges in increasing order of betweenness and adding to graph one at a time**
  - At each step, connected components of graph form clusters
- **Girvan-Newman: Start with the graph and all its edges and remove edges with highest betweenness**
  - **Continue until graph has broken into suitable number of connected components**
  - **Divisive hierarchical clustering** (top down)
    - Start with one cluster (the social network) and recursively split it

Using Betweenness to Find Communities (2)

- (B,D) has highest betweenness (12)
- Removing edge would give natural communities we identified earlier: {A,B,C} and {D,E,F,G}

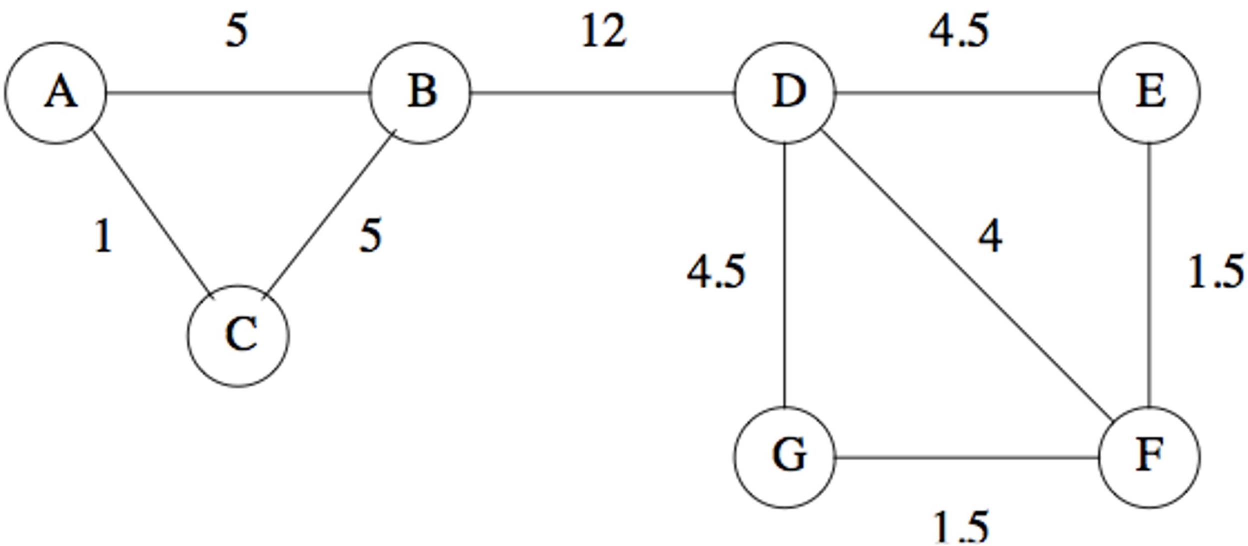


Figure 10.7: Betweenness scores for the graph of Fig. 10.1

Using Betweenness to Find Communities (3): Thresholding

- Could continue to remove edges with highest betweenness

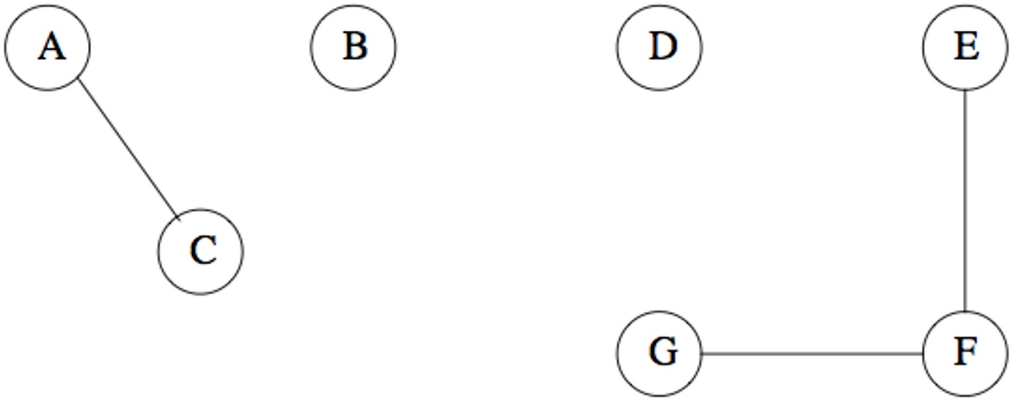


Figure 10.8: All the edges with betweenness 4 or more have been removed

Run Girvan-Newman Iteratively for Community Detection

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

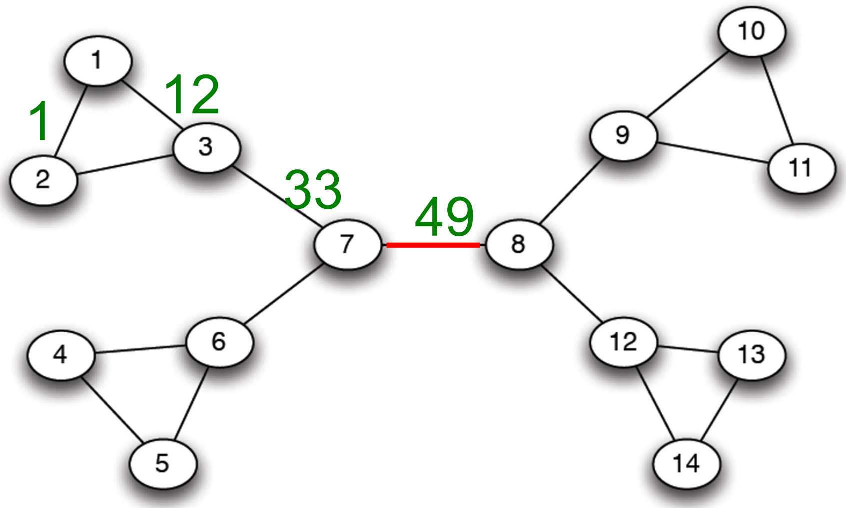
  Number of shortest paths passing through the edge

- **Girvan-Newman Algorithm:**

  » **Undirected unweighted networks**

  - **Repeat until no edges are left:**
    - Calculate betweenness of edges
    - This time: remove edges with highest betweenness
  - Connected components are communities
  - Gives a hierarchical decomposition of the network

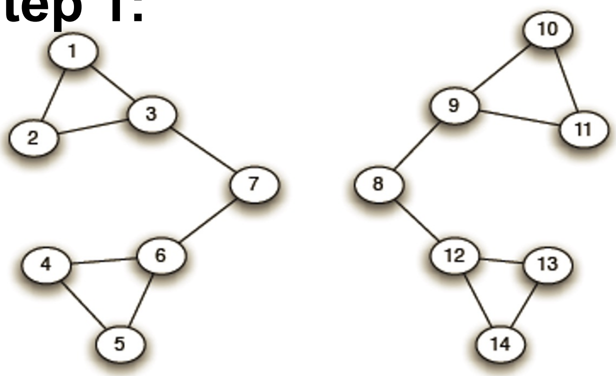
Girvan-Newman: Example



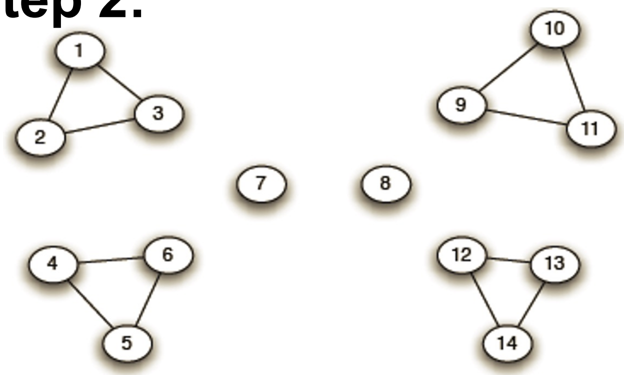
Need to re-compute betweenness at every step

Girvan-Newman: Example

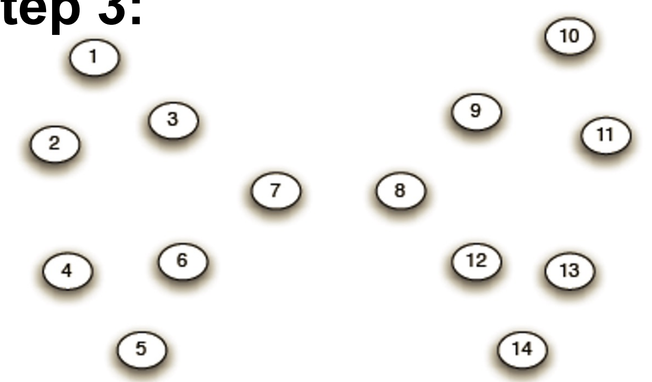
Step 1:



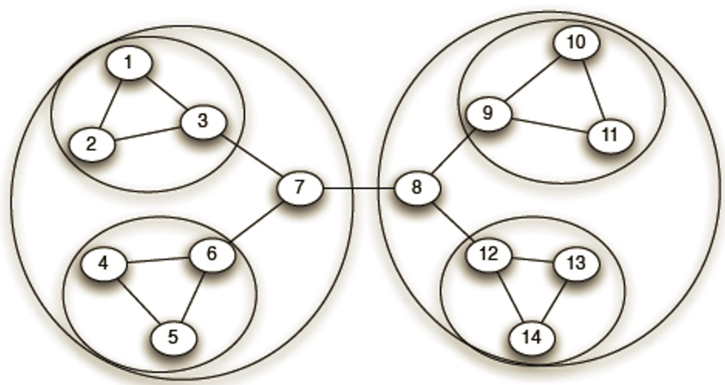
Step 2:



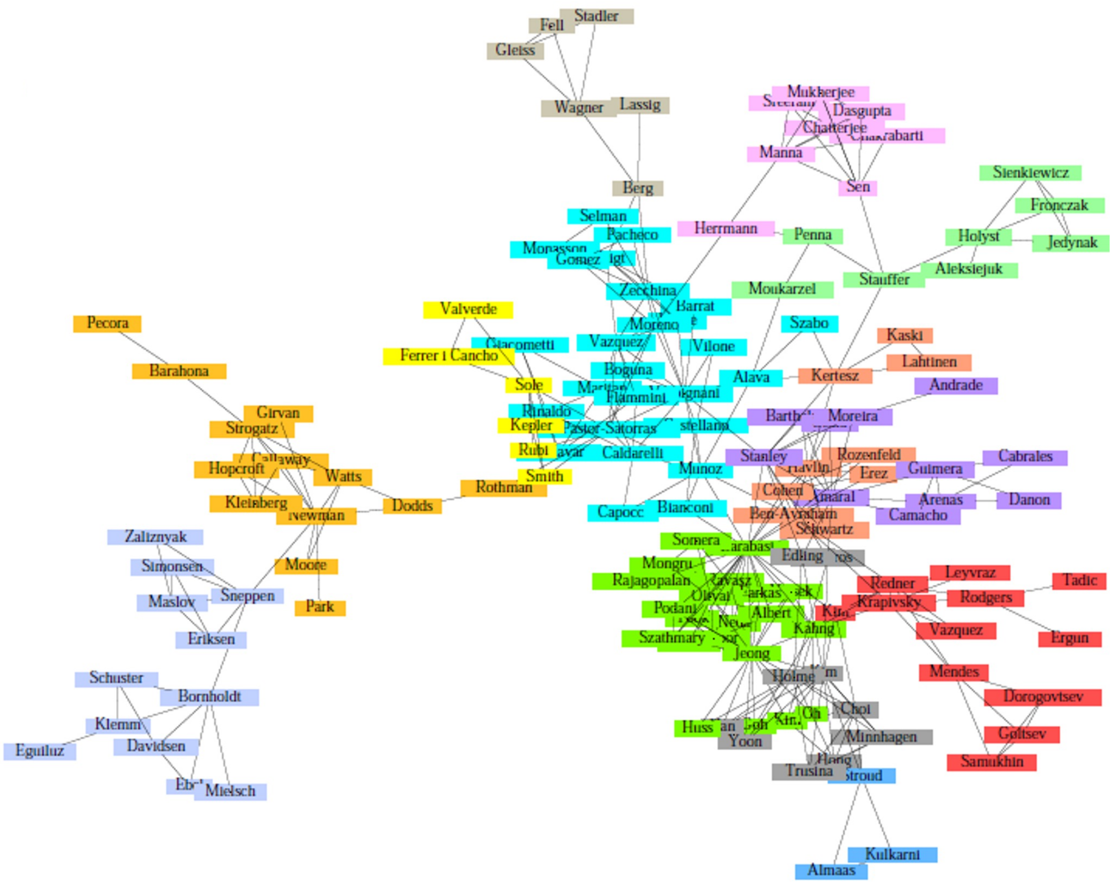
Step 3:



Hierarchical network decomposition:



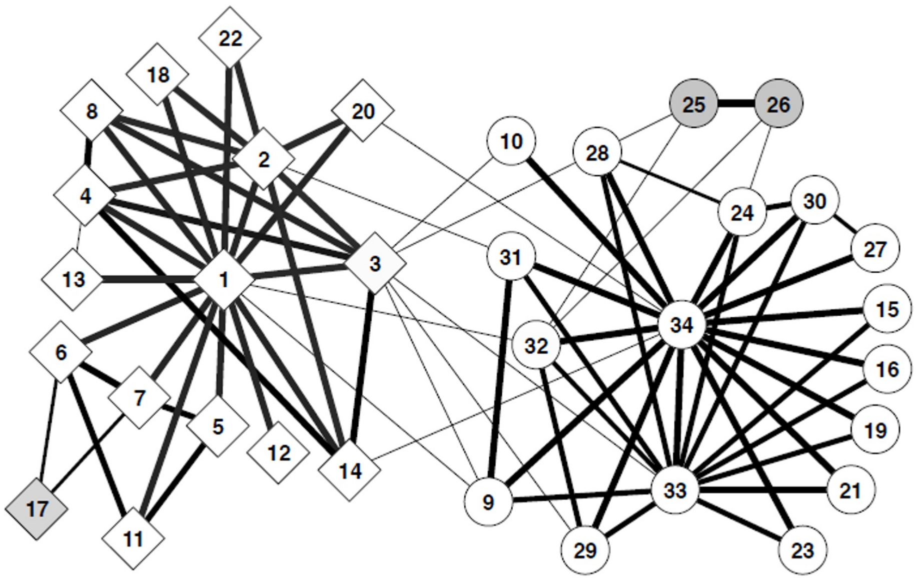
Girvan-Newman: Results

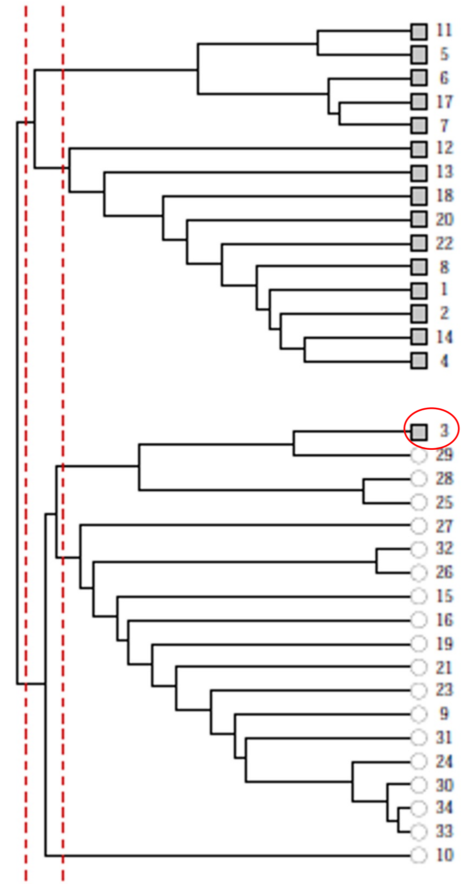


Communities in physics collaborations

Girvan-Newman: Results

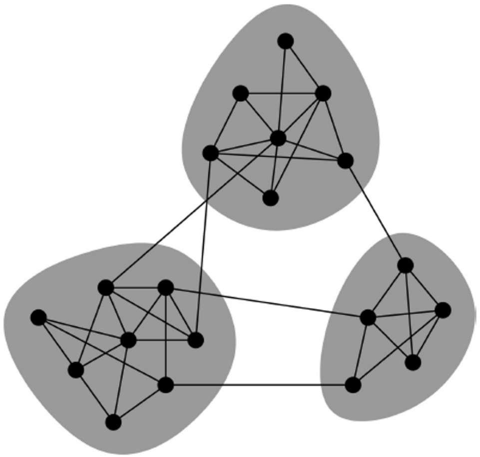
- **Zachary's Karate club:**
  Hierarchical decomposition





(2) **How to select the number of clusters?** ⇒ when do we stop?

Network Communities



- Communities: sets of tightly connected nodes
- Define modularity Q
  - A measure of how well a network is partitioned into communities
  - Given a partition of the network into groups *s* in *S*
  - $Q = Sum_{s\_in\_S}$ [(# edges in group s) – (expected # edges in group s)]

{ >0, good
{ <0, not good

When s is empty?
When s is full?

Need a null model!

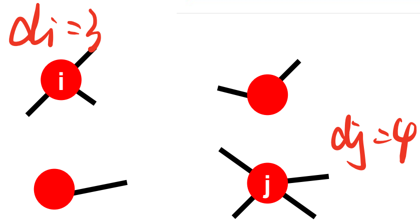
The null model is a graph which matches one specific graph in some of its structural features, but which is otherwise taken to be an instance of a random graph. The null model is used as a term of comparison, to verify whether the graph in question displays some feature, such as community structure, or not.

degree of each node: # of edges coming out of that node

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

Normalizing cost.: -1<Q<1

edge by edge

The actual existence of the edge

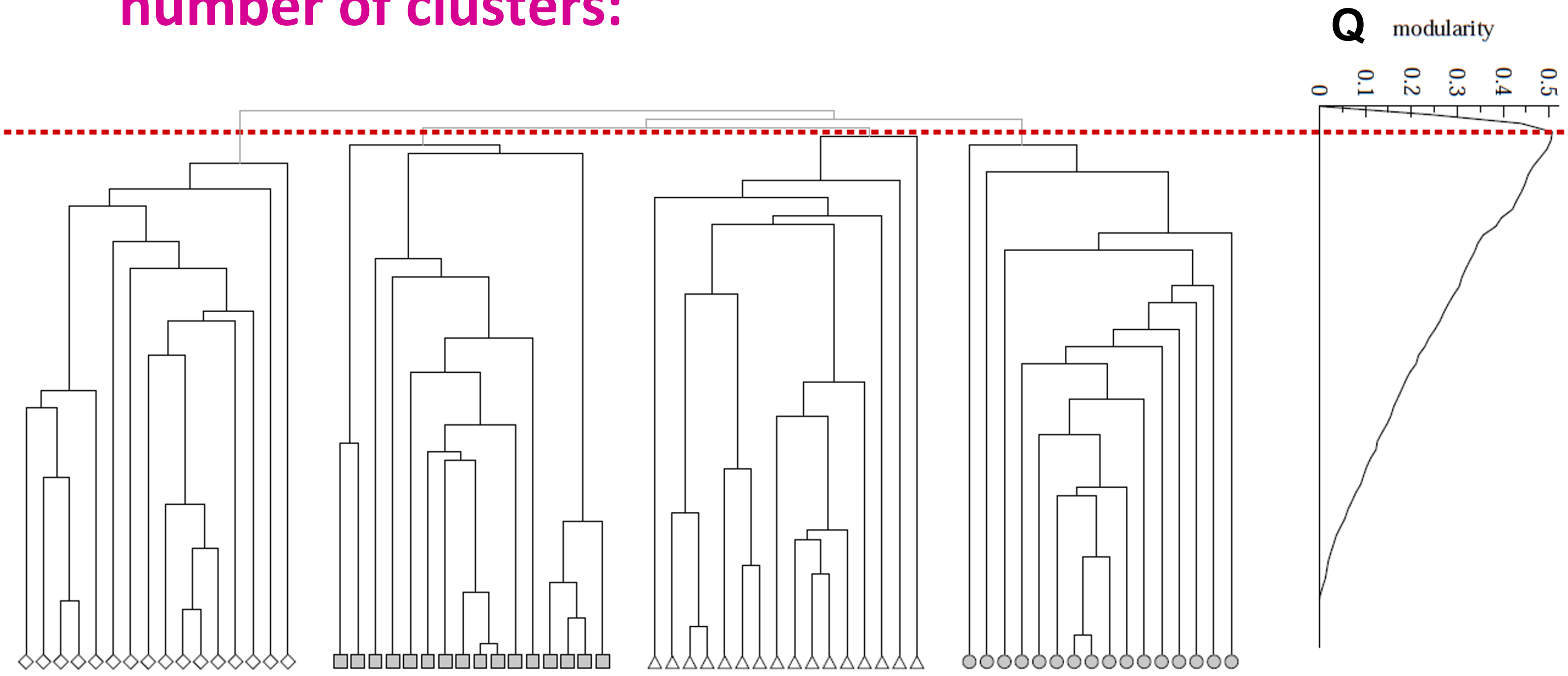
The expected existence of the edge

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

$k \in [0.3, 0.7]$

Modularity: Number of clusters

- Modularity is useful for selecting the number of clusters:



Exercises

What are the modularity of partition S of G:

1: G is partition into itself S=G $Q = 0.$

2: G is partitioned into S of all single nodes $Q = -1$

3: G is partitioned into two communities

S1 = {one single node x}

S2 = {G-x}

4: if G is below, S1={A,B,C}, S2={D,E,F}

```
A -- B  ------------ D -- E
 \   /                \ /
   C                   F
```

二. **SPECTRAL CLUSTERING**

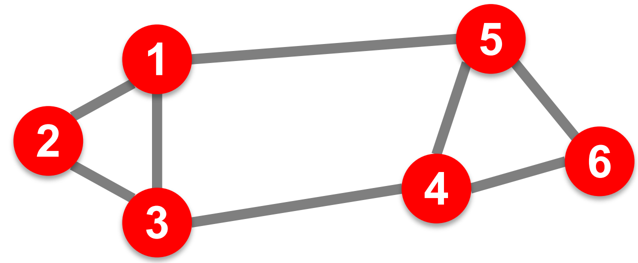
Partitioning Graphs

- Minimizing the cut size by the 2nd Eigenvector

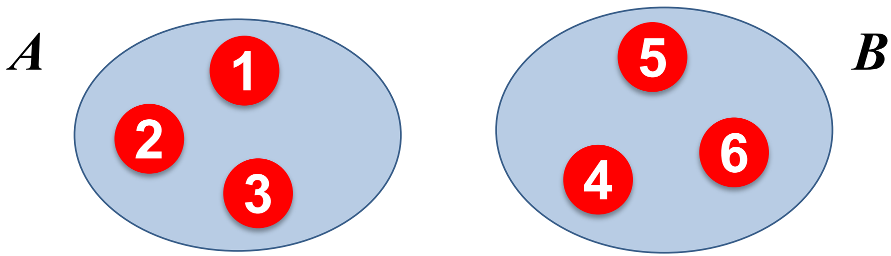
- Another approach to organizing social networking graphs
- Problem: partitioning a graph to minimize the number of edges that connect different components (communities)
- Goal of minimizing the cut size
- If you just joined Facebook with only one friend
  - Don't want to partition the graph with you disconnected from rest of the world
  - Want components to be not too unequal in size

Graph Partitioning

- **Undirected graph $G(V, E)$:**



- **Bi-partitioning task:**
  - Divide vertices into two disjoint groups A, B



- **Questions:**
  - How can we define a "good" partition of G?
  - How can we efficiently identify such a partition?

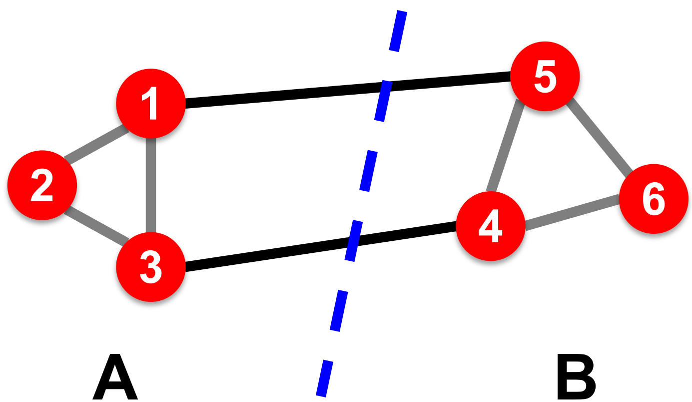
Graph Partitioning

1-

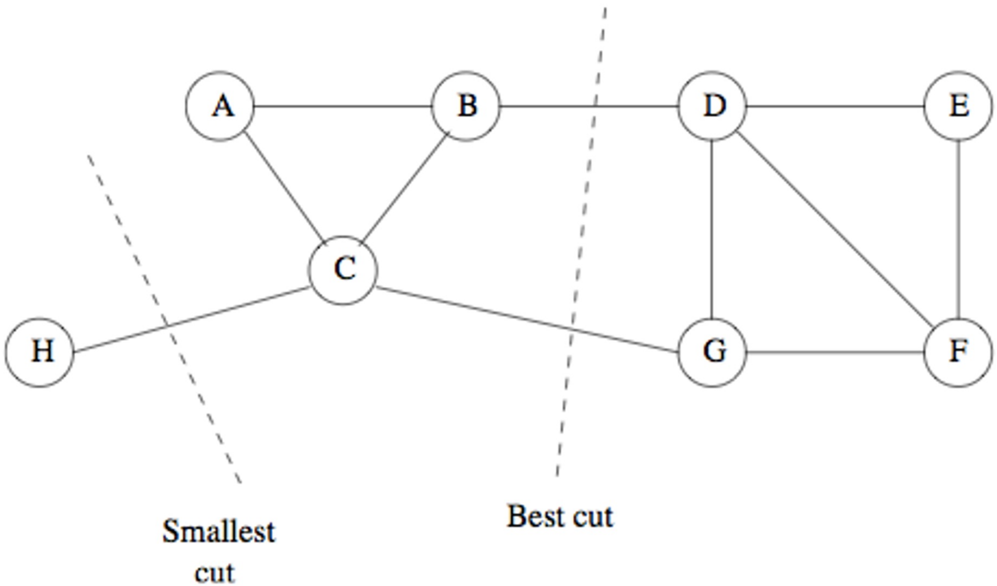
- **What makes a good partition?**

minimize cut size
community balanced
in-group connection↑, between-group connection↓

  - Divide nodes into two sets so that the **cut (set of edges that connect nodes in different sets) is minimized**
  - Want the two sets to be approximately equal in size
  - Maximize the number of within-group connections
  - Minimize the number of between-group connections



Example 10.14



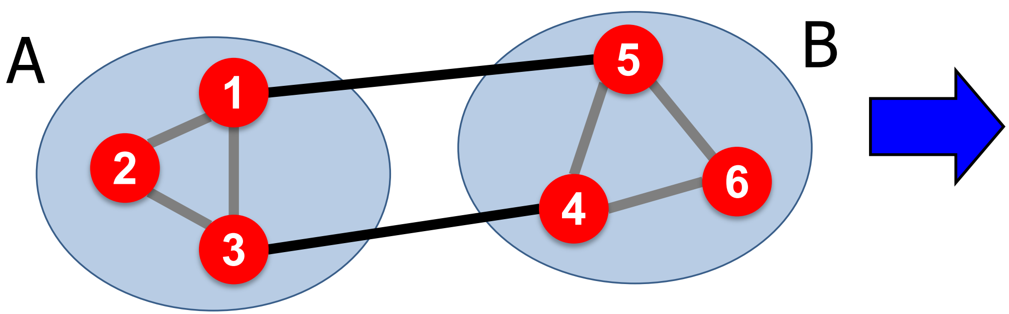
- If we minimize cut: best choice is to put H in one set, other nodes in other set
- But: we reject partitions where one set is too small
- Better is to use cut with (B,D) and (C,G)
- **Smallest cut is not necessarily the best cut**

2、

Graph Cuts

- **Express partitioning objectives as a function of the "edge cut" of the partition**
- **Cut:** Set of edges with only one vertex in a group:

$$cut(A,B) = \sum_{i \in A, j \in B} w_{ij}$$



$cut(A,B) = 2$

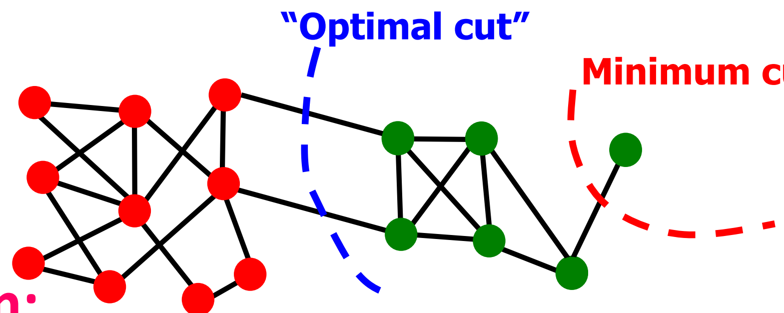
Graph Cut Criterion

3. not reflect balance after cut.

- **Criterion: Minimum-cut**
  - Minimize weight of connections between groups

$$\arg\min_{A,B} cut(A,B)$$

- **Degenerate case:**



- **Problem:**
  - Only considers external cluster connections
  - Does not consider internal cluster connectivity

Graph Cut Criteria

4.

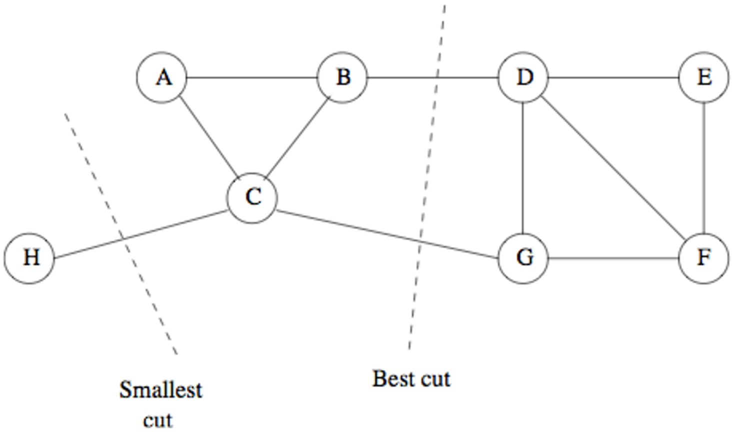
- **Criterion: Normalized-cut** [Shi-Malik, '97]
  - Connectivity between groups relative to the density of each group

$$ncut(A,B) = \frac{cut(A,B)}{vol(A)} + \frac{cut(A,B)}{vol(B)}$$

$\boldsymbol{vol(A)}$: total weight of the edges with at least one endpoint in $\boldsymbol{A}$: $\boldsymbol{vol(A) = \sum_{i \in A} k_i}$

- **Why use this criterion?**
  - Produces more balanced partitions
- **How do we efficiently find a good partition?**
  - **Problem:** Computing optimal cut is NP-hard

Example 10.15



- Partition nodes of graph into two disjoint sets S and T
- Normalized Cut for S and T is:

$$\frac{\text{Cut}(S,T)}{\text{Vol}(S)} + \frac{\text{Cut}(S,T)}{\text{Vol}(T)}$$

- If we choose S={H} and T={A,B,C,D,E,F,G} then Cut(S,T) = 1
  - Vol(S) = 1 (number of edges with at least one end in S)
  - Vol(T) = 11: all edges have at least one node in T
  - Normalized cut is 1/1 + 1/11 = 1.09
- For cut (B,D) and (C,G): S = {A,B,C,H}, T = {D,E,F,G}, Cut(S,T) = 2
- Vol(S) = 6, Vol(T) = 7, normalized cut: 2/6 + 2/7 = 0.62

smaller

Using Matrix Algebra to Find Good Graph Partitions

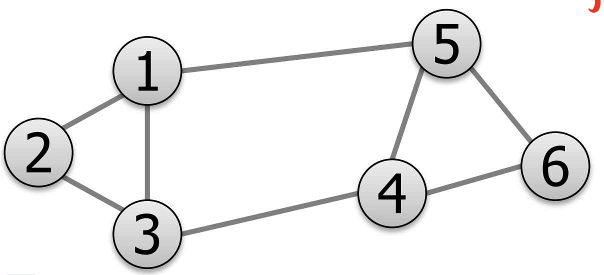
- Three matrices that describe aspects of a graph:
  - Adjacency Matrix if connect to edge, put 1, else 0.
  - Degree Matrix : degree of nodes
  - Laplacian Matrix: difference between degree and adjacency matrix
- Then get a good idea of how to partition graph from eigenvalues and eigenvectors of its Laplacian matrix

Matrix Representations

(1)

- **Adjacency matrix (A):**
  - $n \times n$ matrix
  - $A=[a_{ij}]$, $a_{ij}=1$ if edge between node i and j

diagonal is 0



	1	2	3	4	5	6
1	0	1	1	0	1	0
2	1	0	1	0	0	0
3	1	1	0	1	0	0
4	0	0	1	0	1	1
5	1	0	0	1	0	1
6	0	0	0	1	1	0

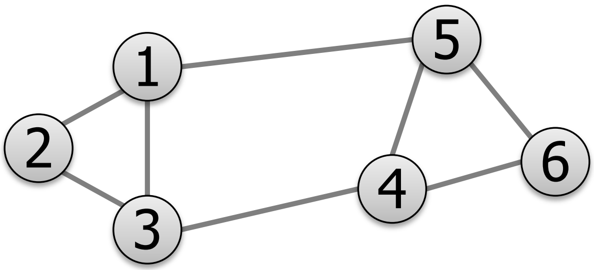
- **Important properties:**
  - Symmetric matrix
  - Eigenvectors are real and orthogonal
    - dot_product(Eigenvectors_i, Eigenvectors_j) = 0

Matrix Representations

(2)

- **Degree matrix (D):**
  - $n \times n$ diagonal matrix
  - $D=[d_{ii}]$, d_{ii} = degree of node i

is 0 except diagonal



	1	2	3	4	5	6
1	3	0	0	0	0	0
2	0	2	0	0	0	0
3	0	0	3	0	0	0
4	0	0	0	3	0	0
5	0	0	0	0	3	0
6	0	0	0	0	0	2

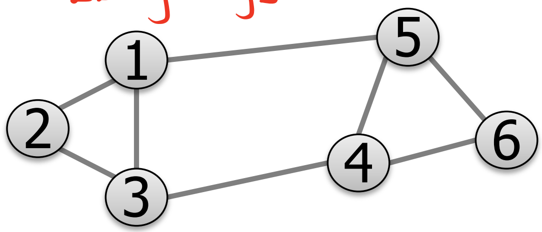
Matrix Representations (*L=D-A*)

(3)

Laplacian matrix (L):

- ***n x n*** symmetric matrix

L[dij - aij]



	1	2	3	4	5	6
1	3	-1	-1	0	-1	0
2	-1	2	-1	0	0	0
3	-1	-1	3	-1	0	0
4	0	0	-1	3	-1	-1
5	-1	0	0	-1	3	-1
6	0	0	0	-1	-1	2

What is trivial eigenpair?

- $\boldsymbol{x} = (\mathbf{1}, \ldots, \mathbf{1})$ then $\boldsymbol{L} \cdot \boldsymbol{x} = \mathbf{0} \cdot \boldsymbol{x}$ and so $\boldsymbol{\lambda} = \boldsymbol{\lambda_1} = \mathbf{0}$ **(smallest eigenvalue)**

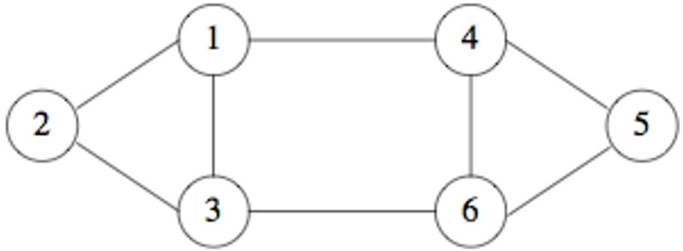
Important properties of symmetric matrices:

- **Eigenvalues** are non-negative real numbers

Why is *L* so interesting? Because *Lx* = 0 when *x*=I

$$\mathrm{x}^{\mathrm{T}}\mathbf{1} = \sum_{i=1}^{n} x_i = 0$$

Example 10.19



$$\begin{bmatrix} 3 & -1 & -1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ -1 & -1 & 3 & 0 & 0 & -1 \\ -1 & 0 & 0 & 3 & -1 & -1 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & -1 & -1 & 3 \end{bmatrix}$$

- Graph and its Laplacian matrix
  - **Why is *L* so interesting? Because *Lx* = 0 when *x*=I**

(4)

Eigenvalues and Eigenvectors

- The transformation matrix $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ preserves the direction of vectors parallel to **v** = $(1,-1)^T$ (in purple) and **w** = $(1,1)^T$ (in blue). The vectors in red are not parallel to either eigenvector, so, their directions are changed by the transformation.

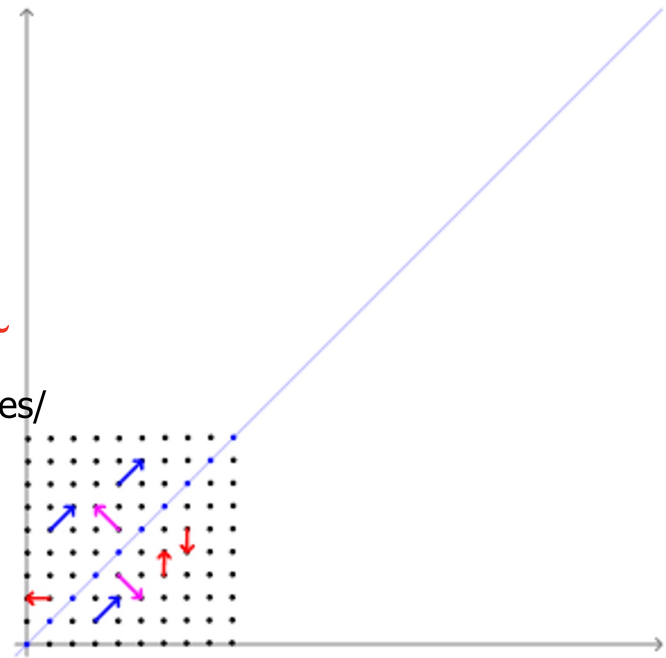
eigenvector of A

$$\underbrace{\lambda}_{\text{constant, eigenvalue}} \text{ in } A\mathbf{v} = \lambda \mathbf{v}$$

constant, eigenvalue

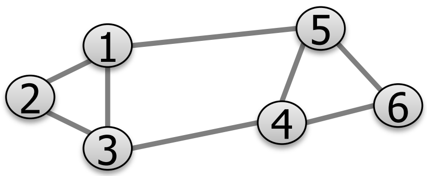
http://setosa.io/ev/eigenvectors-and-eigenvalues/

https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors



Examples of Eigenpairs for L

- **Laplacian matrix (L=D-A)**
  - $n \times n$ symmetric matrix



$L = D - A$

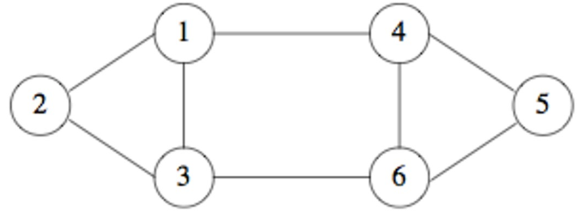
	1	2	3	4	5	6
1	3	-1	-1	0	-1	0
2	-1	2	-1	0	0	0
3	-1	-1	3	-1	0	0
4	0	0	-1	3	-1	-1
5	-1	0	0	-1	3	-1
6	0	0	0	-1	-1	2

- **Eigenpair x and λ for L: $\boxed{Lx=\lambda x}$**
- **For L, what is a trivial eigenpair x and λ?** the second eigenvector tells how to cut graph.
  - $\boldsymbol{x} = (\mathbf{1}, \ldots, \mathbf{1})$, $\boldsymbol{L} \cdot \boldsymbol{x} = \mathbf{0} \cdot \boldsymbol{x}$, so $\boldsymbol{\lambda} = \boldsymbol{\lambda_1} = \mathbf{0}$ **(smallest eigenvalue)**

  — first eigenvector
- **In general, important properties of a symmetric matrix**
  - **Eigenvalues** λ are non-negative real numbers
  - **Eigenvectors** x are real and orthogonal

$$\mathbf{x}^{\mathrm{T}}\mathbf{1} = \sum_{i=1}^{n} x_i = 0$$

Example (cont.)



Eigenvalue	0	1	3	3	4	5
Eigenvector	1	1	-5	-1	-1	-1
	1	2	4	-2	1	0
	1	1	1	3	-1	1
	1	-1	-5	-1	1	1
	1	-2	4	-2	-1	0
	1	-1	1	3	1	-1

positive as a community,
negative as another one!

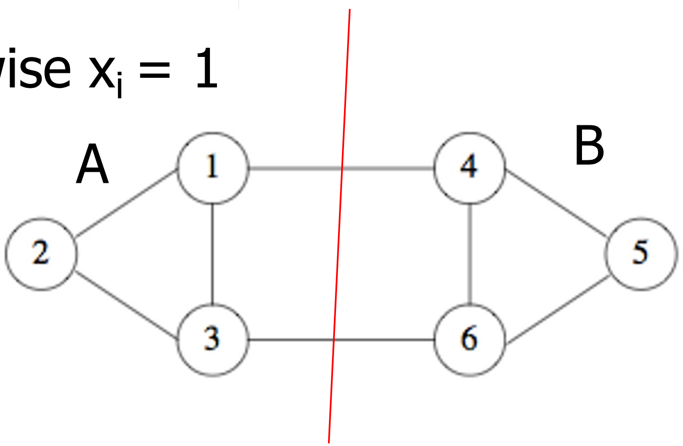
- Use standard math package to find all eigenvalues and eigenvectors
  - (Have not scaled eigenvectors to length 1, but could so easily)
- Second eigenvector has three positive and three negative components
- Suggest obvious partitioning of {1,2,3} and {4,5,6}

Ncut as an optimization problem

Let: x be an $N = |V|$ dimensional indicator vector,

x_i = -1 if i node is in A; otherwise x_i = 1

x = (-1, -1, -1, 1, 1, 1)



Let degree $\boldsymbol{d}(i) = \sum_j w(i,j)$ when in our case, w(i, j) = 1

Let $k = \frac{\sum_{x_i>0} \boldsymbol{d}_i}{\sum_i \boldsymbol{d}_i}$ and $b = \frac{k}{1-k}$.

Let $\boldsymbol{y} = (\mathbf{1} + \boldsymbol{x}) - b(\mathbf{1} - \boldsymbol{x})$,

Read [Shi-Malik, '97] $min_{\boldsymbol{x}} Ncut(\boldsymbol{x}) = min_{\boldsymbol{y}} \frac{\boldsymbol{y}^T(\mathbf{D} - \mathbf{W})\boldsymbol{y}}{\boldsymbol{y}^T \mathbf{D} \boldsymbol{y}}$,

Ncut as an optimization problem

Rayleigh Quotient:

$$\frac{\boldsymbol{x}^T \mathbf{A} \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}$$ is minimized by the next smallest eigenvector of A

Graph Theory

(5)

λ_2 as optimization problem

- **Fact:** For symmetric matrix M:

$$\lambda_2 = \min_x \frac{x^T \overset{\rightarrow L}{\underline{M}} x}{x^T x}$$

- **What is the meaning of min $x^T L x$ on G?**
  - $x^T L x = \sum_{i,j=1}^{n} L_{ij} x_i x_j = \sum_{i,j=1}^{n} (D_{ij} - A_{ij}) x_i x_j$
  - $= \sum_i D_{ii} x_i^2 - \sum_{(i,j)\in E} 2 x_i x_j$
  - $= \sum_{(i,j)\in E} (\underbrace{x_i^2 + x_j^2} - 2 x_i x_j) = \sum_{(i,j)\in E} (x_i - x_j)^2$

Node i has degree d_i. So, value x_i^2 needs to be summed up d_i times.
But each edge (i,j) has two endpoints so we need $x_i^2 + x_j^2$

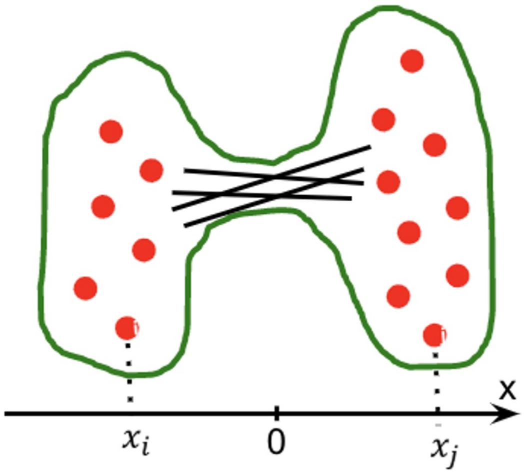
λ_2 as optimization problem

- **What else do we know about x?**
  - $\boldsymbol{x}$ is unit vector: $\sum_i \boldsymbol{x}_i^2 = \mathbf{1}$
  - $\boldsymbol{x}$ is orthogonal to $\mathbf{1}^{st}$ eigenvector $(\mathbf{1}, \dots, \mathbf{1})$ thus: $\sum_i \boldsymbol{x}_i \cdot \mathbf{1} = \sum_i \boldsymbol{x}_i = \mathbf{0}$

- **Remember:**

$$\lambda_2 = \min_{\substack{\text{All labelings} \\ \text{of nodes } i \text{ so} \\ \text{that } \sum x_i = 0}} \frac{\sum_{(i,j)\in E} (x_i - x_j)^2}{\sum_i x_i^2}$$

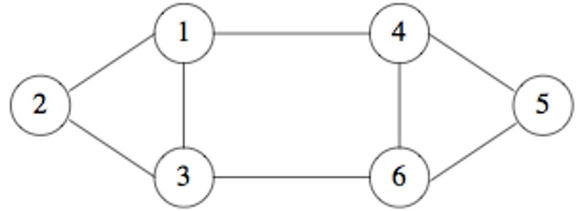
We want to assign values x_i to nodes i such that few edges cross 0.
(we want x_i and x_j to subtract each other)



Balance to minimize

Graph → Laplacian matrix → eigenvector

Recall: Example



Eigenvalue	0	1	3	3	4	5
Eigenvector	1	1	-5	-1	-1	-1
	1	2	4	-2	1	0
	1	1	1	3	-1	1
	1	-1	-5	-1	1	1
	1	-2	4	-2	-1	0
	1	-1	1	3	1	-1

- Use standard math package to find all eigenvalues and eigenvectors
  - (Have not scaled eigenvectors to length 1, but could)
- Second eigenvector has three positive and three negative components
- Suggest obvious partitioning of {1,2,3} and {4,5,6}

So far…

- **How to define a "good" partition of a graph?**
  - Minimize a given graph cut criterion

- **How to efficiently identify such a partition?**
  - Approximate using information provided by the eigenvalues and eigenvectors of a graph

- **Spectral Clustering**
  - Naïve approach:
    - Split at **0**

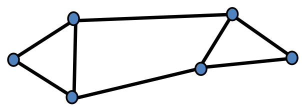
Spectral Clustering Algorithms

6. step.

- **Three basic stages:**
  - **1) Pre-processing**
    - Construct a matrix representation of the graph
  - **2) Decomposition**
    - Compute eigenvalues and eigenvectors of the matrix
    - Map each point to a lower-dimensional representation based on one or more eigenvectors
  - **3) Grouping**
    - Assign points to two or more clusters, based on the new representation

Spectral Partitioning Algorithm

- **1) Pre-processing:**
  - Build Laplacian matrix L of the graph



	1	2	3	4	5	6
1	3	-1	-1	0	-1	0
2	-1	2	-1	0	0	0
3	-1	-1	3	-1	0	0
4	0	0	-1	3	-1	-1
5	-1	0	0	-1	3	-1
6	0	0	0	-1	-1	2

- **2) Decomposition:**
  - Find eigenvalues λ and eigenvectors x of the matrix L

$\lambda =$

0.0
1.0
3.0
3.0
4.0
5.0

$X =$

0.4	0.3	-0.5	-0.2	-0.4	-0.5
0.4	0.6	0.4	-0.4	0.4	0.0
0.4	0.3	0.1	0.6	-0.4	0.5
0.4	-0.3	0.1	0.6	0.4	-0.5
0.4	-0.3	-0.5	-0.2	0.4	0.5
0.4	-0.6	0.4	-0.4	-0.4	0.0

  - Map vertices to corresponding components of λ_2

1	0.3
2	0.6
3	0.3
4	-0.3
5	-0.3
6	-0.6

How do we now find the clusters?

Spectral Partitioning

- **3) Grouping:**
  - Sort components of reduced 1-dimensional vector
  - Identify clusters by splitting the sorted vector in two
- **How to choose a splitting point?**
  - Naïve approaches:
    - Split at **0** or median value
  - More expensive approaches:
    - Attempt to minimize normalized cut in 1-dimension (sweep over ordering of nodes induced by the eigenvector)

1	0.3
2	0.6
3	0.3
4	-0.3
5	-0.3
6	-0.6

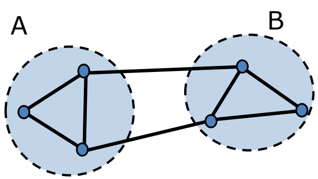
Split at 0:

Cluster A: Positive points

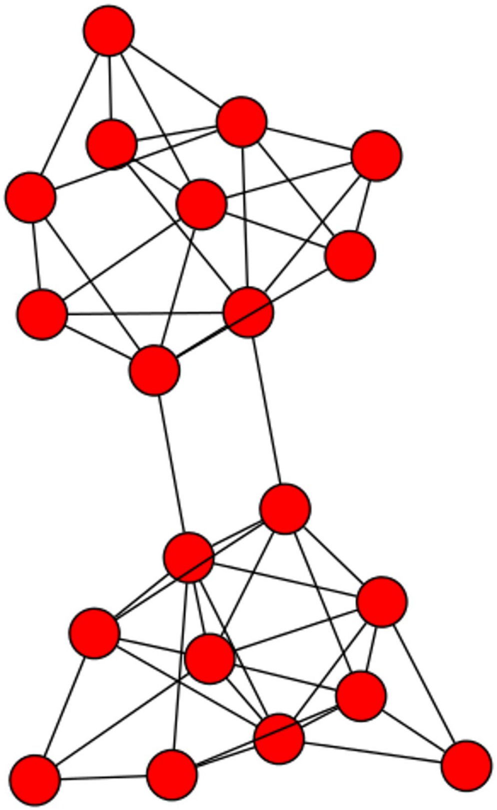
Cluster B: Negative points

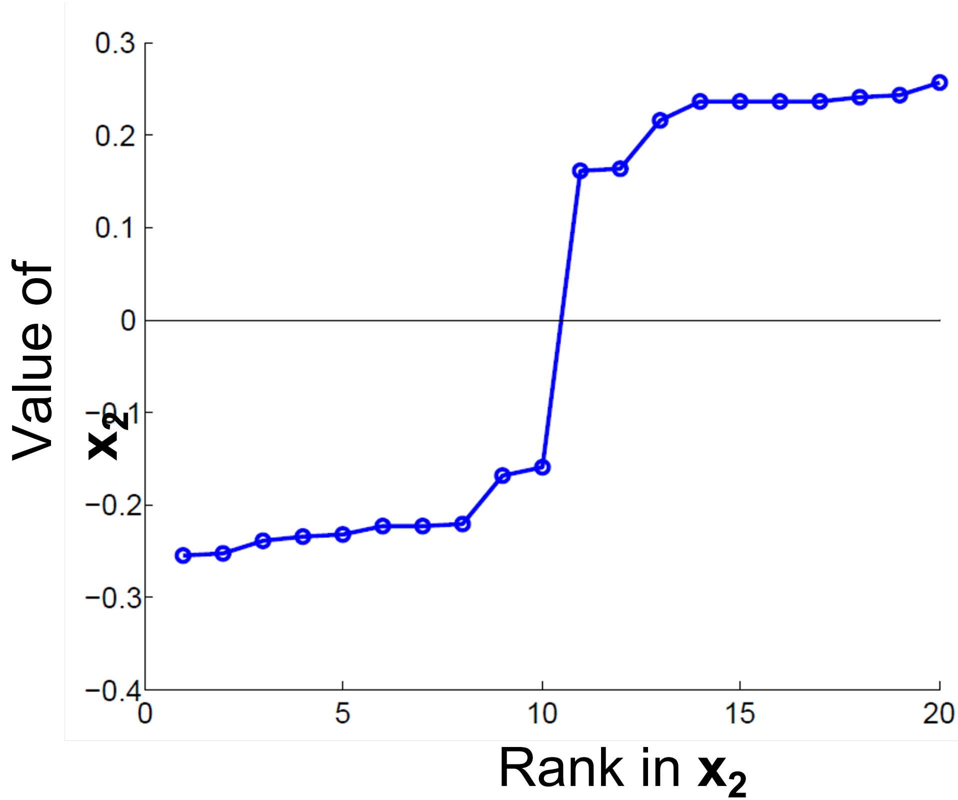
1	0.3
2	0.6
3	0.3

4	-0.3
5	-0.3
6	-0.6

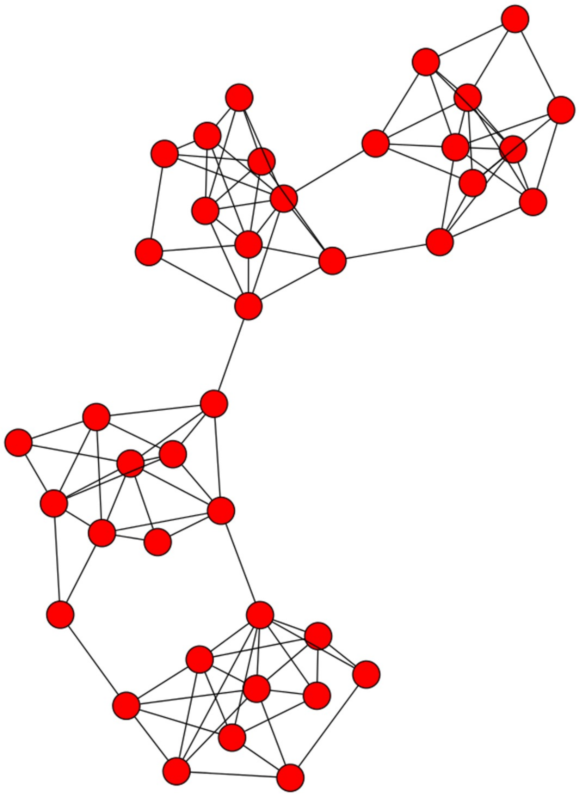


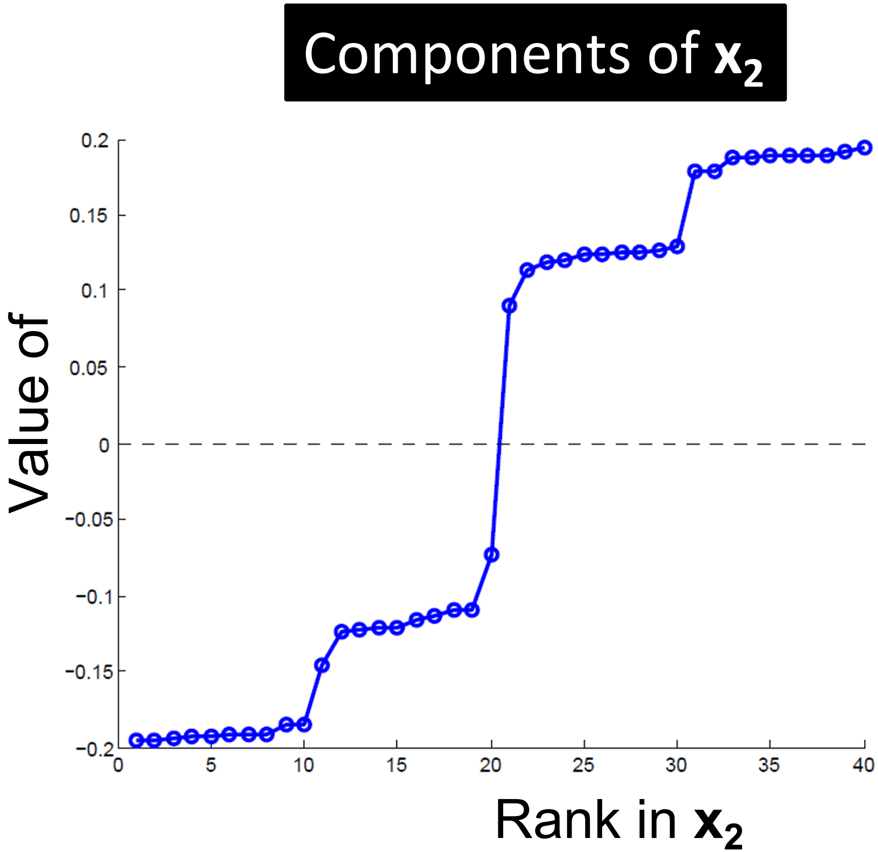
Example: Spectral Partitioning



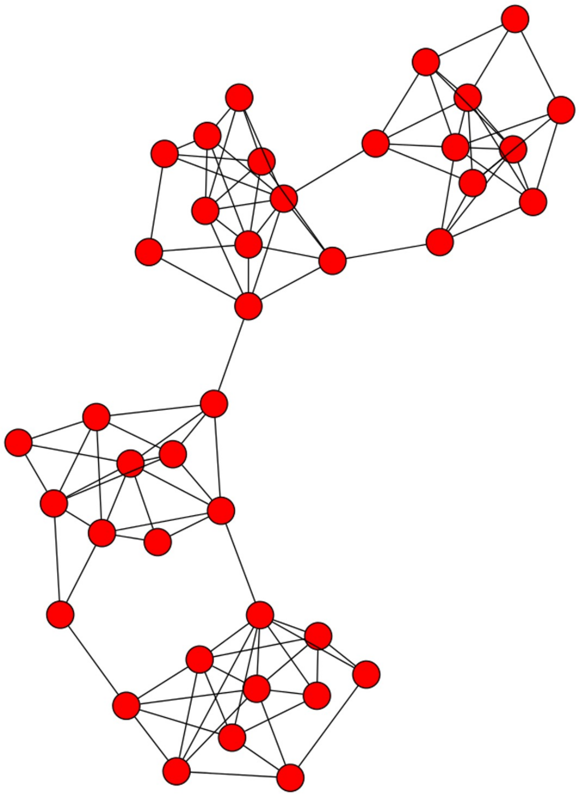


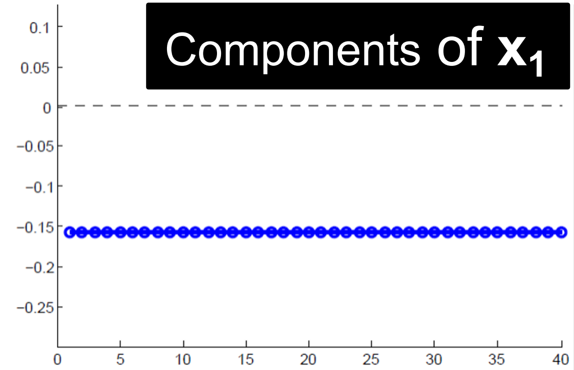
Example: Spectral Partitioning

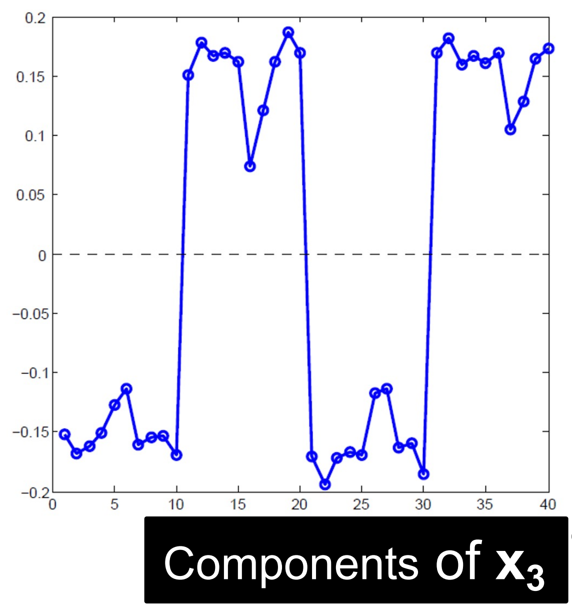




Example: Spectral partitioning







7.

k-Way Spectral Clustering

- How do we partition a graph into k clusters?

- Two basic approaches:
  - **Recursive bi-partitioning** [Hagen et al., '92]
    - Recursively apply bi-partitioning algorithm in a hierarchical divisive manner
    - Disadvantages: Inefficient, unstable
  - **Cluster multiple eigenvectors** [Shi-Malik, '00]
    - Build a reduced space from multiple eigenvectors
    - Commonly used in recent papers
    - Multiple eigenvectors prevent instability due to information loss
    - A preferable approach…

DIRECT DISCOVERY OF COMMUNITIES: TRAWLING

Web community

- Groups of individuals who share common interests, together with the web pages most popular among them
- Web page collections with a shared topic

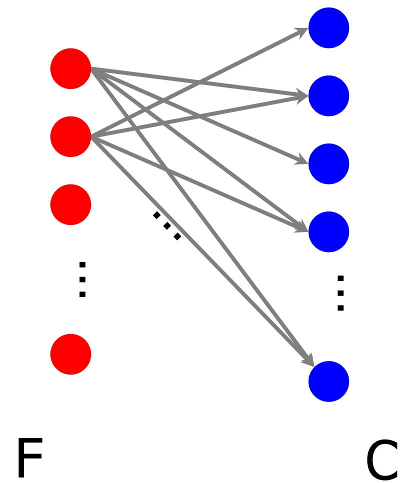
Types of Communities

- Explicitly-defined
  - Communities that manifest themselves as newsgroups or as resource collections on directories such as Yahoo!
- Implicitly-defined
  - Communities that result from nature of content-creation of the web

2、

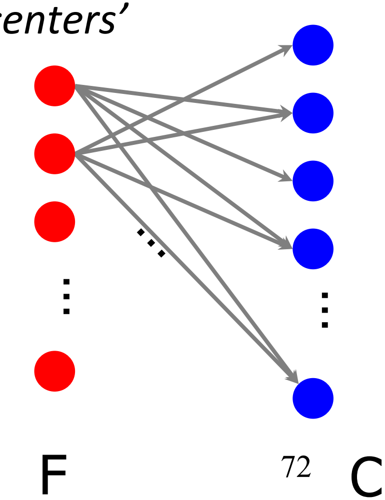
Terms and Definitions (1)

- (1) *Directed Bipartite Graph:* A graph whose nodes set can be partitioned into two sets *F* and *C*, and every directed edge in the graph is from a node *u* in *F* to a node *v* in *C*



Terms and Definitions (2)

- *A Complete Bipartite Graph:* A bipartite graph that contains all possible edges between a vertex of *F* and a vertex of *C*
- *Core:* A complete bipartite subgraph with at least i nodes from *F* and at least *j* nodes from *C*
  - In the web world, the *i* pages the contains the links are referred to as '*fans*' and the *j* pages that are referenced as '*centers*'

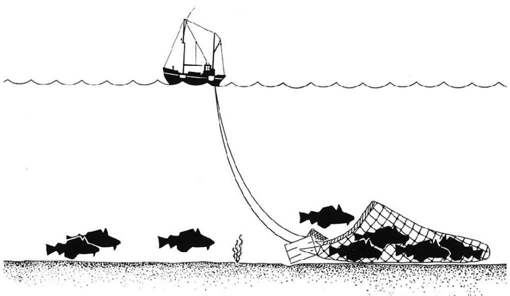


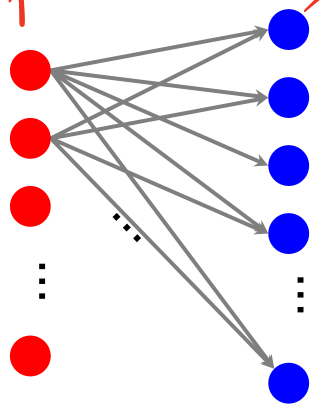
3. Trawling

- Searching for small communities in the Web graph
- What is the signature of a community / discussion in a Web graph?

seen as basket

seen as item





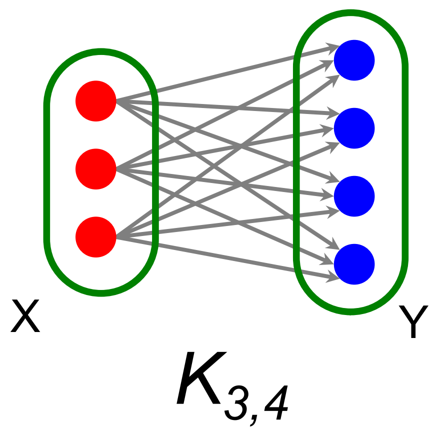
Dense 2-layer

Use this to define "topics": What the same people on the left talk about on the right

Intuition: Many people all talking about the same things

Searching for Small Communities

- **A more well-defined problem:**
  Enumerate complete bipartite subgraphs $K_{s,t}$
  - Where $K_{s,t}$: s nodes on the "left" where each links to the same t other nodes on the "right"



$|X| = s = 3$
$|Y| = t = 4$

Fully connected

Frequent Itemset Enumeration

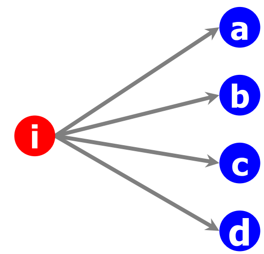
- **Market basket analysis.** Setting:
  - **Market:** Universe U of n items
  - **Baskets:** m subsets of U: $S_1, S_2, \ldots, S_m \subseteq U$
    (S_i is a set of items one person bought)
  - **Support:** Frequency threshold f
- **Goal:**
  - **Find all subsets T s.t. $T \in S_i$ of at least f sets S_i**
    (items in T were bought together at least f times)
- **What's the connection between the itemsets and complete bipartite graphs?**

From Itemsets to Bipartite $K_{s,t}$

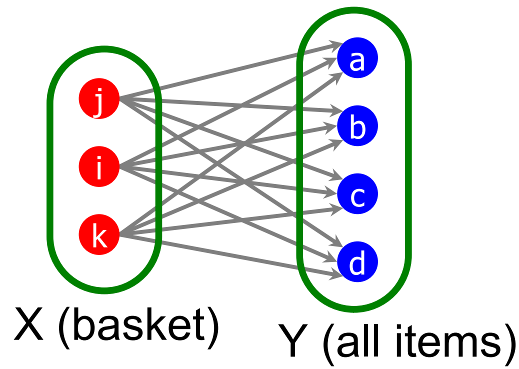
Frequent itemsets = complete bipartite graphs!

How?

- View each node i as a set S_i of nodes i points to
- $K_{s,t}$ = a set Y of size t (all items) that occurs in s (a basket) sets S_i
- Looking for $K_{s,t}$? set of frequency threshold to s and look at layer t – all frequent sets of size t



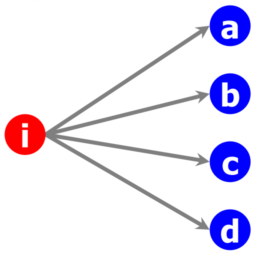
S_i={a,b,c,d}



s … minimum support (|X|=s)
t … itemset size (|Y|=t)

From Itemsets to Bipartite $K_{s,t}$

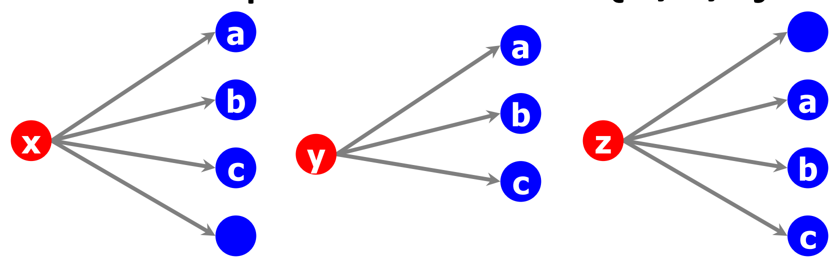
View each node ***i*** as a set $\boldsymbol{S_i}$ of nodes ***i*** points to



S_i={a,b,c,d}

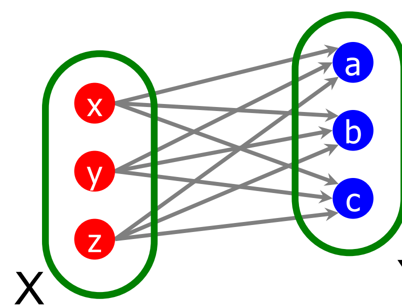
Find frequent itemsets:
- **s** … minimum support
- **t** … itemset size

Say we find a **frequent itemset** ***Y={a,b,c}*** of supp ***s***
So, there are ***s*** nodes that link or point to all of **{a,b,c}**:

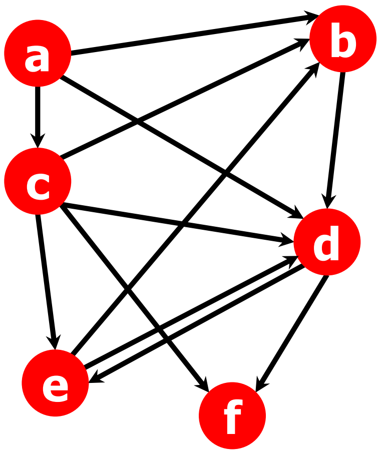


We found *$K_{s,t}$*!

$K_{s,t}$ = a set ***Y*** of size ***t*** that occurs in *s* sets $\boldsymbol{S_i}$



Example



Itemsets:

a = {b,c,d}

b = {d}

c = {b,d,e,f}

d = {e,f}

e = {b,d}

f = {}

- Support threshold s=2
  - **{b,d}**: support 3
  - **{e,f}**: support 2
- **And we just found 2 bipartite subgraphs:**