Community Detection in Graphs

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
http://cs246.stanford.edu

Link Analysis:
HITS: Hubs and Authorities

Hubs and Authorities

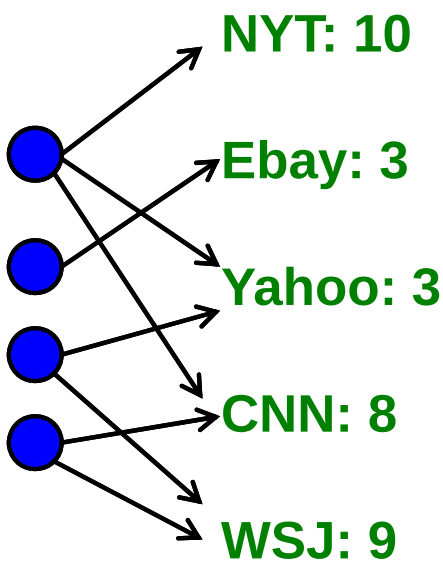
- **HITS (Hypertext-Induced Topic Selection)**
  - **Is a measure of importance of pages or documents, similar to PageRank**
  - Proposed at around same time as PageRank ('98)
- **Goal**: Say we want to find good newspapers
  - Don't just find newspapers. Find "experts" – people who link in a coordinated way to good newspapers
- **Idea: Links as votes**
  - **Page is more important if it has more links**
    - In-coming links? Out-going links?

Finding newspapers

- **Hubs and Authorities**

  Each page has 2 scores:

  - **Quality as an expert (hub):**
    - Total sum of votes of authorities pointed to
  - **Quality as a content (authority):**
    - Total sum of votes coming from experts



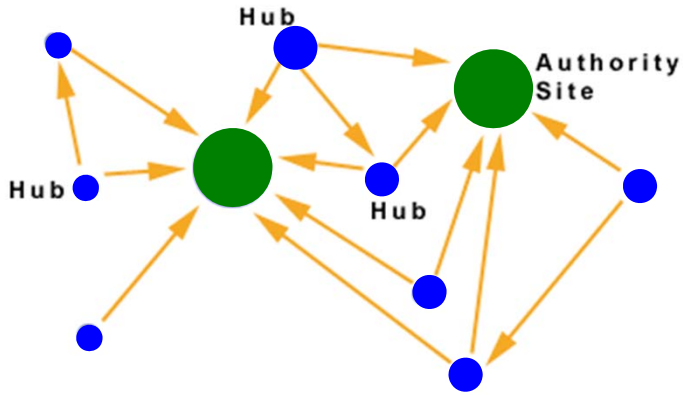
- **Principle of repeated improvement**

Hubs and Authorities

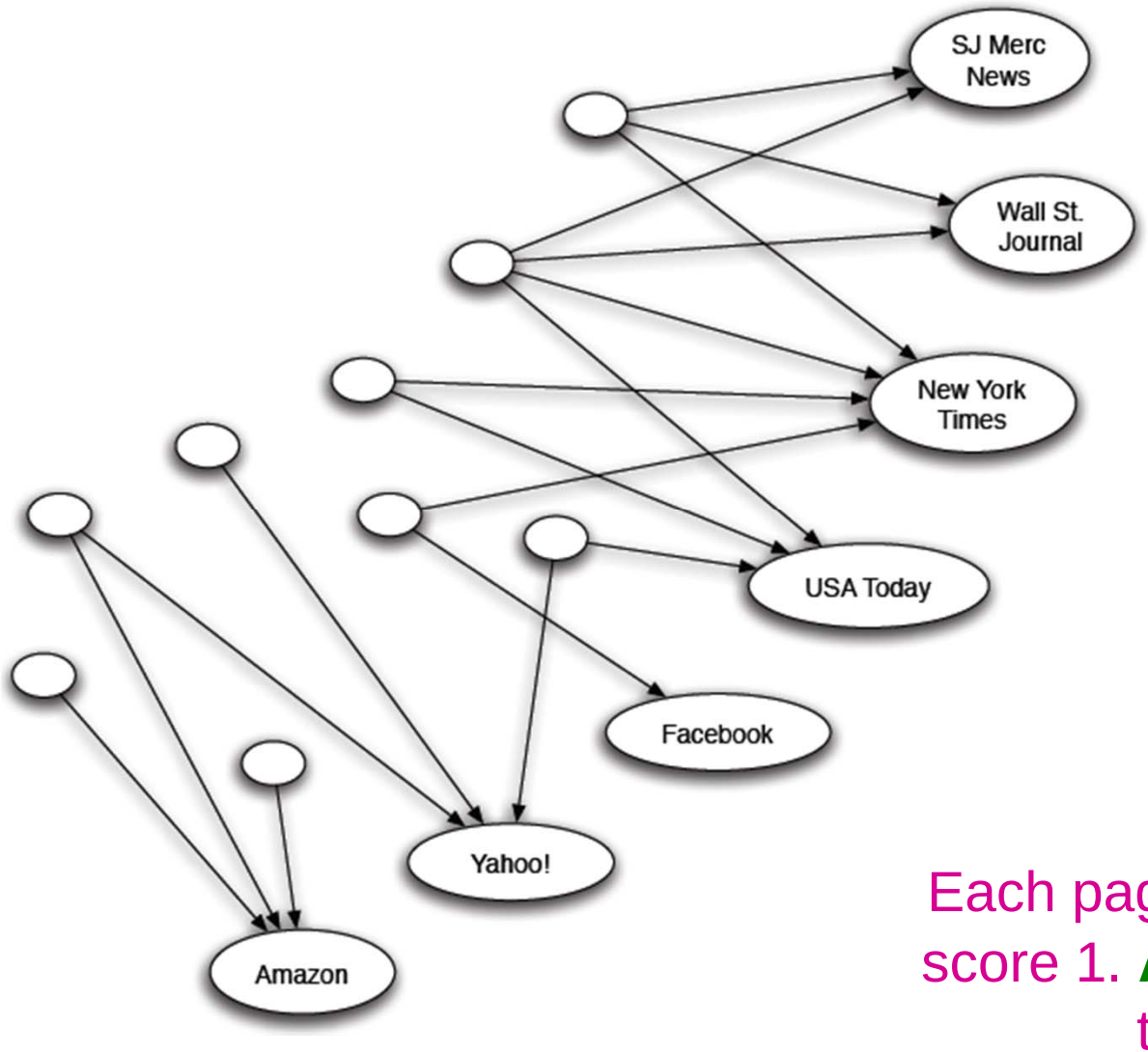
Interesting pages fall into two classes:

1. **Authorities** are pages containing useful information
   - Newspaper home pages
   - Course home pages
   - Home pages of auto manufacturers

2. **Hubs** are pages that link to authorities
   - List of newspapers
   - Course bulletin
   - List of US auto manufacturers



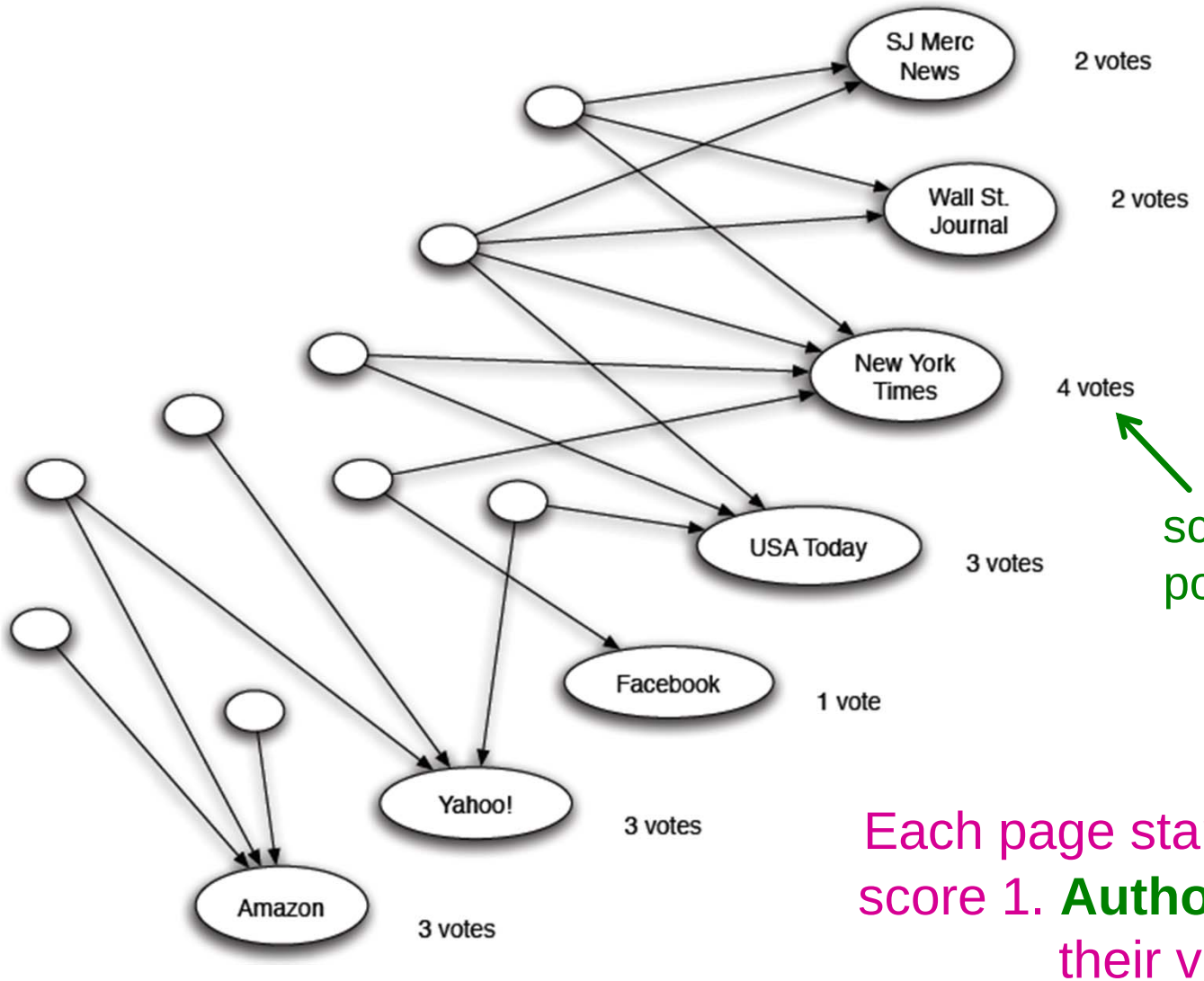
Counting in-links: Authority



Each page starts with **hub** score 1. **Authorities** collect their votes

(Note this is idealized example. In reality graph is not bipartite and each page has both the hub and authority score)

Counting in-links: Authority



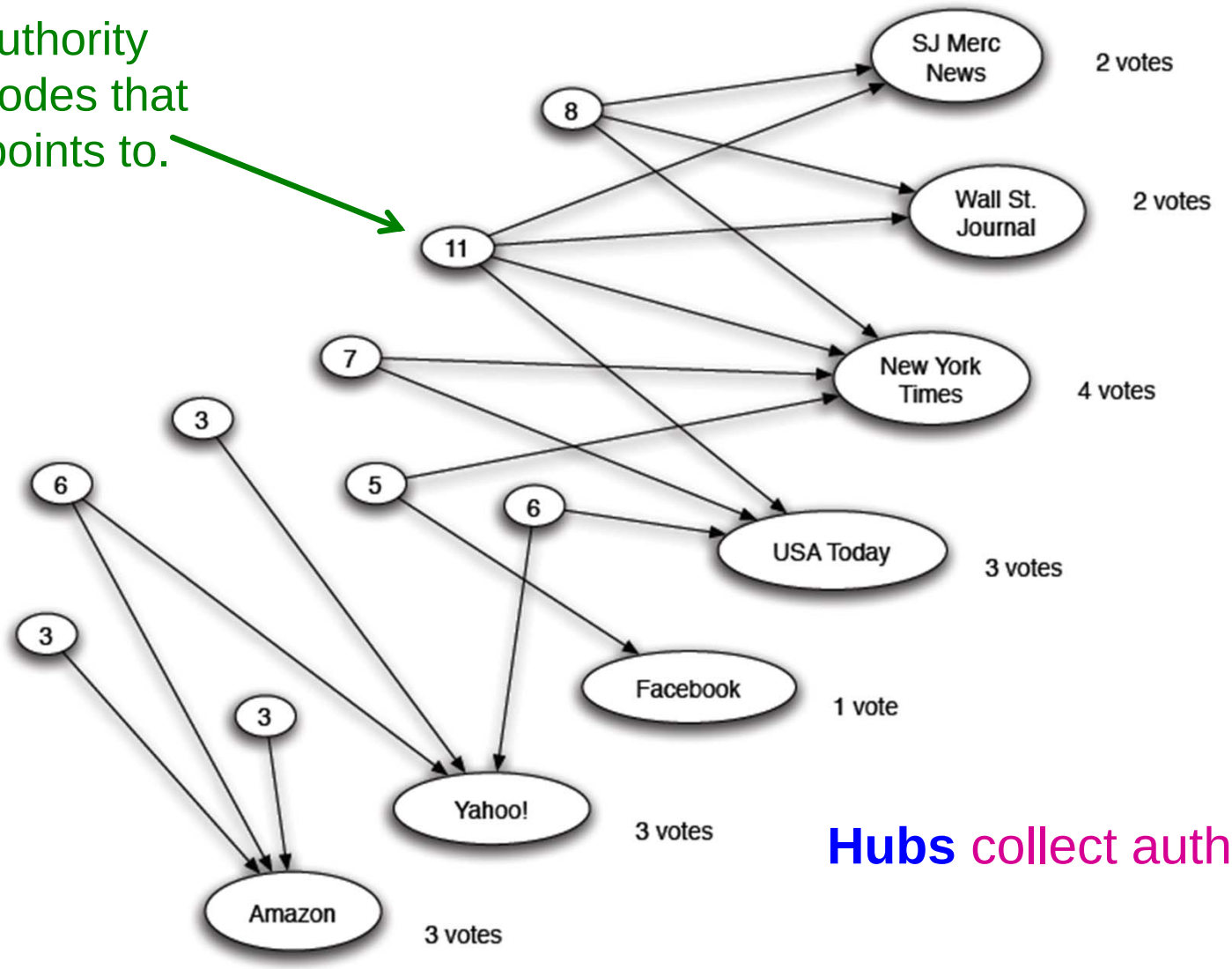
Sum of **hub** scores of nodes pointing to NYT.

Each page starts with **hub** score 1. **Authorities** collect their votes

(Note this is idealized example. In reality graph is not bipartite and each page has both the hub and authority score)

Expert Quality: Hub

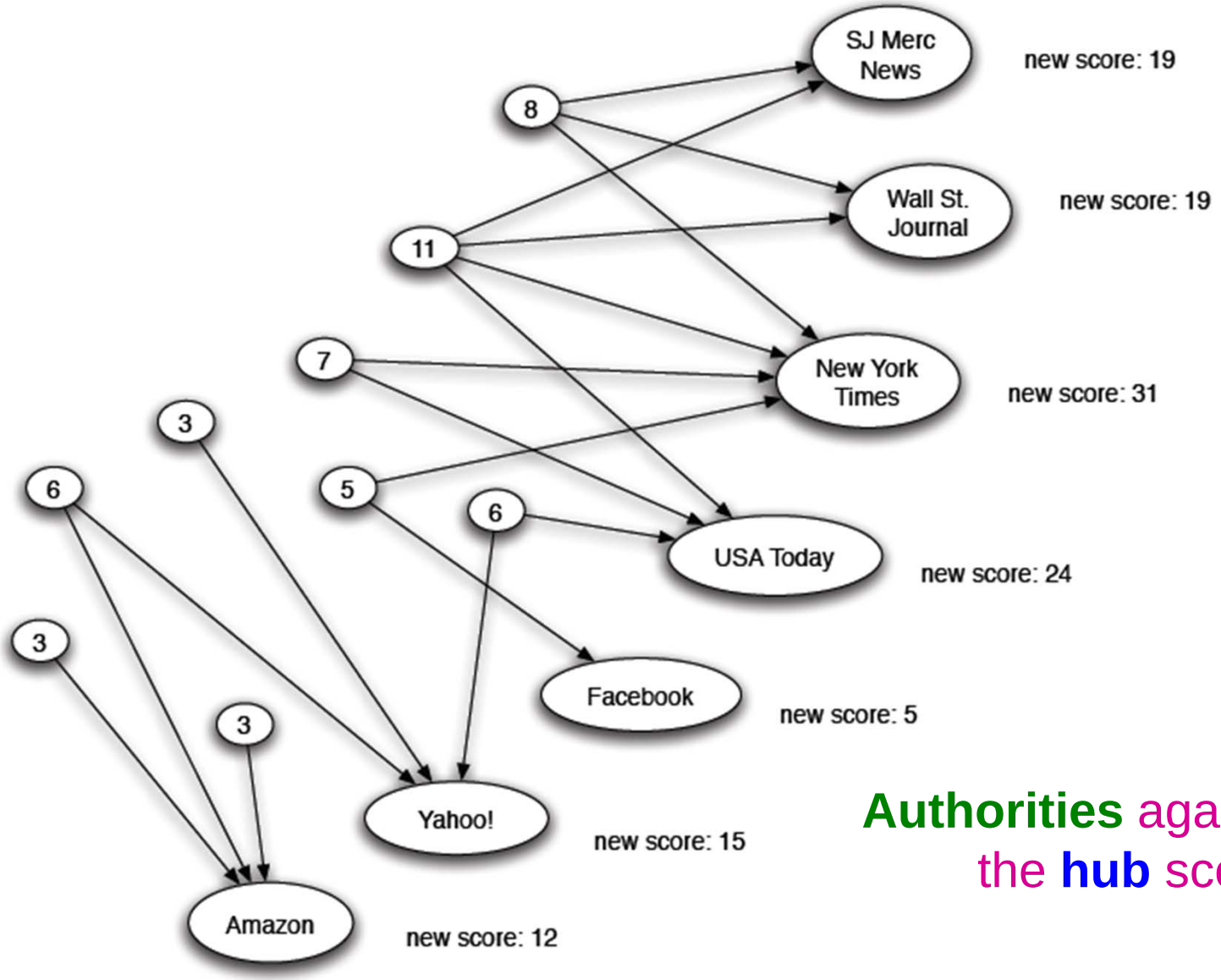
Sum of authority scores of nodes that the node points to.



Hubs collect authority scores

(Note this is idealized example. In reality graph is not bipartite and each page has both the hub and authority score)

Reweighting



Authorities again collect the **hub** scores

(Note this is idealized example. In reality graph is not bipartite and each page has both the hub and authority score)

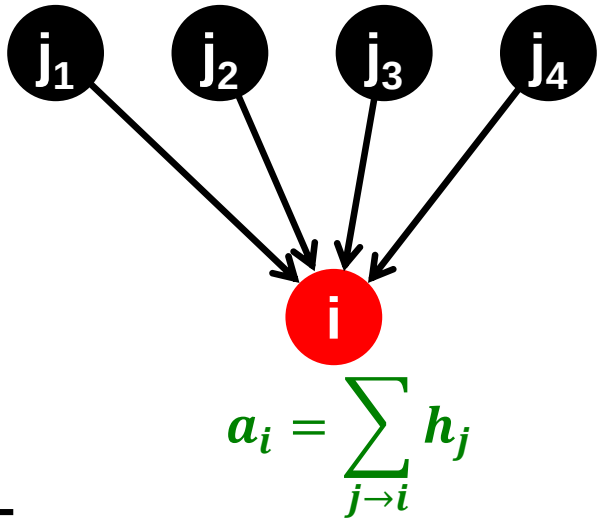
Mutually Recursive Definition

- **A good hub links to many good authorities**
- **A good authority is linked from many good hubs**
- **Model using two scores for each node:**
  - **Hub** score and **Authority** score
  - Represented as vectors $\boldsymbol{h}$ and $\boldsymbol{a}$

[Kleinberg '98]

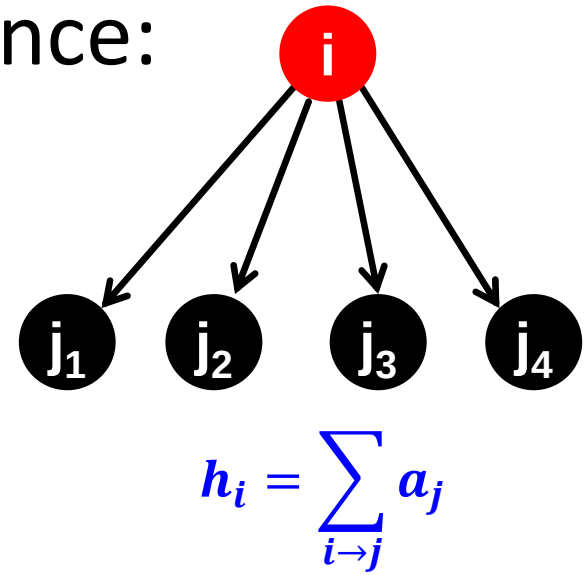
Hubs and Authorities

- **Each page i has 2 scores:**
  - Authority score: $\boldsymbol{a_i}$
  - Hub score: $\boldsymbol{h_i}$



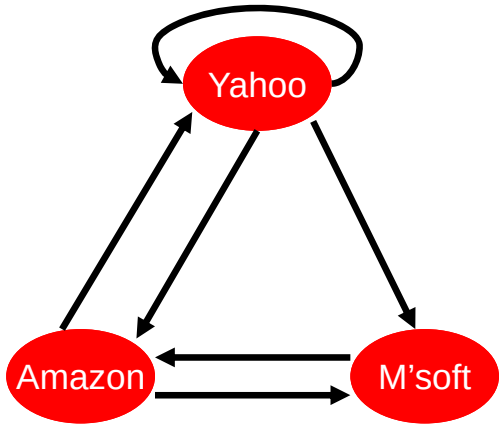
HITS algorithm: n…number of node in a graph

- Initialize: $\boldsymbol{a_i} = 1/\sqrt{n},\ \boldsymbol{h_i} = 1/\sqrt{n}$
- Then keep iterating until convergence:
  - $\forall i$: **Authority**: $\boldsymbol{a_i} = \sum_{j \to i} \boldsymbol{h_j}$
  - $\forall i$: **Hub**: $\boldsymbol{h_i} = \sum_{i \to j} \boldsymbol{a_j}$
  - $\forall i$: **Normalize a, h such that:**
    $\sum_i \boldsymbol{a_i^2} = 1, \sum_i \boldsymbol{h_i^2} = 1$



Example

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$



h(yahoo)	=	.58	.80	.80	.79	. . .	.788
h(amazon)	=	.58	.53	.53	.57	. . .	.577
h(m'soft)	=	.58	.27	.27	.23	. . .	.211

a(yahoo)	=	.58	.58	.62	.62	. . .	.628
a(amazon)	=	.58	.58	.49	.49	. . .	.459
a(m'soft)	=	.58	.58	.62	.62	. . .	.628

[Kleinberg '98]

Hubs and Authorities

- **HITS converges to a single stable point**
- **Notation:**
  - Vector $\boldsymbol{a} = (a_1 \ldots, a_n), \quad \boldsymbol{h} = (h_1 \ldots, h_n)$
  - Adjacency matrix $\boldsymbol{A}$ (n x n): $\boldsymbol{A_{ij}} = 1$ if $i \rightarrow j$
- **Then $\boldsymbol{h_i} = \sum_{i \rightarrow j} \boldsymbol{a_j}$**

  can be rewritten as $\boldsymbol{h_i} = \sum_j \boldsymbol{A_{ij}} \cdot \boldsymbol{a_j}$

  So: $\boldsymbol{h} = \boldsymbol{A} \cdot \boldsymbol{a}$
- **Similarly, $\boldsymbol{a_i} = \sum_{j \rightarrow i} \boldsymbol{h_j}$**

  can be rewritten as $\boldsymbol{a_i} = \sum_j \boldsymbol{A_{ji}} \cdot \boldsymbol{h_j} = \boldsymbol{A^T} \cdot \boldsymbol{h}$

Hub and Authority Equations

- The **hub** score of page ***i*** is proportional to the sum of the **authority** scores of the pages it links to: $h = \lambda A a$
  - $\boldsymbol{\lambda}$ is a scale factor: $\lambda = \frac{1}{\sqrt{\sum_i h_i^2}}$
- The **authority** score of page ***i*** is proportional to the sum of the hub scores of the pages it is linked from: $a = \mu A^T h$
  - $\boldsymbol{\mu}$ is scale factor: $\mu = \frac{1}{\sqrt{\sum_i a_i^2}}$

Hubs and Authorities

- **HITS algorithm in vector notation:**
  - Set: $\boldsymbol{a_i} = \boldsymbol{h_i} = \frac{\mathbf{1}}{\sqrt{n}}$

  Repeat until convergence:
  - $\boldsymbol{h} = \boldsymbol{A} \cdot \boldsymbol{a}$
  - $\boldsymbol{a} = \boldsymbol{A}^{\boldsymbol{T}} \cdot \boldsymbol{h}$
  - Normalize $\boldsymbol{a}$ and $\boldsymbol{h}$

Convergence criterion:

$$\sum_i \left(h_i^{(t)} - h_i^{(t-1)}\right)^2 < \varepsilon$$

$$\sum_i \left(a_i^{(t)} - a_i^{(t-1)}\right)^2 < \varepsilon$$

- **Then:** $\boldsymbol{a} = \boldsymbol{A}^{\boldsymbol{T}} \cdot (\underbrace{\boldsymbol{A} \cdot \boldsymbol{a}}_{\text{new } \boldsymbol{h}})$ (new $\boldsymbol{a}$)
- **Thus, in $2k$ steps:**

$$a = (A^T \cdot A)^k \cdot a$$
$$h = (A \cdot A^T)^k \cdot h$$

a is updated (in 2 steps):

$$a = A^T (A\, a) = (A^T A)\, a$$

h is updated (in 2 steps):

$$h = A\, (A^T h) = (A\, A^T)\, h$$

Repeated matrix powering

Existence and Uniqueness

- $\boldsymbol{h} = \lambda A \boldsymbol{a}$
- $\boldsymbol{a} = \mu A^T \boldsymbol{h}$
- $\boldsymbol{h} = \lambda \mu A A^T \boldsymbol{h}$
- $\boldsymbol{a} = \lambda \mu A^T A \boldsymbol{a}$

$\lambda = 1/\sqrt{\sum_i h_i^2}$

$\mu = 1/\sqrt{\sum_i a_i^2}$

- Under reasonable assumptions about **A**, HITS **converges to vectors h^* and a^***:
  - $\boldsymbol{h}^*$ is the **principal eigenvector** of matrix $\boldsymbol{A}\,\boldsymbol{A}^T$
  - $\boldsymbol{a}^*$ is the **principal eigenvector** of matrix $\boldsymbol{A}^T\boldsymbol{A}$

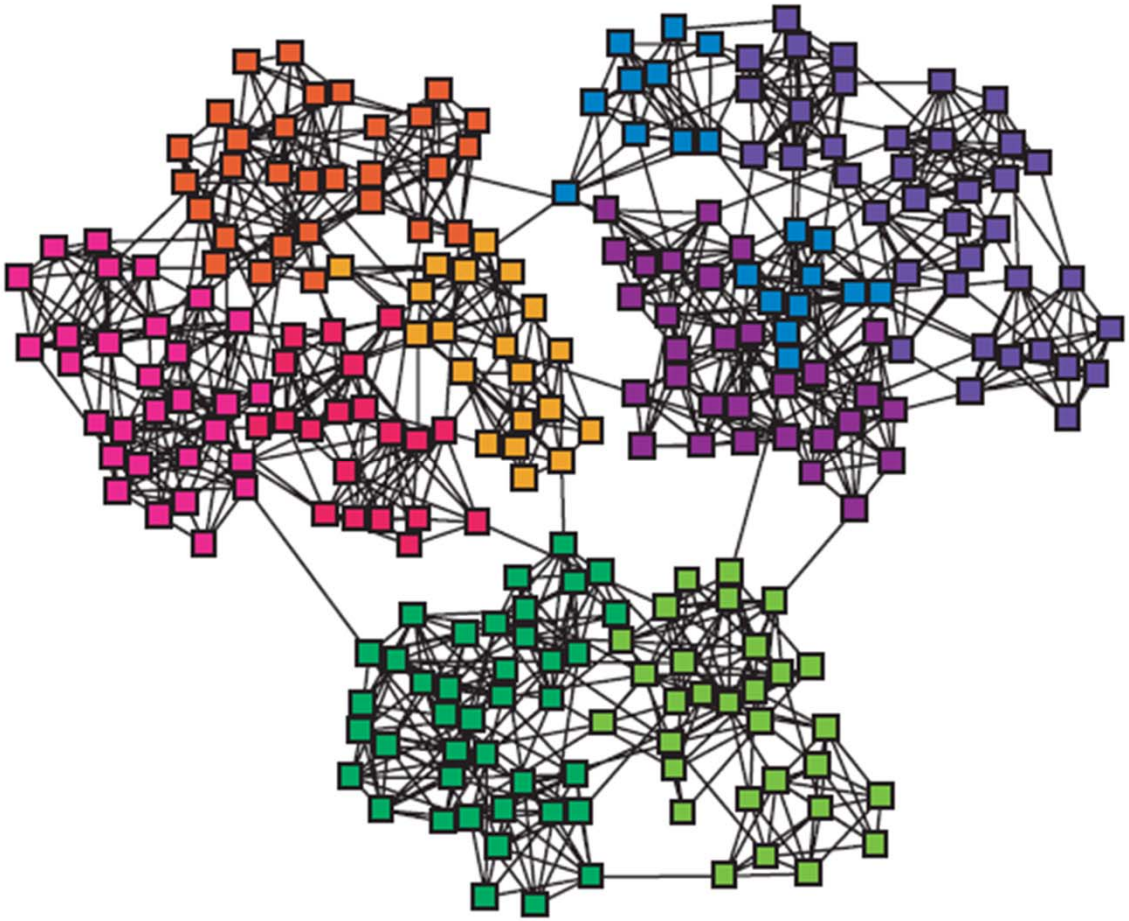
PageRank and HITS

- **PageRank and HITS are two solutions to the same problem:**
  - **What is the value of an in-link from *u* to *v*?**
  - In the PageRank model, the value of the link depends on the links **into** *u*
  - In the HITS model, it depends on the value of the other links **out of** *u*
- **The destinies of PageRank and HITS post-1998 were very different**

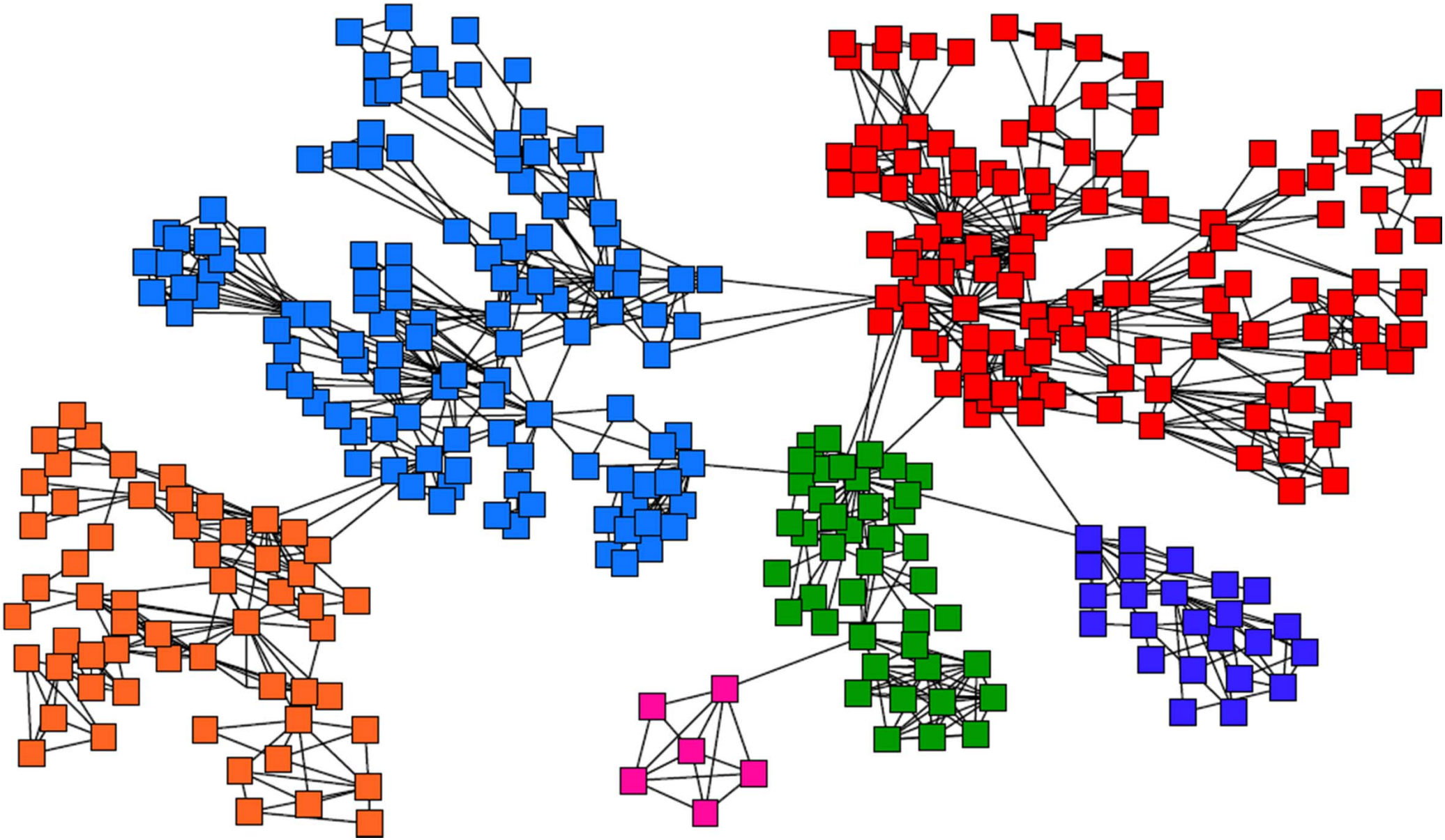
New topic:
Finding Communities in Networks using PageRank

Networks & Communities

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

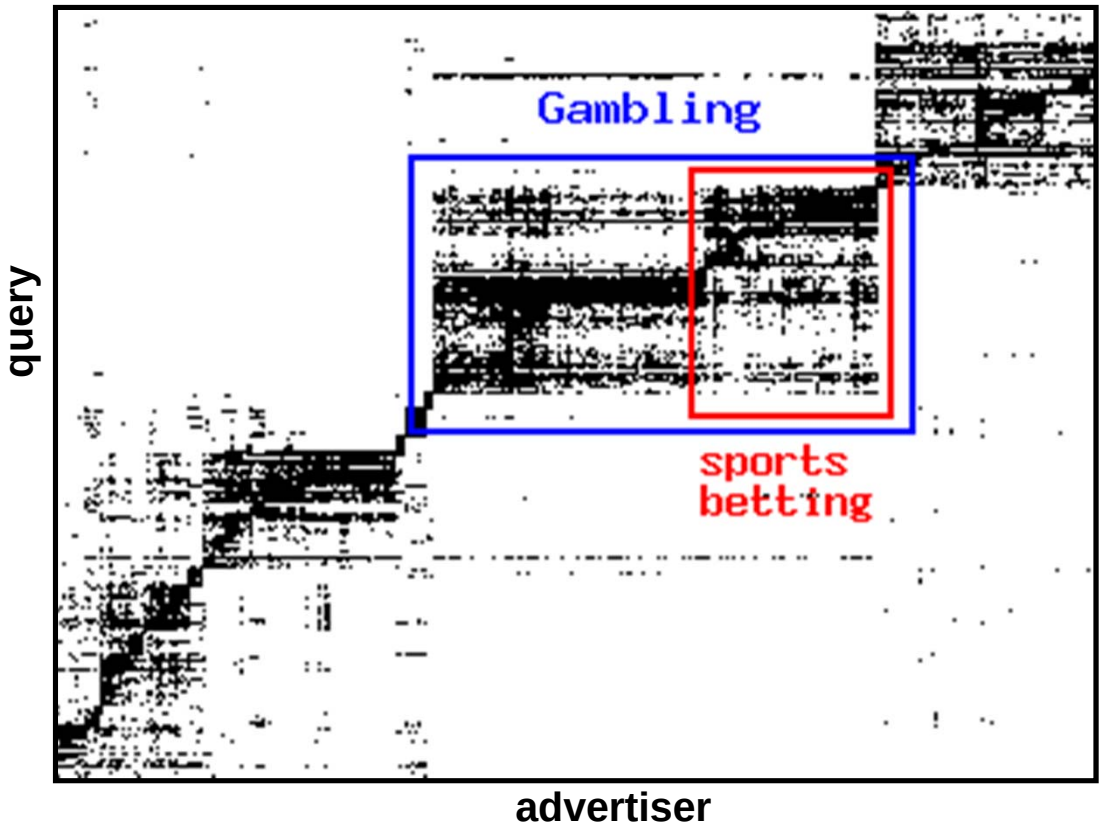


Goal: Find Densely Linked Clusters



Micro-Markets in Sponsored Search

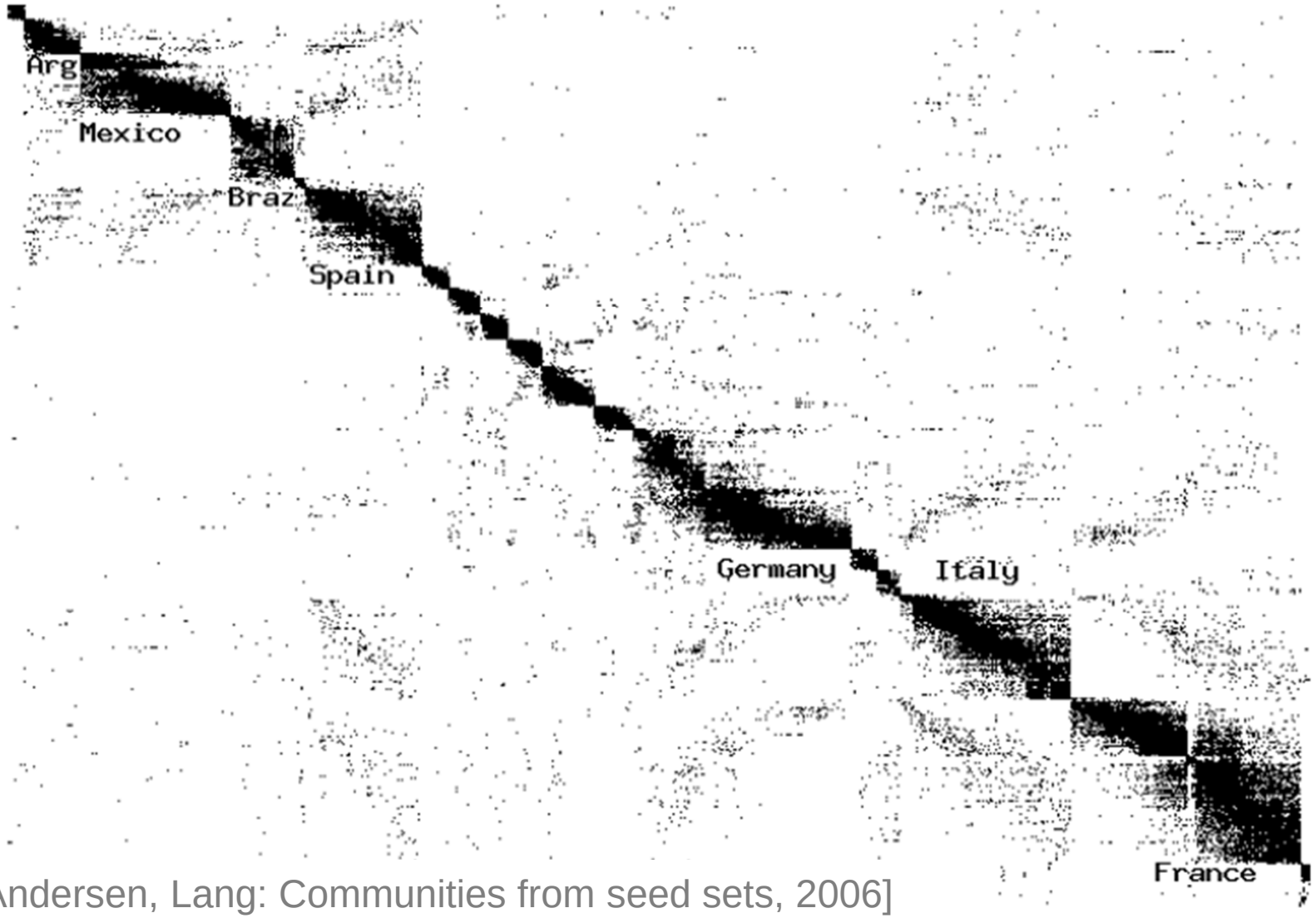
- **Find micro-markets by partitioning the query-to-advertiser graph:**



[Andersen, Lang: Communities from seed sets, 2006]

Movies and Actors

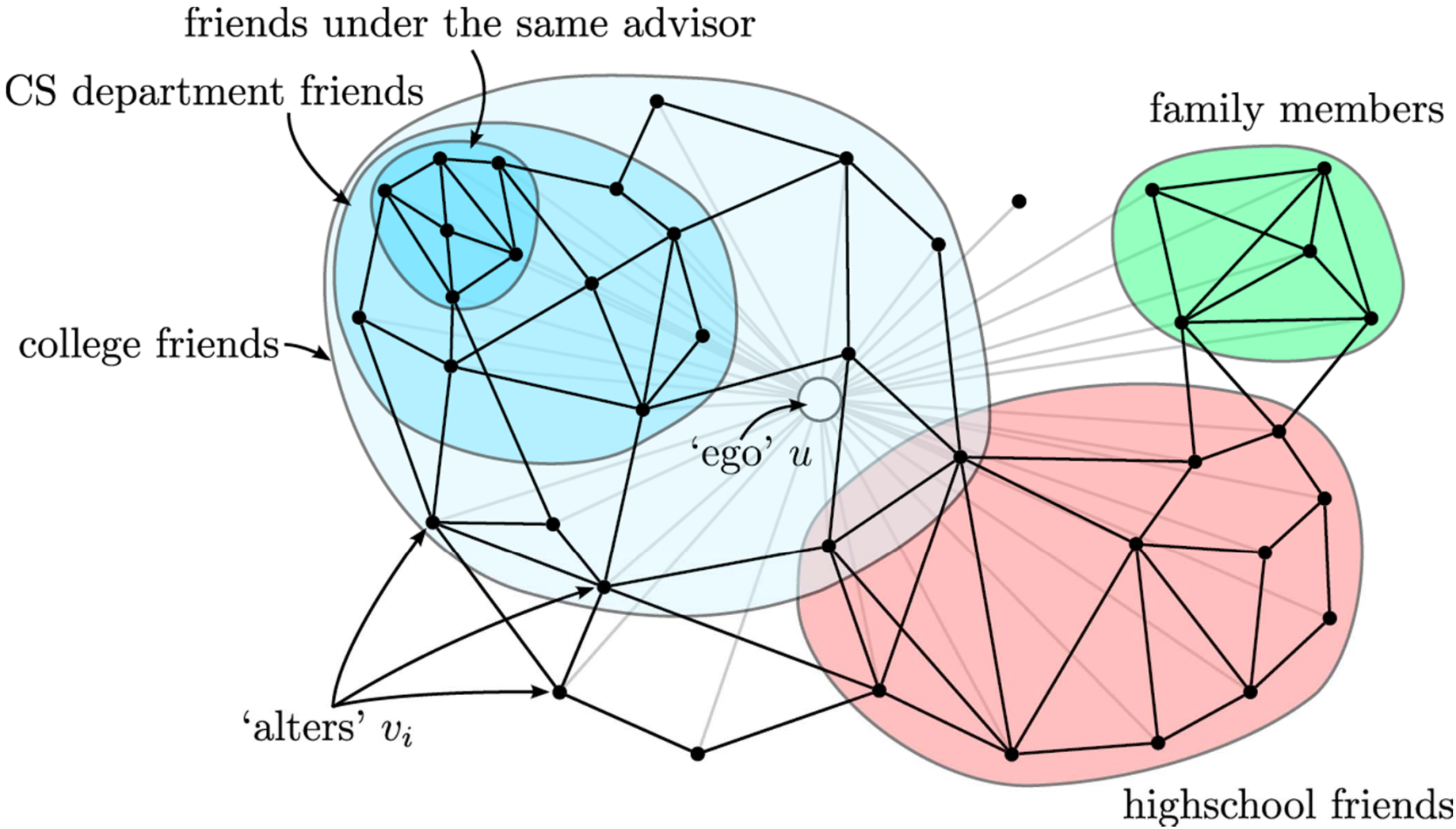
- **Clusters in Movies-to-Actors graph:**



[Andersen, Lang: Communities from seed sets, 2006]

Twitter & Facebook

- **Discovering social circles, circles of trust:**



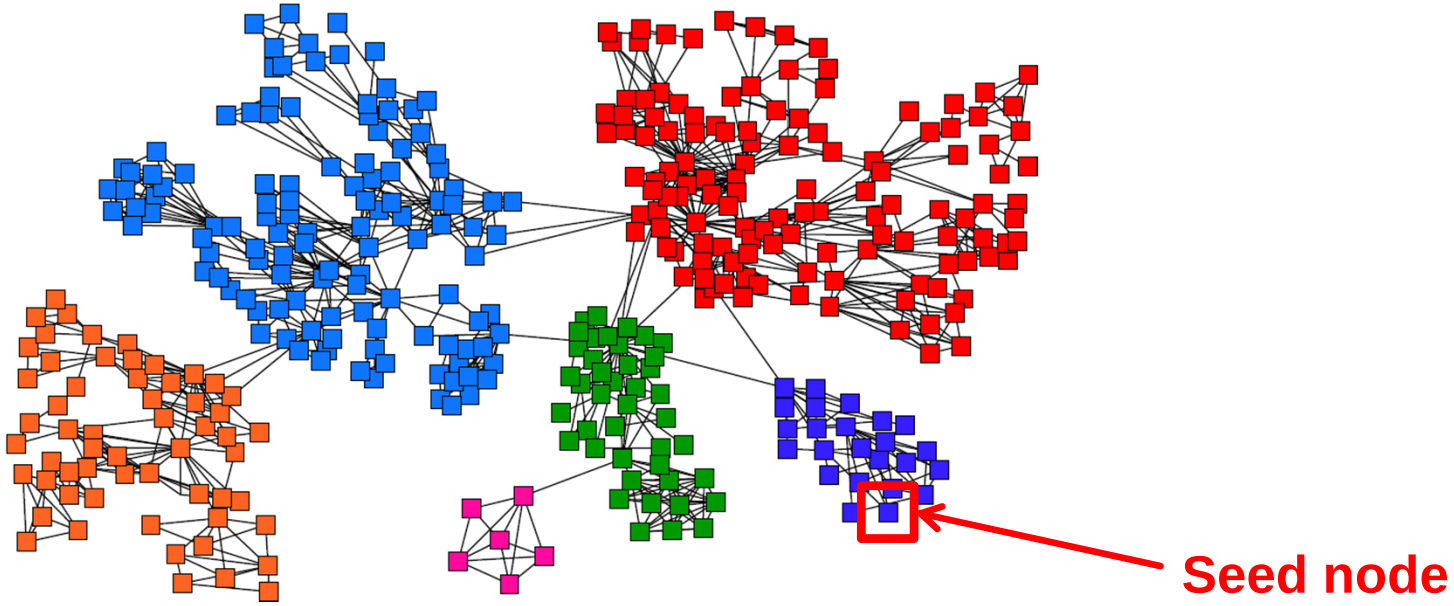
[McAuley, Leskovec: Discovering social circles in ego networks, 2012]

The Setting

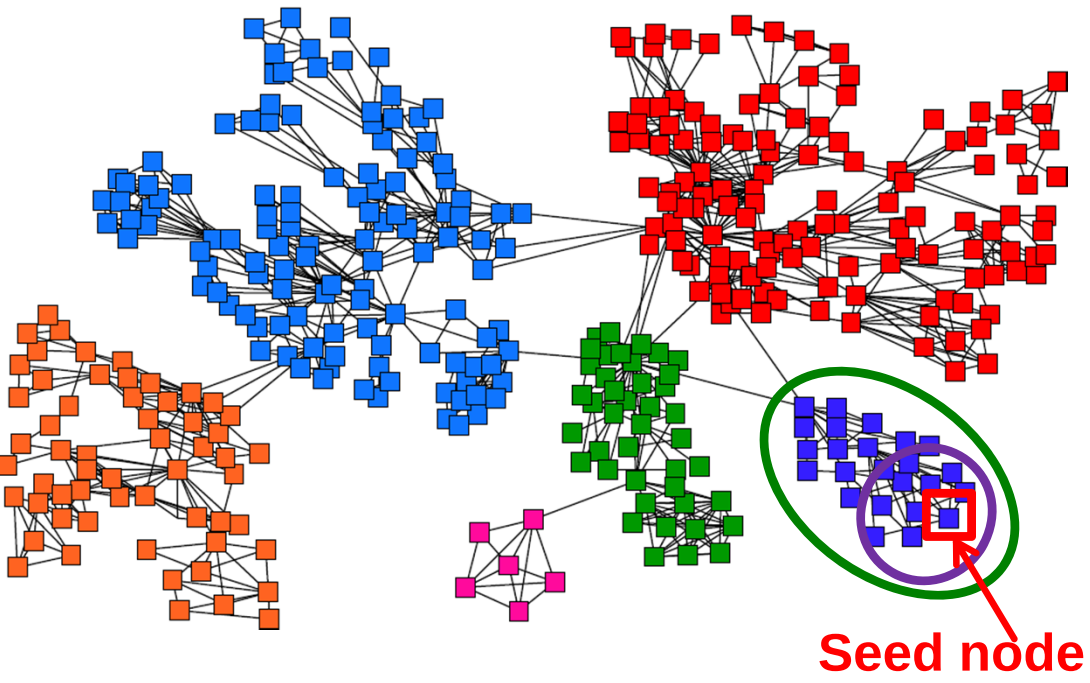
- **Graph is large**
  - **Assume the graph fits in main memory**
    - For example, to work with a 200M node and 2B edge graph one needs approx. 16GB RAM
  - But the graph is too big for running anything more than linear time algorithms
- **We will cover a PageRank based algorithm for finding dense clusters**
  - **The runtime of the algorithm will be proportional to the cluster size (not the graph size!)**

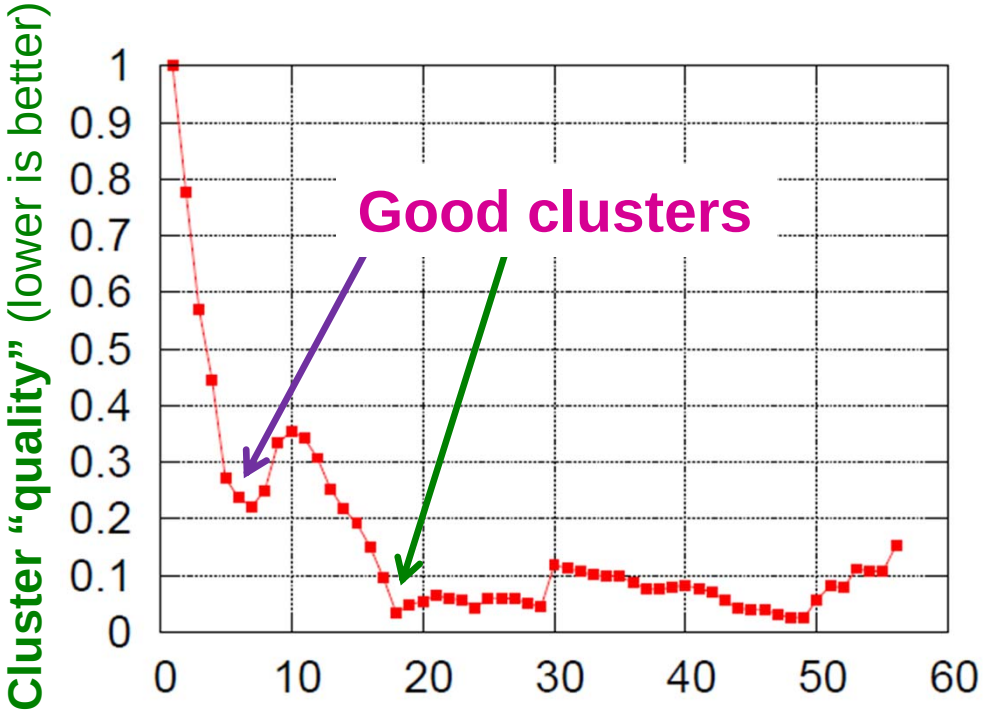
Idea: Seed Nodes

- **Discovering clusters based on seed nodes**
  - **Given:** Seed node **s**
  - Compute (approximate) **P**ersonalized **P**age**R**ank (**PPR**) around node **s** (teleport set={**s**})
  - Idea is that if **s** belongs to a nice cluster, the random walk will get **trapped** inside the cluster



Seed Node: Intuition

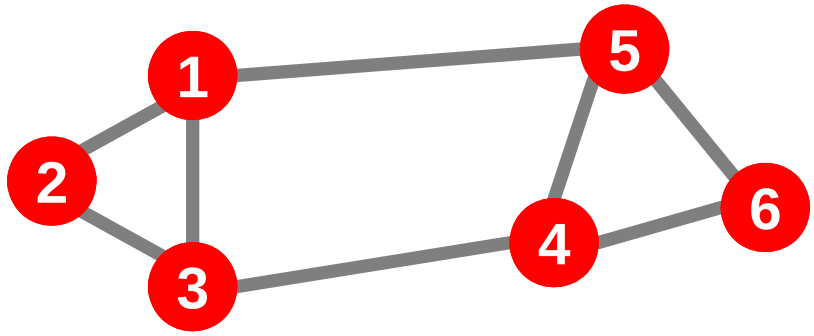




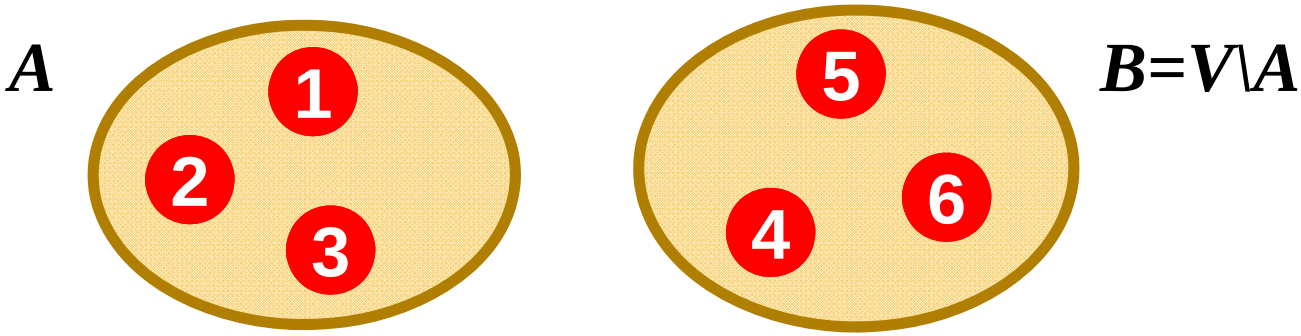
- **Algorithm outline:**
  - Pick a seed node ***s*** of interest
  - Run **PPR** with teleport set = {***s***}
  - Sort the nodes by the decreasing **PPR** score
  - **Sweep** over the nodes and find **good clusters**

What makes a good cluster?

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



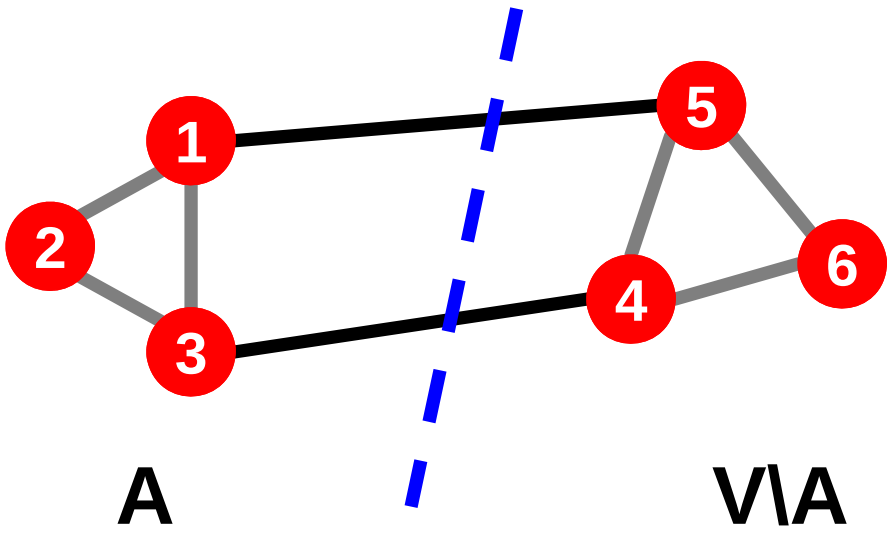
- **Partitioning task:**
  - Divide vertices into 2 disjoint groups $A, B = V \setminus A$



- **Question:**
  - **How can we define a "good" cluster in G?**

What makes a good cluster?

- **What makes a good cluster?**
  - **Maximize the number of within-cluster connections**
  - **Minimize the number of between-cluster connections**

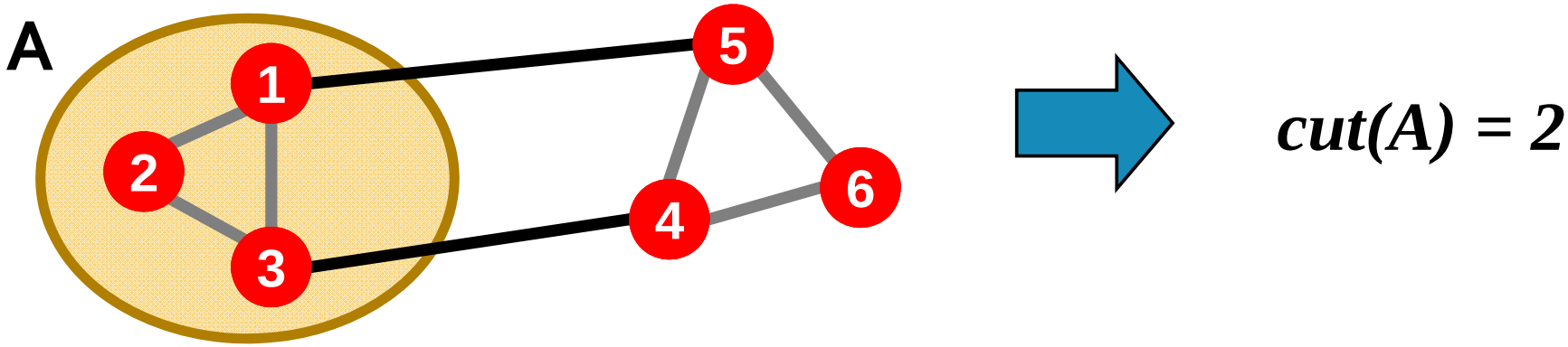


Graph Cuts

- **Express cluster quality as a function of the "edge cut" of the cluster**
- **Cut:** Set of edges with only one node in the cluster:

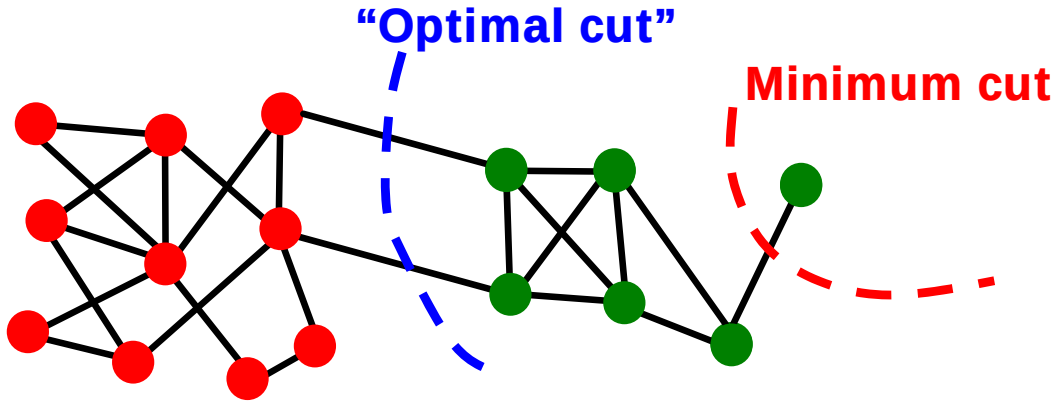
$$cut(A) = \sum_{i \in A, j \notin A} w_{ij}$$

Note: This works for weighed and unweighted (set all $w_{ij}=1$) graphs



Cut Score

- **Partition quality: Cut score**
  - Quality of a cluster is the weight of connections pointing outside the cluster
- **Degenerate case:**



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

[Shi-Malik]

Graph Partitioning Criteria

- **Criterion: Conductance:**
  Connectivity of the group to the rest of the network relative to the density of the group

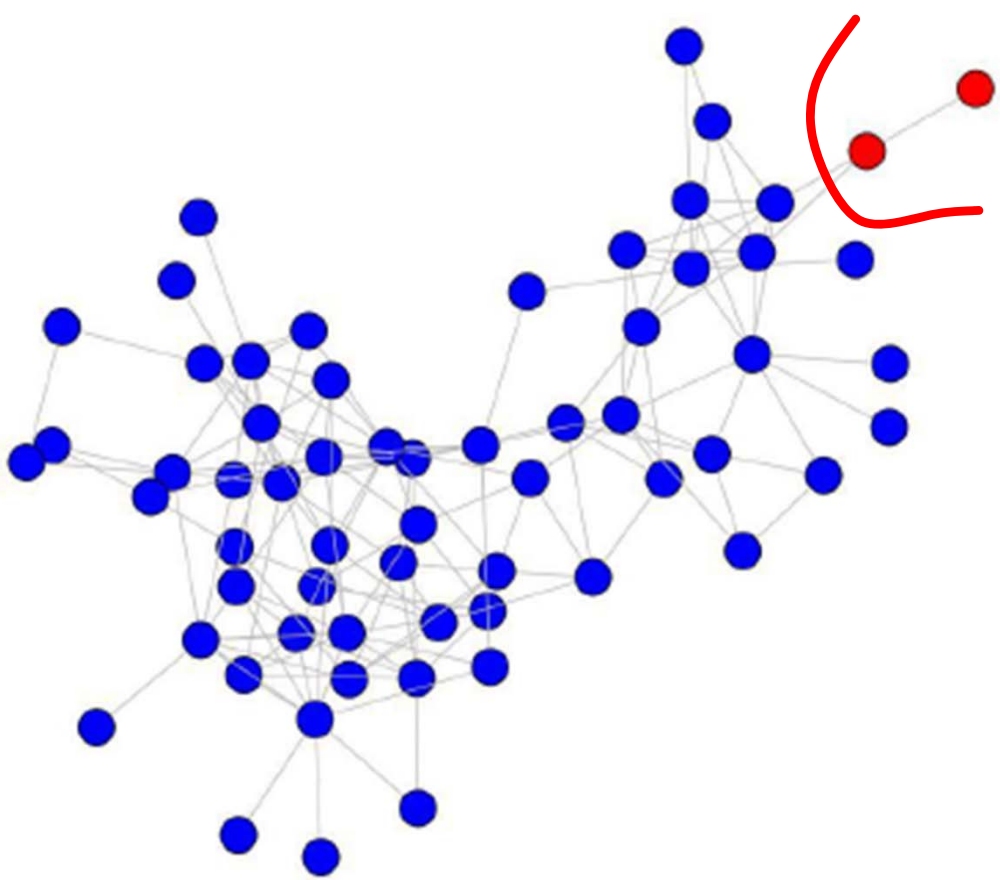
$$\phi(A) = \frac{|\{(i, j) \in E; i \in A, j \notin A\}|}{\min(vol(A), \ 2m - vol(A))}$$

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

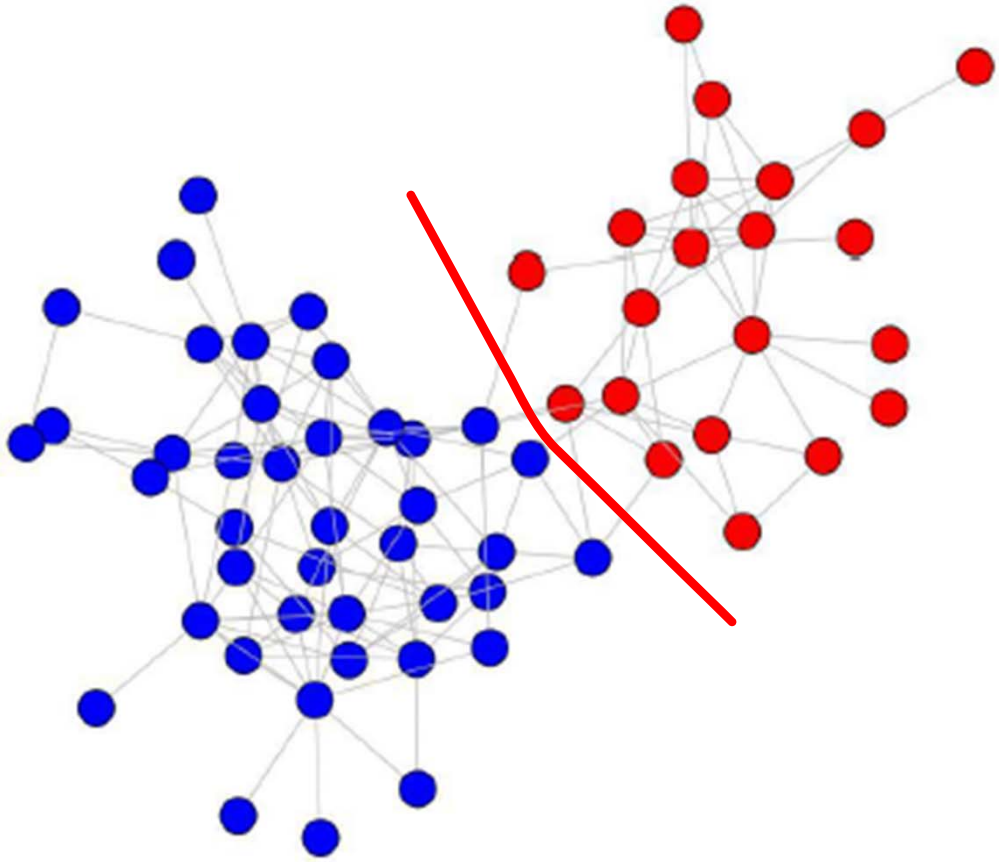
m… number of edges of the graph
$\mathbf{d_i}$… degree of node ***i***

- **Why use this criterion?**
  - Produces more balanced partitions

Example: Conductance Score



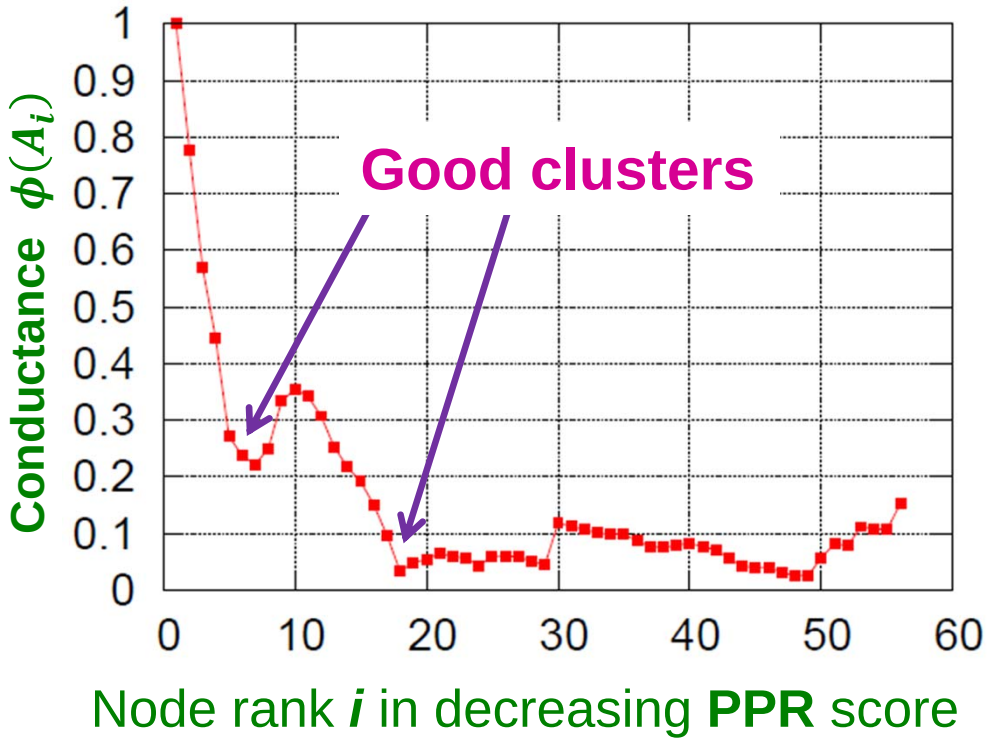
$\phi = 2/4 = 0.5$



$\phi = 6/92 = 0.065$

Algorithm Outline: Sweep

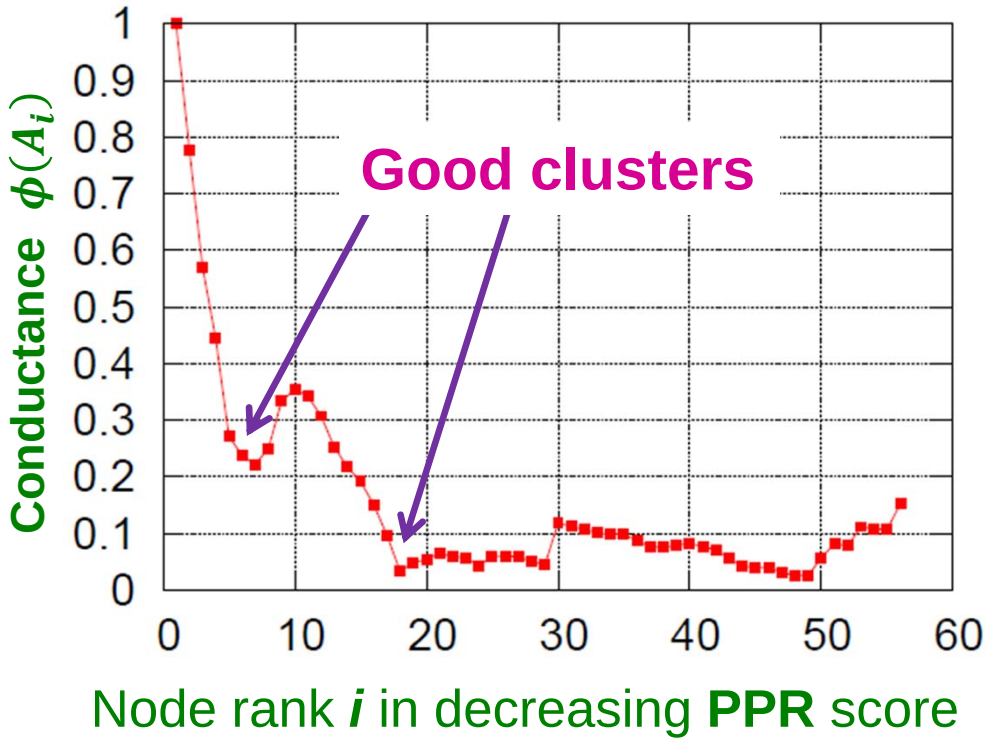
- **Algorithm outline:**
  - Pick a seed node ***s*** of interest
  - Run **PPR** w/ teleport={***s***}
  - Sort the nodes by the decreasing **PPR** score
  - **Sweep** over the nodes and find good clusters



- **Sweep:**
  - Sort nodes in decreasing PPR score $r_1 > r_2 > \cdots > r_n$
  - For each $\boldsymbol{i}$ compute $\boldsymbol{\phi(A_i = \{r_1, \ldots r_i\})}$
  - **Local minima** of $\boldsymbol{\phi(A_i)}$ correspond to good clusters

Computing the Sweep

- The whole Sweep curve can be computed in **linear time:**
  - For loop over the nodes
  - Keep hash-table of nodes in A_i
  - To compute $\phi(A_{i+1}) = Cut(A_{i+1})/Vol(A_{i+1})$
    - $Vol(A_{i+1}) = Vol(A_i) + d_{i+1}$
    - $Cut(A_{i+1}) = Cut(A_i) + d_{i+1} - 2\#edges\ of\ u_{i+1}\ to\ A_i$



Computing PPR

- **How to compute Personalized PageRank (PPR) without touching the whole graph?**
  - Power method won't work since each single iteration accesses all nodes of the graph:
    $\mathbf{r}^{(t+1)} = \beta \mathbf{M} \cdot \mathbf{r}^{(t)} + (1-\beta)\boldsymbol{a}$
    At index **s**
    - $\boldsymbol{a}$ is a teleport vector: $\boldsymbol{a} = [0 \ \ldots 0 \ 1 \ 0 \ \ldots 0]^T$
    - $\boldsymbol{r}$ is the personalized PageRank vector
- **PageRank-Nibble** [Andersen, Chung, Lang, '07]
  - A fast method for computing approximate Personalized PageRank (**PPR**) with teleport set ={**s**}
  - **ApproxPageRank(s, β, ε)**
    - **s** … seed node
    - **β** … teleportation parameter
    - **ε** … approximation error parameter

Approximate PPR: Overview

- **Overview of the approximate PPR**
  - **Lazy random walk**, which is a variant of a random walk that stays put with probability 1/2 at each time step, and walks to a random neighbor the other half of the time:

$$r_u^{(t+1)} = \frac{1}{2} r_u^{(t)} + \frac{1}{2} \sum_{i \to u} \frac{1}{d_i} r_i^{(t)}$$

d_i… degree of i

  - Keep track of **residual PPR score** $\boldsymbol{q_u = p_u - r_u^{(t)}}$
    - Residual tells us how well PPR score of $\boldsymbol{u}$ is approximated
    - $\boldsymbol{p_u}$… is the 'true' PageRank of node $\boldsymbol{u}$
    - $\boldsymbol{r_u^{(t)}}$… is PageRank estimate at around $\boldsymbol{t}$

    If **residual** $\boldsymbol{q_u}$ of node $\boldsymbol{u}$ is too big $\frac{\boldsymbol{q_u}}{\boldsymbol{d_u}} \geq \boldsymbol{\varepsilon}$ then *push the walk further (***for each** $\boldsymbol{v}$ **such that** $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{E}$: $\boldsymbol{q'_v = q_v + \frac{1}{2}\beta \frac{q_u}{d_u}}$*)*, else don't touch the node

Towards approximate PPR

- **A different way to look at PageRank:**

  [Jeh&Widom. *Scaling Personalized Web Search*, 2002]

$$\boldsymbol{p_\beta(a) = (1-\beta)a + \beta\, p_\beta(Ma)}$$

  - $\boldsymbol{p}$ is the true PageRank vector with teleport parameter $\boldsymbol{\beta}$, and teleport vector $\boldsymbol{a}$.
  - $\boldsymbol{p_\beta(Ma)}$ is the PageRank vector with teleportation vector (set) $\boldsymbol{Ma}$ and teleportation parameter $\boldsymbol{\beta}$
    - And $\boldsymbol{M}$ is the stochastic PageRank transition matrix

- **Gives an idea how to compute PageRank:**
  - Node $\boldsymbol{u}$'s "view" of the graph ($\boldsymbol{p_u}$) is the average of its out-neighbors views plus $\boldsymbol{u}$'s own importance

"Push" Operation

- **Idea:**
  - $\boldsymbol{r}$… approx. PageRank, $\boldsymbol{q}$… its residual PageRank
  - Start with trivial approximation: $\boldsymbol{r} = \boldsymbol{0}$ and $\boldsymbol{q} = \boldsymbol{a}$
  - Iteratively **push** PageRank from $\boldsymbol{q}$ to $\boldsymbol{r}$ until $\boldsymbol{q}$ is small
  - **Maintain invariant:** $\boldsymbol{r = p_\beta(a - q)}$

    [Andersen, Chung, Lang. *Local graph partitioning using PageRank vectors*, 2007]

- **Push: 1 step of a lazy random walk from node $\boldsymbol{u}$:**

$\boldsymbol{Push(u, r, q)}$:

$\boldsymbol{r' = r,\ q' = q}$

$\boldsymbol{r'_u = r_u + (1 - \beta) q_u}$

$\boldsymbol{q'_u = \frac{1}{2}\beta q_u}$

for each $\boldsymbol{v}$ **such that** $\boldsymbol{(u, v) \in E}$:

$\boldsymbol{q'_v = q_v + \frac{1}{2}\beta \frac{q_u}{d_u}}$

return $\boldsymbol{r',\ q'}$

Update **r**
Do 1 step of a walk:
Stay at **u** with prob. ½
Spread remaining ½ fraction of $\mathbf{q_u}$ as if a single step of random walk were applied to **u**

Intuition Behind Push Operation

- If q_u is large, this means that we have underestimated the importance of node u

$Push(u, r, q)$:
$r' = r,\ q' = q$
$r'_u = r_u + (1 - \beta) q_u$
$q'_u = \frac{1}{2} \beta q_u$
for each v **such that** $(u, v) \in E$:
$q'_v = q_v + \frac{1}{2} \beta \frac{q_u}{d_u}$
return $r',\ q'$

- Then we want take some of that 'gap' (q_u) and give it away, since we know that we have too much of it
- So, we keep $\frac{1}{2} \beta q_u$ and then give away the rest to our neighbors, so that we can get rid of it
  - This correspond to the spreading of $\frac{1}{2} \beta\ q_u / d_u$ term
- Each node wants to keep giving away this excess PageRank until we have settled close to the all nodes have no or a very small gap

Approximate PPR

- **ApproxPageRank(S, β, ε):**

  Set $\boldsymbol{r} = \vec{0}, \boldsymbol{q} = [0 \,..\, 0\ 1\ 0 \,...\, 0]$ ← At index **S**

  While $\max_{u \in V} \frac{q_u}{d_u} \geq \varepsilon$:

    Choose any vertex $\boldsymbol{u}$ where $\frac{q_u}{d_u} \geq \varepsilon$

    $\boldsymbol{Push(u, r, q)}$:

      $\boldsymbol{r' = r,\ q' = q}$

      $\boldsymbol{r'_u = r_u + (1 - \beta) q_u}$

      $\boldsymbol{q'_u = \frac{1}{2} \beta q_u}$

      For each v such that $(u, v) \in E$:

        $\boldsymbol{q'_v = q_v + \frac{1}{2} \beta q_u / d_u}$

      Update $r = r',\ q = q'$

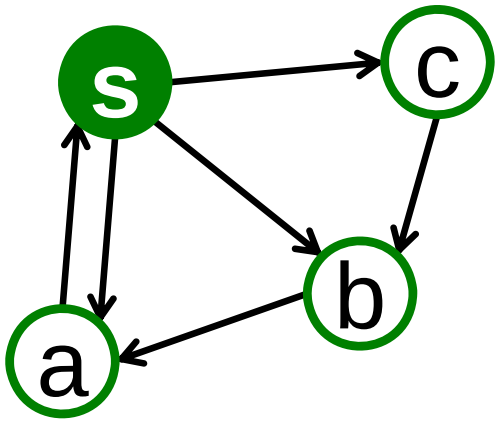
  Return r

r … **PPR** vector
$\mathbf{r_u}$ …**PPR** score of **u**
q …residual **PPR** vector
$\mathbf{q_u}$ … residual of node **u**
$\mathbf{d_u}$ … degree of **u**

Update **r:** Move $(1 - \beta)$ of the prob. from $\mathbf{q_u}$ to $\mathbf{r_u}$

1 step of a lazy random walk:
- Stay at $\mathbf{q_u}$ with prob. ½
- Spread remaining ½ $\boldsymbol{\beta}$ fraction of $\mathbf{q_u}$ as if a single step of random walk were applied to **u**

Example: $\beta = 0.5$



ApproxPageRank(S, β, ε):

Set $\boldsymbol{r} = \vec{0}$, $\boldsymbol{q} = [0\,..\,0\;1\;0\;...\;0]$

while $\max_{u \in V} \frac{q_u}{d_u} \geq \varepsilon$:

Choose any vertex $\boldsymbol{u}$ where $\frac{q_u}{d_u} \geq \varepsilon$

$\boldsymbol{Push(u, r, q)}$:

$\boldsymbol{r' = r,\ q' = q}$

$\boldsymbol{r'_u = r_u + (1 - \beta) q_u}$

$\boldsymbol{q'_u = \frac{1}{2} \beta q_u}$

For each v such that $(u, v) \in E$:

$\boldsymbol{q'_v = q_v + \frac{1}{2} \beta \frac{q_u}{d_u}}$

Update $\boldsymbol{r = r',\ q = q'}$

return $\boldsymbol{r}$

s a b c

Init:

r = [0, 0, 0, 0]

q = [1, 0, 0, 0]

Push(s, r, q):

r = [.5, 0, 0, 0]

q = [.25, .08, .08, .08]

Push(s, r, q):

r = [.62, 0, 0, 0]

q = [.06, .10, .10, .10]

Push(a, r, q):

r = [0.62, .05, 0, 0]

q = [.09, .03, .10, .10]

Push(b, r, q):

r = [0.62, .05, .05, 0]

q = [.09, .05, .03, .10]

….

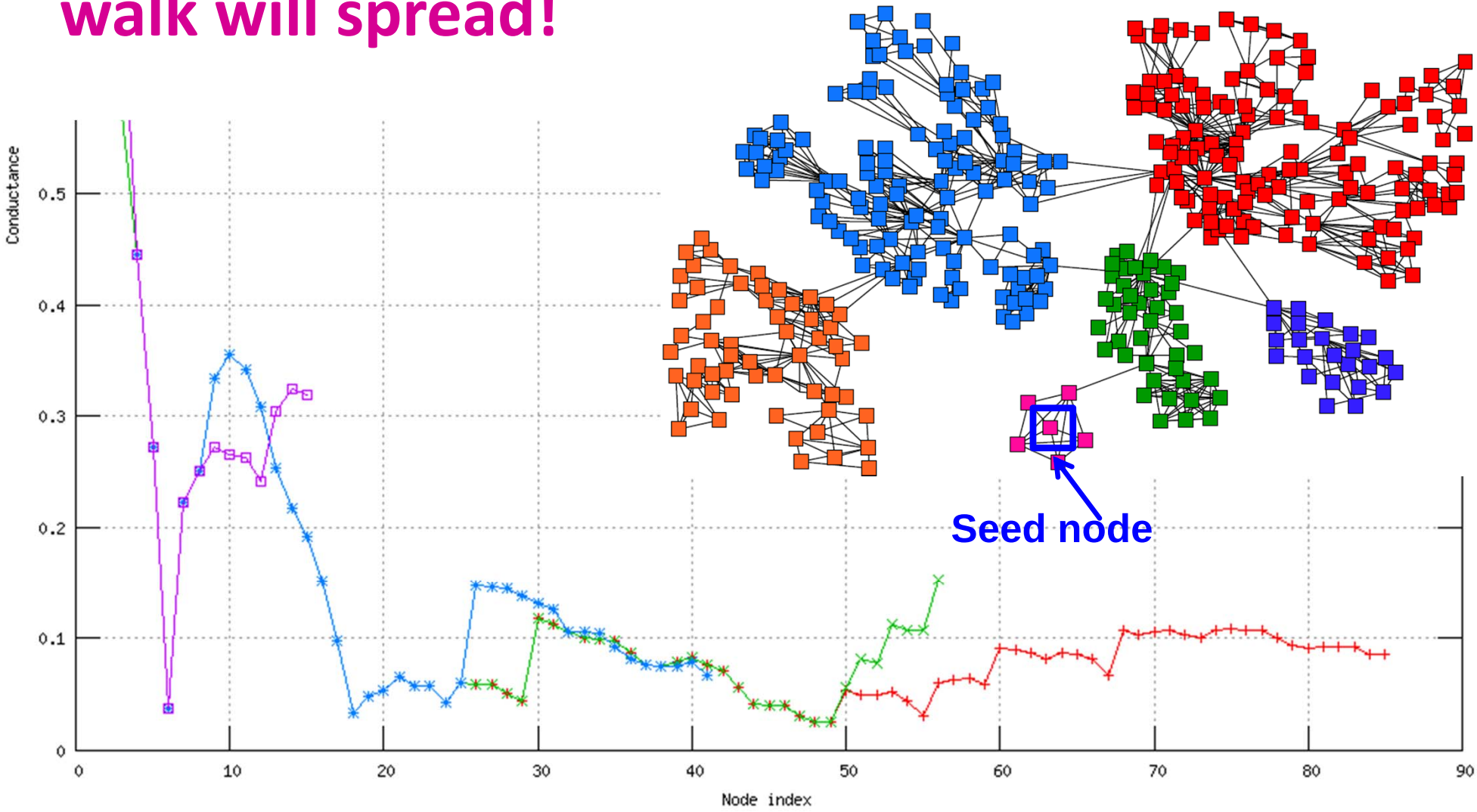
r = [.57, .19, .14, .09]

Observations (1)

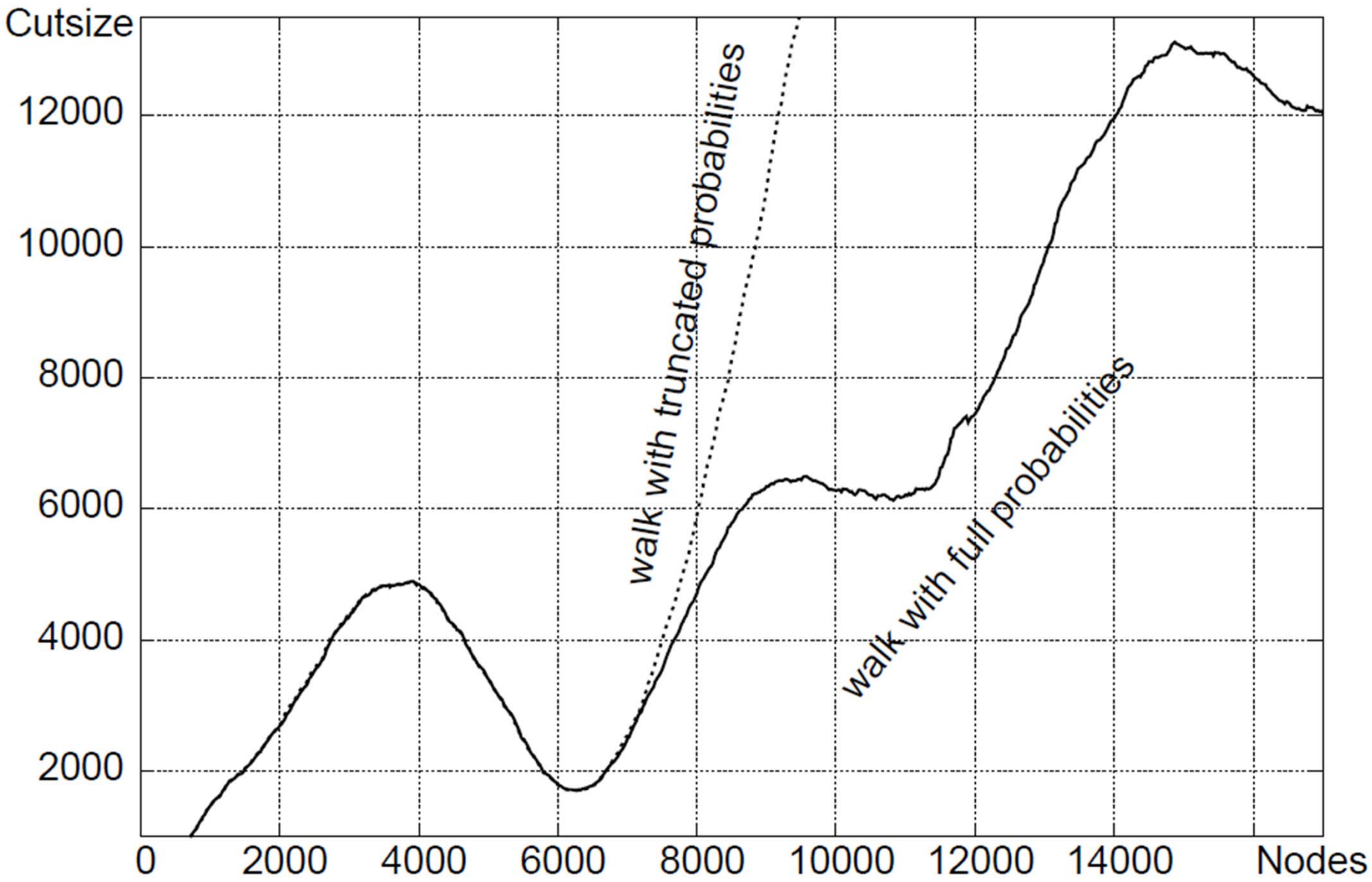
- **Runtime:**
  - PageRank-Nibble computes PPR in time $\left(\frac{1}{\varepsilon(1-\beta)}\right)$ with residual error $\leq \varepsilon$
    - Power method would take time $O(\frac{\log n}{\varepsilon(1-\beta)})$
- **Graph cut approximation guarantee:**
  - If there exists a cut of conductance ϕ then the method finds a cut of conductance $\Omega(\phi^2 / \log m)$
  - Details in [Andersen, Chung, Lang. *Local graph partitioning using PageRank vectors*, 2007]
    http://www.math.ucsd.edu/~fan/wp/localpartfull.pdf

Observations (2)

- **The smaller the ε the farther the random walk will spread!**

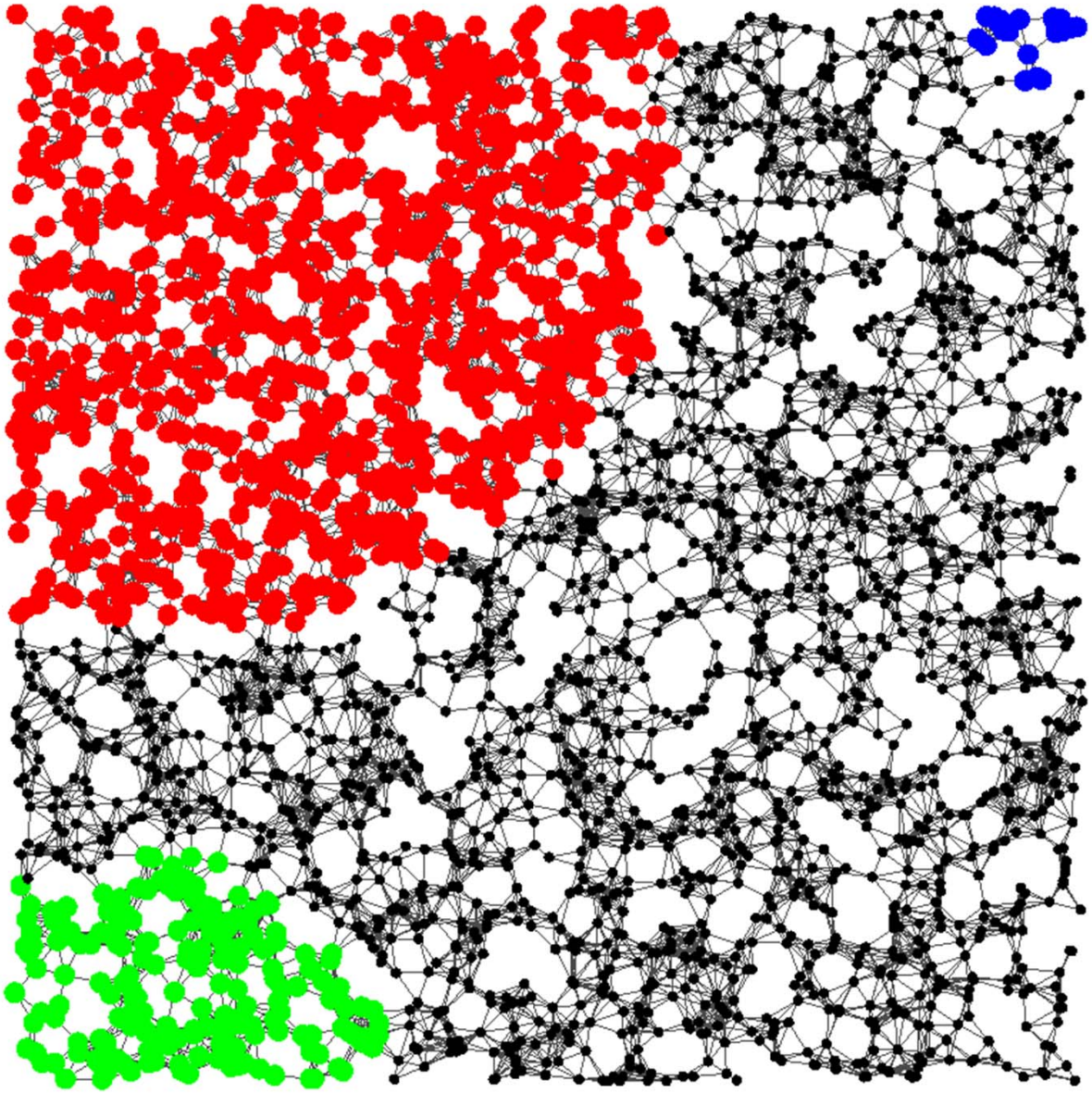


Observations (3)

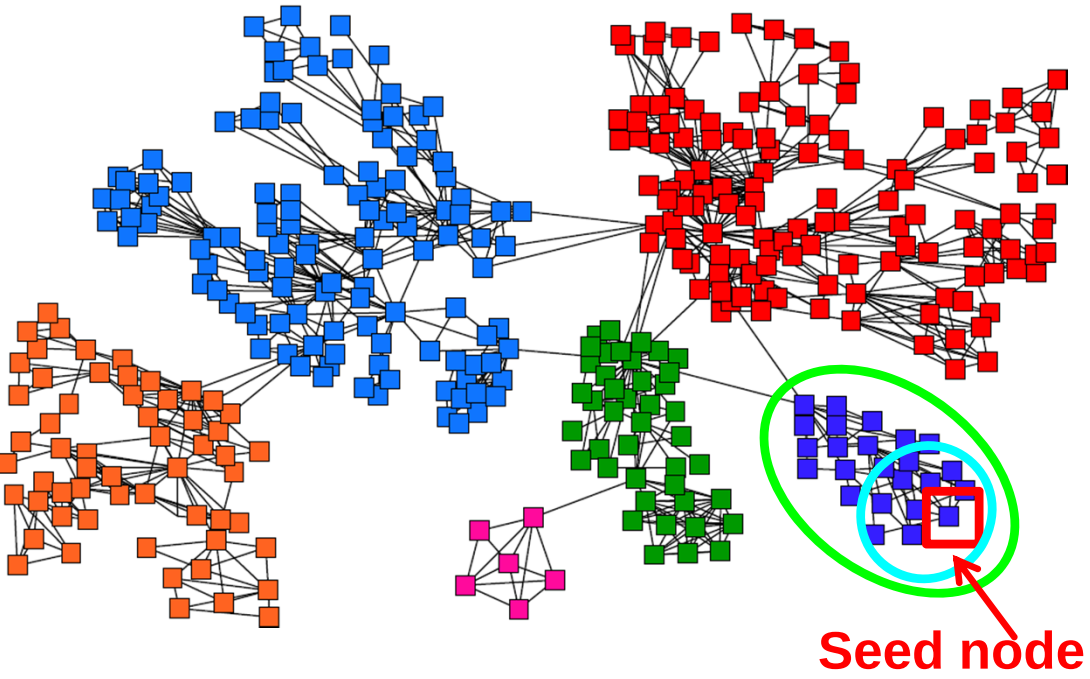


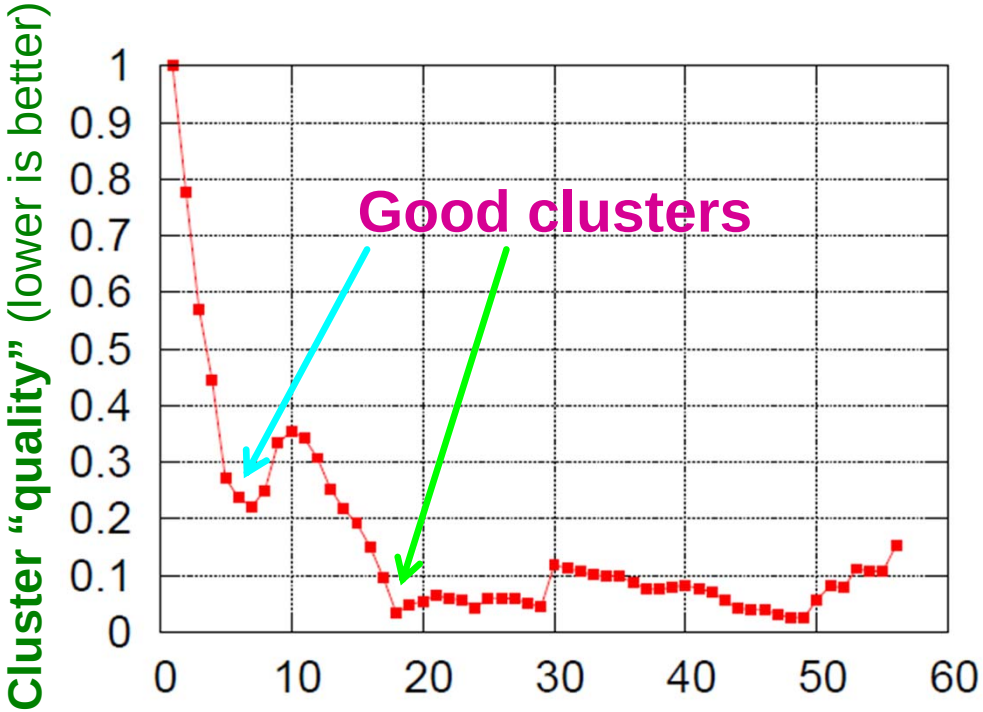
[Andersen, Lang: Communities from seed sets, 2006]

Example



Summary





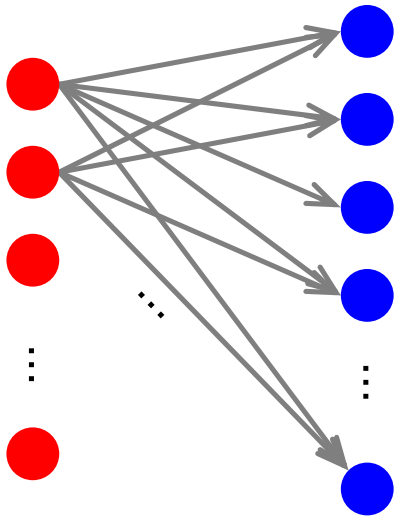
- **Algorithm summary:**
  - Pick a seed node ***s*** of interest
  - Run **PPR** with teleport set = {***s***}
  - Sort the nodes by the decreasing **PPR** score
  - **Sweep** over the nodes and find good clusters

Analysis of Large Graphs: Trawling

[Kumar et al. '99]

Trawling

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



Dense 2-layer graph

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

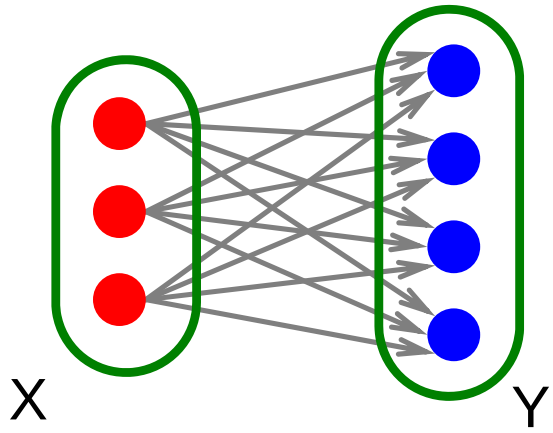
Intuition: Many people all talking about the same things

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"



$K_{3,4}$

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

Fully connected

[Agrawal-Srikant '99]

Frequent Itemset Enumeration

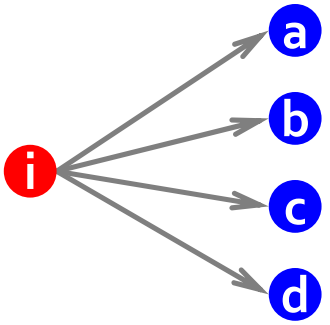
- **Market basket analysis.** Setting:
  - **Market:** Universe U of n items
  - **Baskets:** m subsets of U: $S_1, S_2, \ldots, S_m \subseteq U$ (S_i is a set of items one person bought)
  - **Support:** Frequency threshold f
- **Goal:**
  - **Find all subsets T s.t. $T \subseteq S_i$ of at least f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

[Kumar et al. '99]

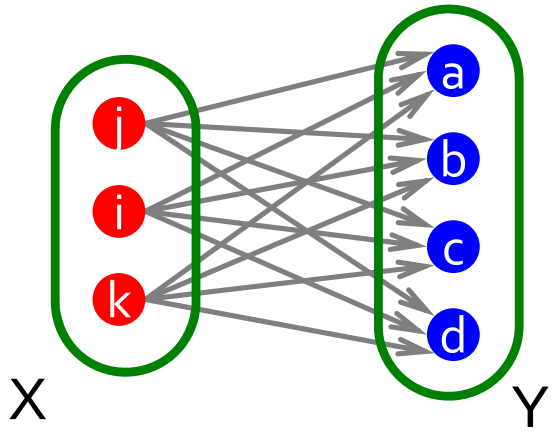
From Itemsets to Bipartite $K_{s,t}$

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 that occurs in s 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}

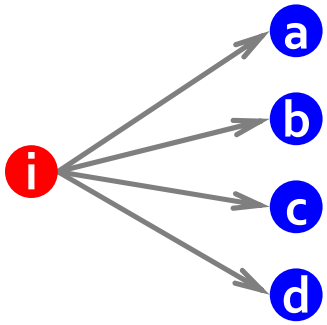


s … minimum support (|X|=s)
t … itemset size (|Y|=t)

[Kumar et al. '99]

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

View each node ***i*** as a set S_i of nodes ***i*** points to



S_i={a,b,c,d}

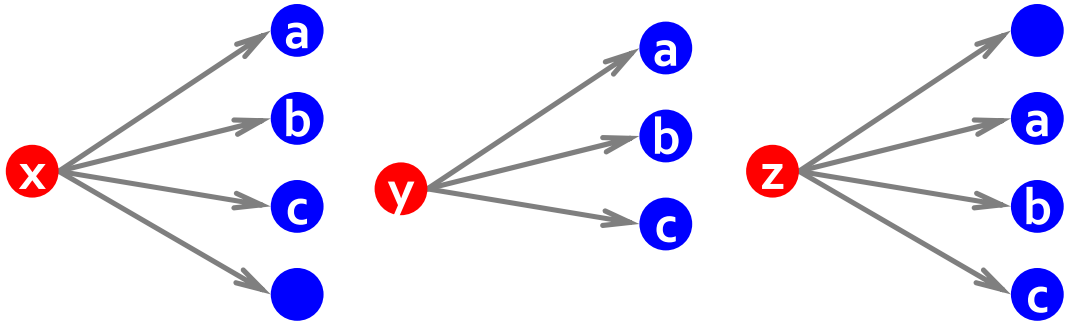
Find frequent itemsets:

s … minimum support

t … itemset size

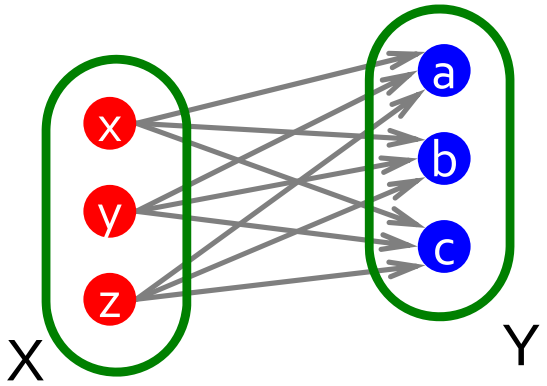
Say we find a **frequent itemset** ***Y={a,b,c}*** of supp ***s***

So, there are ***s*** nodes that link to all of **{a,b,c}**:

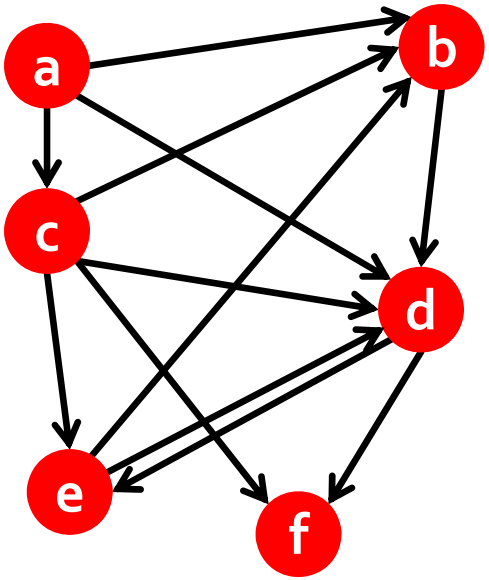


We found $K_{s,t}$!

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



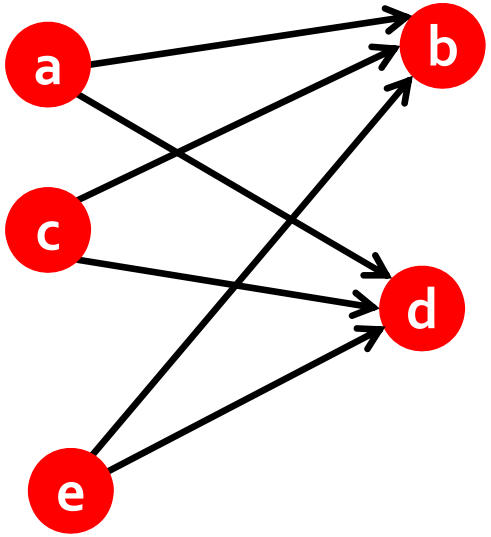
Example (1)

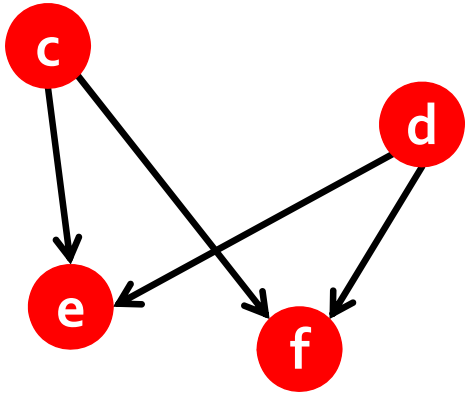


- **Support threshold s=2**
  - **{b,d}**: support 3
  - **{e,f}**: support 2
- **And we just found 2 bipartite subgraphs:**

Itemsets:

a = {b,c,d}
b = {d}
c = {b,d,e,f}
d = {e,f}
e = {b,d}
f = {}





Example (2)

- **Example of a community from a web graph**

A community of Australian fire brigades

Nodes on the right	Nodes on the left
NSW Rural Fire Service Internet Site	New South Wales Fir...ial Australian Links
NSW Fire Brigades	Feuerwehrlinks Australien
Sutherland Rural Fire Service	FireNet Information Network
CFA: County Fire Authority	The Cherrybrook Rur...re Brigade Home Page
"The National Cente...ted Children's Ho...	New South Wales Fir...ial Australian Links
CRAFTI Internet Connexions-INFO	Fire Departments, F... Information Network
Welcome to Blackwoo... Fire Safety Serv...	The Australian Firefighter Page
The World Famous Guestbook Server	Kristiansand brannv...dens brannvesener...
Wilberforce County Fire Brigade	Australian Fire Services Links
NEW SOUTH WALES FIR...ES 377 STATION	The 911 F,P,M., Fir...mp; Canada A Section
Woronora Bushfire Brigade	Feuerwehrlinks Australien
Mongarlowe Bush Fire – Home Page	Sanctuary Point Rural Fire Brigade
Golden Square Fire Brigade	Fire Trails "l...ghters around the...
FIREBREAK Home Page	FireSafe – Fire and Safety Directory
Guises Creek Volunt...fficial Home Page...	Kristiansand Firede...departments of th...

[Kumar, Raghavan, Rajagopalan, Tomkins: Trawling the Web for emerging cyber-communities 1999]