# Outbreak Detection in Networks

CS 322: (Social and Information) Network Analysis

Jure Leskovec
Stanford University

# Announcements

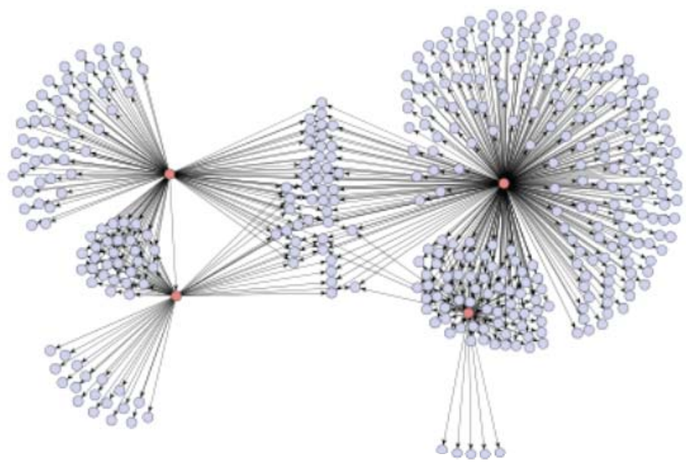
Thursday: guest lecture

Lars Backstrom (Facebook/Cornell)

on networks & geography

# Finding influencers

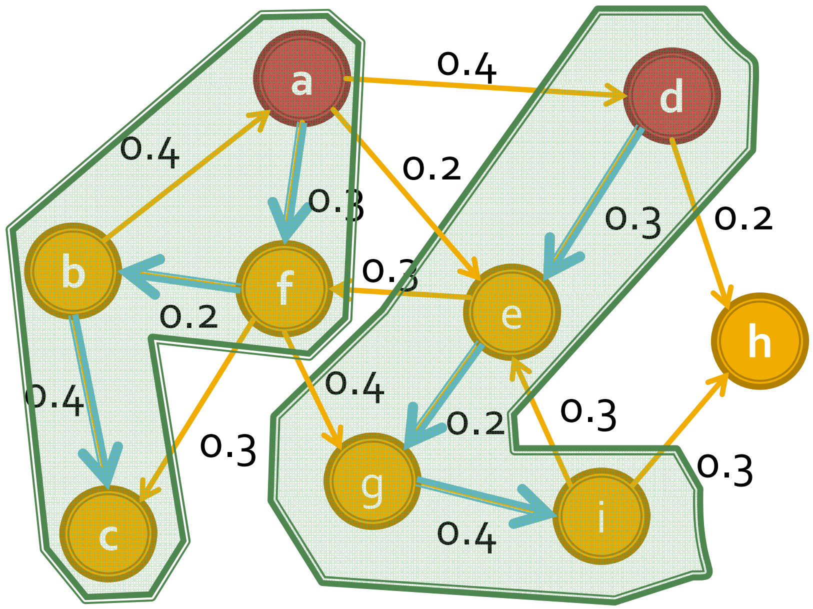
- Blogs – information epidemics
  - Which are the influential/infectious blogs?
- Viral marketing
  - Who are the trendsetters?
  - Influential people?



- Disease spreading
  - Where to place monitoring stations to detect epidemics?

# Most Influential Subset of Nodes

- Most influential set of size k: set S of k nodes producing largest expected cascade size f(S) if activated [Domingos-Richardson '01]



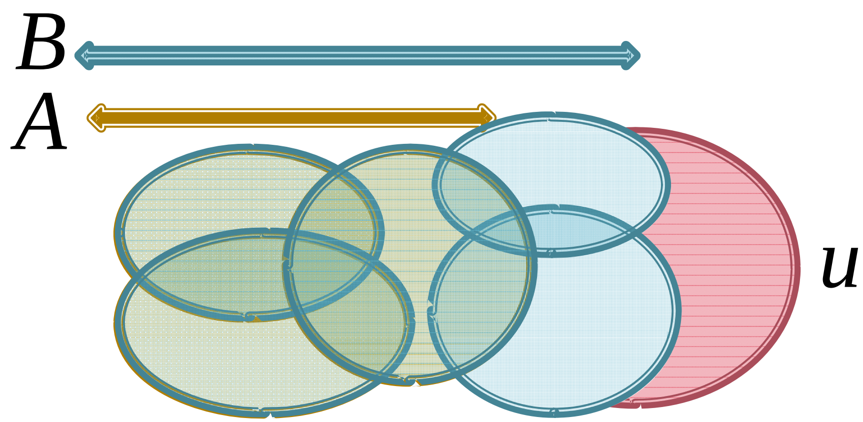

- Optimization problem: $\max_{\text{S of size k}} f(S)$

# Problem structure: Submodularity

- f is submodular: $\boldsymbol{S \subseteq T}$

$$\underbrace{f(S \cup \{u\}) - f(S)}_{\text{Gain of adding a node to a small set}} \geq \underbrace{f(T \cup \{u\}) - f(T)}_{\text{Gain of adding a node to a large set}}$$

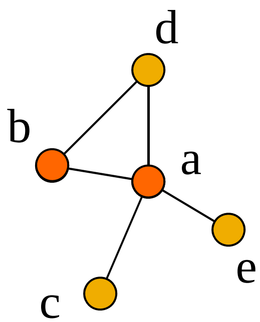
- Natural example:
  - Sets $A_1, A_2, \ldots, A_n$
  - f(A) = size of union of $A_i$ (size of covered area)

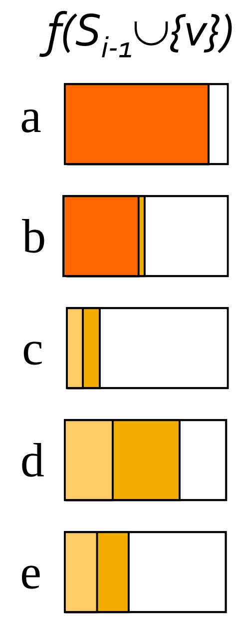



- If $f_1, \ldots, f_K$ are submodular, then $\sum p_i f_i$ is submodular

# Hill climbing

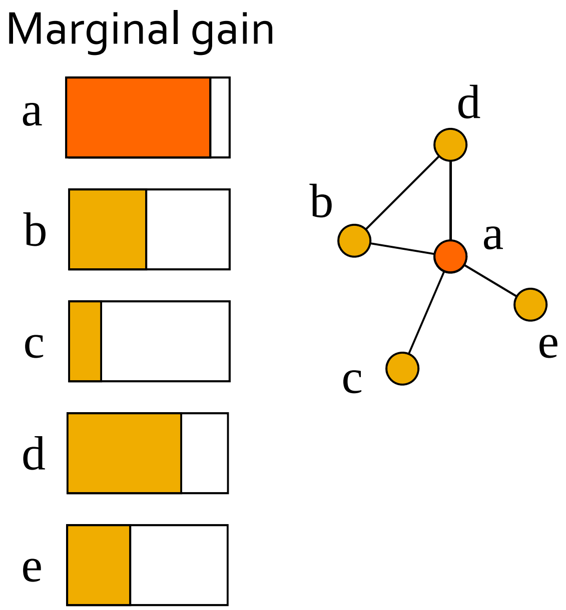
- Start with $S_0=\{\}$
- For i=1…k
  - Choose node *v* that max $f(S_{i-1} \cup \{v\})$
  - Let $S_i = S_{i-1} \cup \{v\}$
- Hill climbing produces a solution S where f(S) ≥(1-1/e) of optimal value (~63%) when f is monotone and submodular [Hemhauser, Fisher, Wolsey '78]



# Lazy evaluation

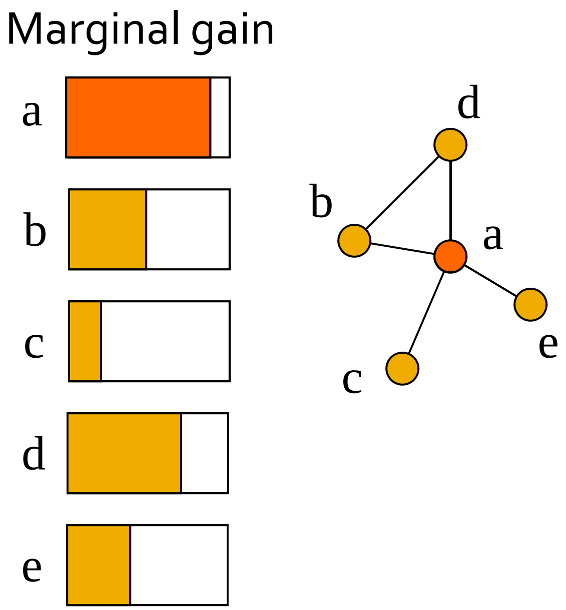
- Lazy hill-climbing:
  - Keep an ordered list of marginal benefits $b_i$ from previous iteration
  - Re-evaluate $b_i$ only for top node
  - Re-sort and prune



[Leskovec et al., KDD '07]

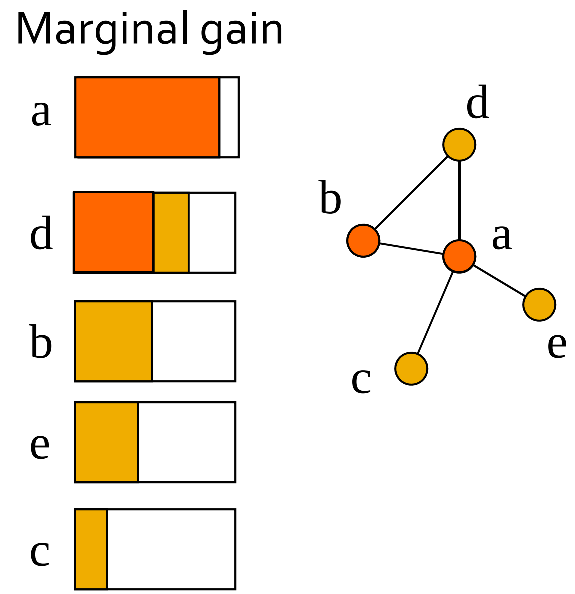
# Lazy evaluation

- Lazy hill-climbing:
  - Keep an ordered list of marginal benefits $b_i$ from previous iteration
  - Re-evaluate $b_i$ only for top node
  - Re-sort and prune



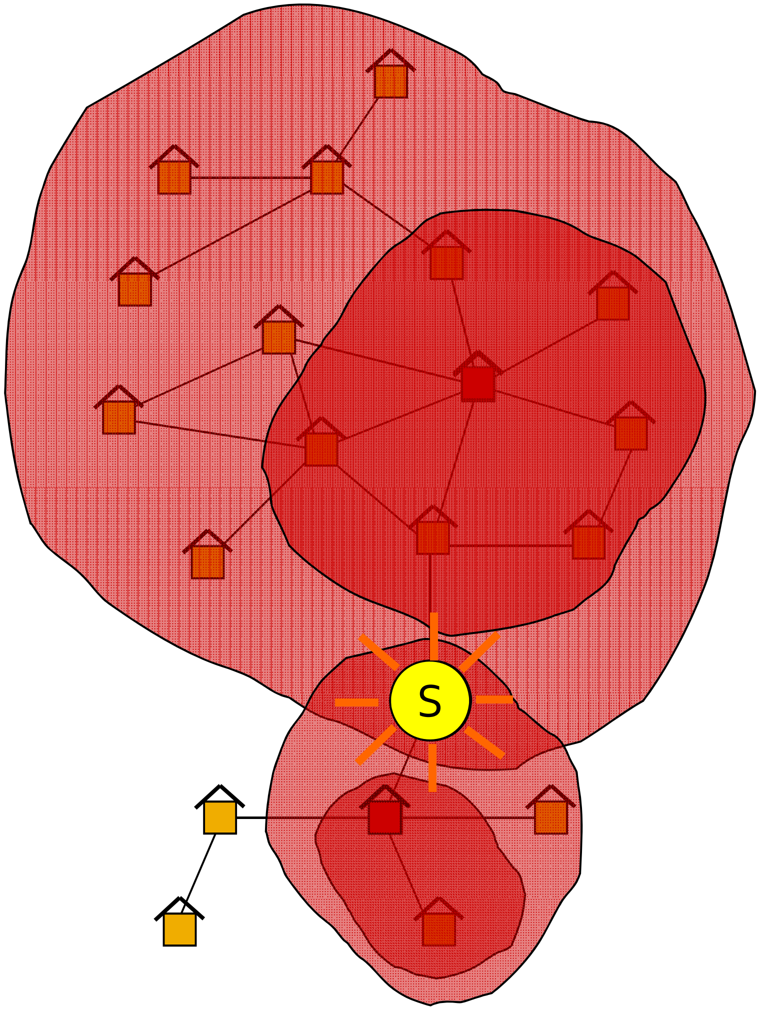
# Lazy evaluation

- Lazy hill-climbing:
  - Keep an ordered list of marginal benefits $b_i$ from previous iteration
  - Re-evaluate $b_i$ only for top node
  - Re-sort and prune



# Problem: Water Network

- Given a real city water distribution network
- And data on how contaminants spread in the network
- Problem posed by *US Environmental Protection Agency*



[Leskovec et al., KDD '07]

# Problem Setting

- Given a graph $G(V,E)$
- and a budget $B$ for sensors
- and data on how contaminations spread over the network:
  - for each contamination $i$ we know the time $T(i,u)$ when it contaminated node $u$
- Select a subset of nodes $A$ that maximize the expected reward

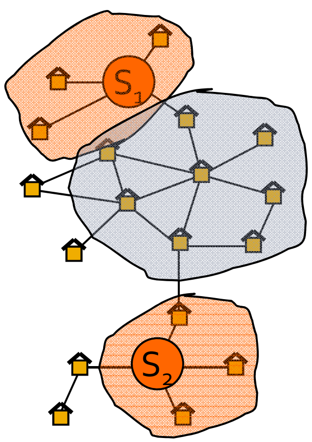
$$\max_{\mathcal{A} \subseteq \mathcal{V}} R(\mathcal{A}) \equiv \sum_i P(i) \underbrace{R_i(T(i,\mathcal{A}))}_{\text{Reward for detecting contamination } i}$$

Reward for detecting contamination $i$

subject to $cost(A) < B$

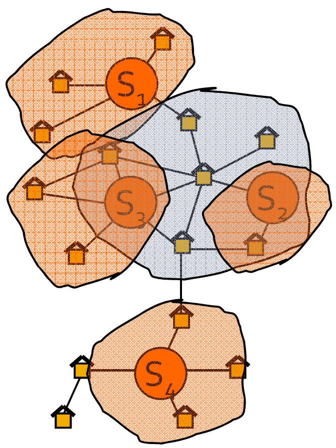
# Structure of the Problem

- Observation: Diminishing returns




Placement A={$S_1$, $S_2$}

New sensor:


Placement B={$S_1$, $S_2$, $S_3$, $S_4$}

Adding S' helps a lot

Adding S' helps very little

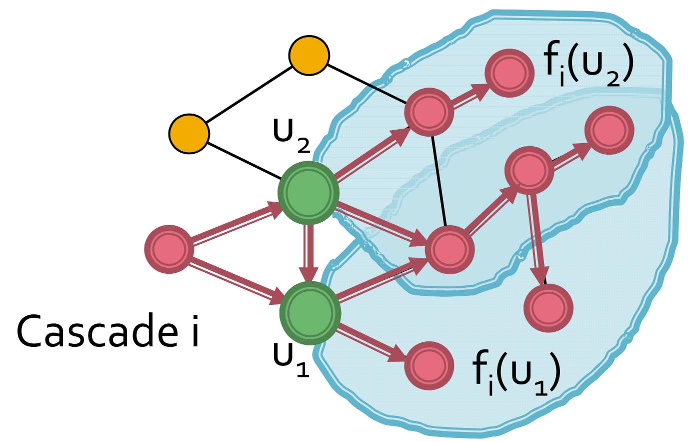
[Leskovec et al., KDD '07]

# Reward function is submodular

- <u>Claim:</u>
  - Reward function is submodular
- Consider cascade *i*:
  - $f_i(u_k)$ = set of nodes saved from $u_k$
  - $f_i(A)$ = size of union $f_i(u_k)$, $u_k \in A$

  $\Rightarrow f_i$ is submodular
- Global optimization:
  - $f(A) = \sum \text{Prob}(i)\, f_i(A)$

  $\Rightarrow$ f is submodular



[Leskovec et al., KDD '07]

# Towards a New Algorithm

- Consider: hill-climbing ignoring the cost
  - Ignore sensor cost. Repeatedly select sensor with highest marginal gain
- It always prefers more expensive sensor with reward *r* to a cheaper sensor with reward *r-ε*
  → For variable cost it can fail arbitrarily badly

- **Idea:** What if we optimize benefit-cost ratio?

$$s_k = \underset{s \in \mathcal{V} \setminus \mathcal{A}_{k-1}}{\operatorname{argmax}} \frac{R(\mathcal{A}_{k-1} \cup \{s\}) - R(\mathcal{A}_{k-1})}{c(s)}$$

[Leskovec et al., KDD '07]

# Benefit-Cost: More Problems

- Benefit-cost ratio can fail arbitrarily badly
- Consider: budget $B$:
  - 2 locations $s_1$ and $s_2$:
    - Costs: $c(s_1)=\varepsilon$, $c(s_2)=B$
    - Only 1 cascade: $R(s_1)=2\varepsilon$, $R(s_2)=B$
  - Then benefit-cost ratio is
    - $bc(s_1)=2$ and $bc(s_2)=1$
  - So, we first select $s_1$ and then can not afford $s_2$

→We get reward $2\varepsilon$ instead of $B$

Now send $\varepsilon$ to $0$ and we get arbitrarily bad

# Solution: CELF Algorithm

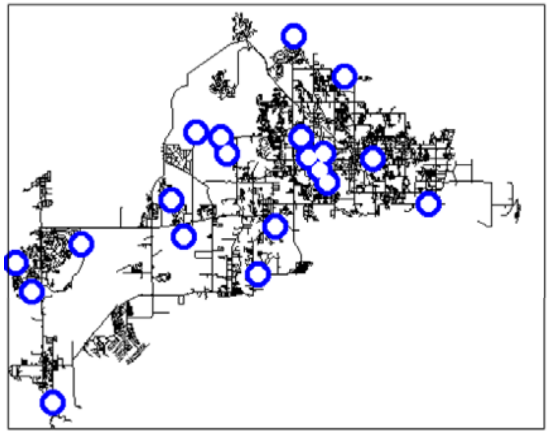
- CELF (cost-effective lazy forward-selection): A two pass greedy algorithm:
  - Set (solution) A: use benefit-cost greedy
  - Set (solution) B: use unit cost greedy
  - Final solution: *argmax(R(A), R(B))*

- Theorem: CELF is near optimal
  - CELF achieves $\frac{1}{2}(1-1/e)$ factor approximation

[Leskovec et al., KDD '07]

# Case study: Water Network

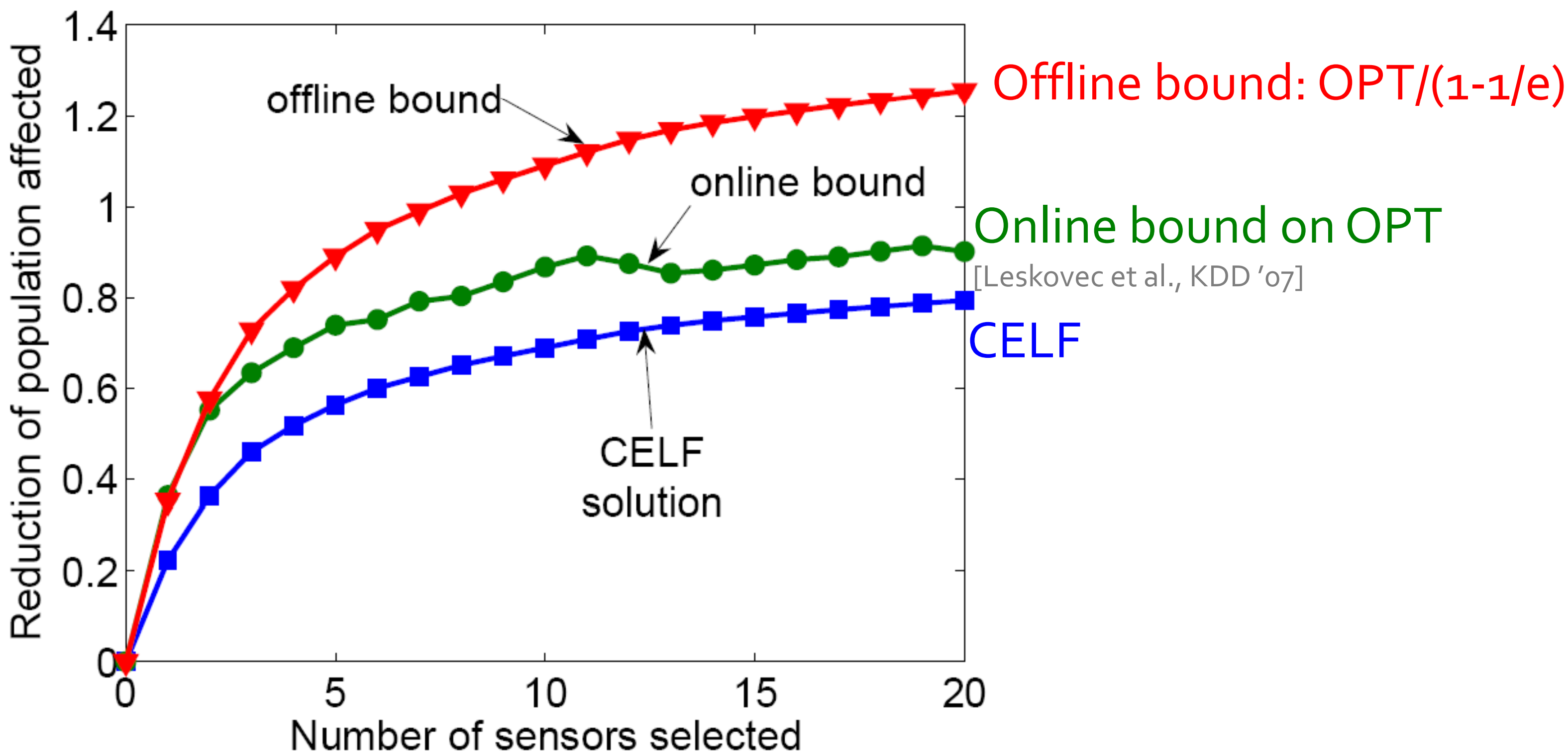
- Real metropolitan area water network
  - V = 21,000 nodes
  - E = 25,000 pipes



- Use a cluster of 50 machines for a month
- Simulate 3.6 million epidemic scenarios (152 GB of epidemic data)
- By exploiting sparsity we fit it into main memory (16GB)

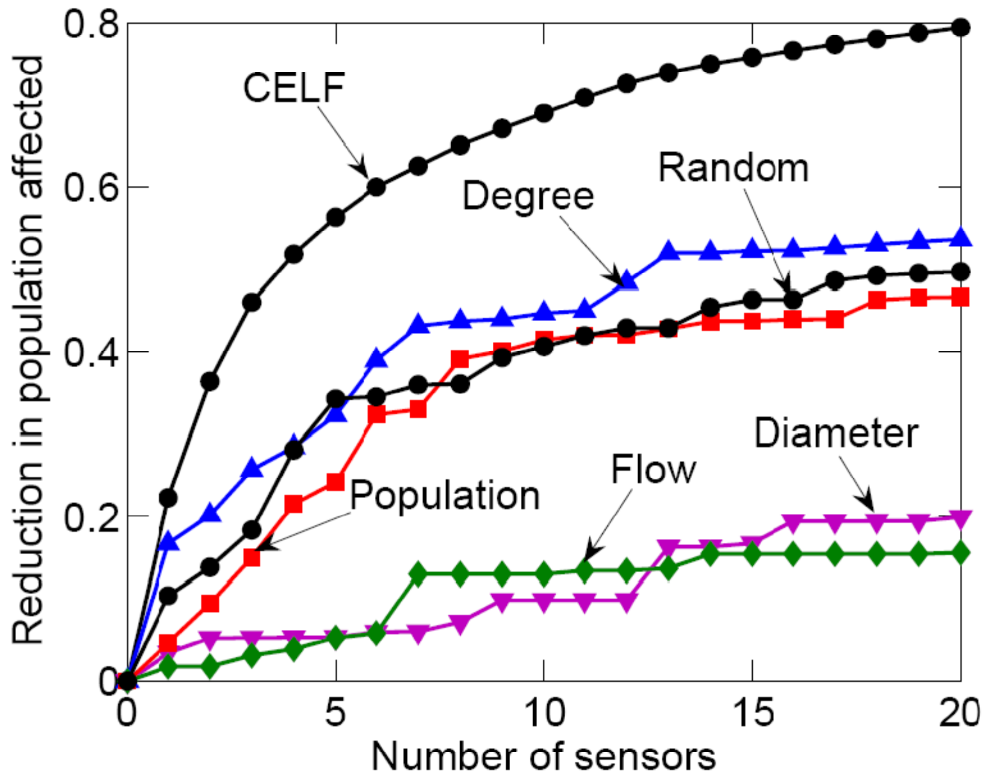
[Leskovec et al., KDD '07]

# Water: Solution Quality




- The online (data dependent) bound gives much better estimate of how far from unknown optimal solution is the CELF solution

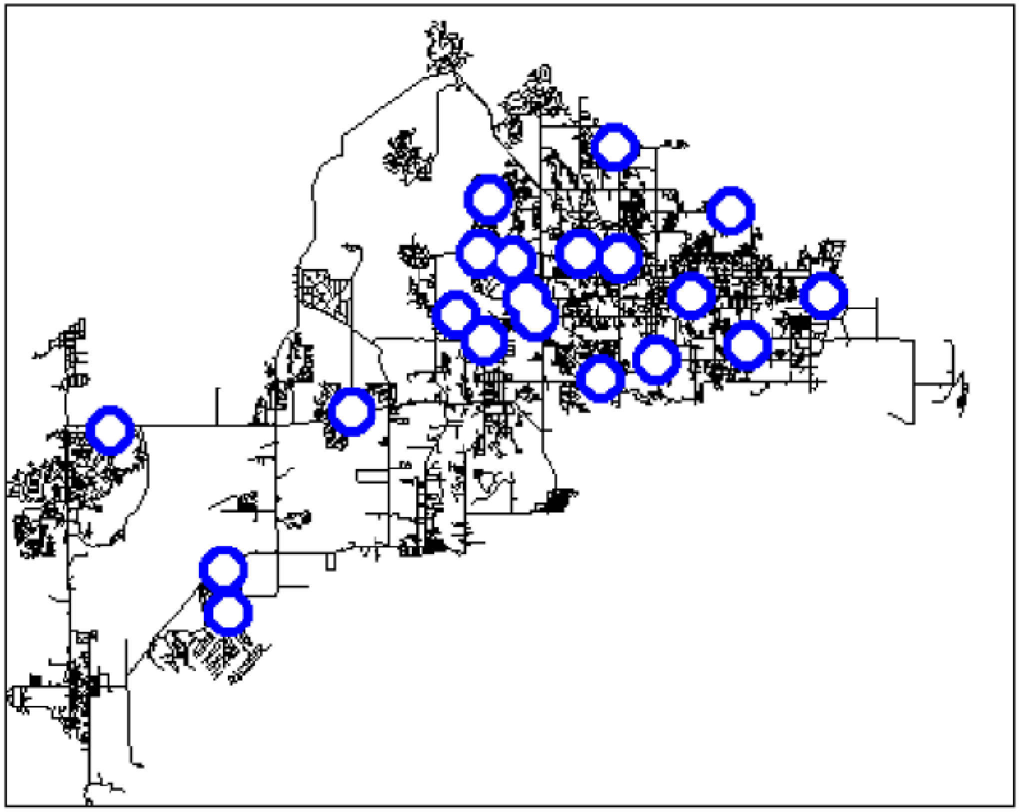
# Water: Heuristic Placement




- Heuristics placements perform much worse
- One really needs to consider the spread of epidemics

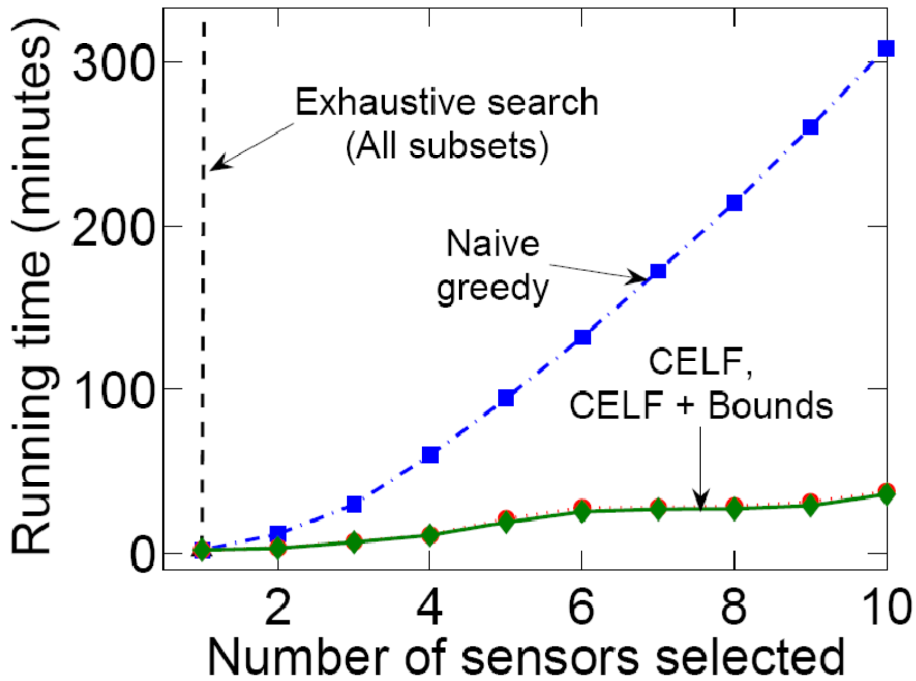
[Leskovec et al., KDD '07]

# Water: Placement Visualization



[Leskovec et al., KDD '07]

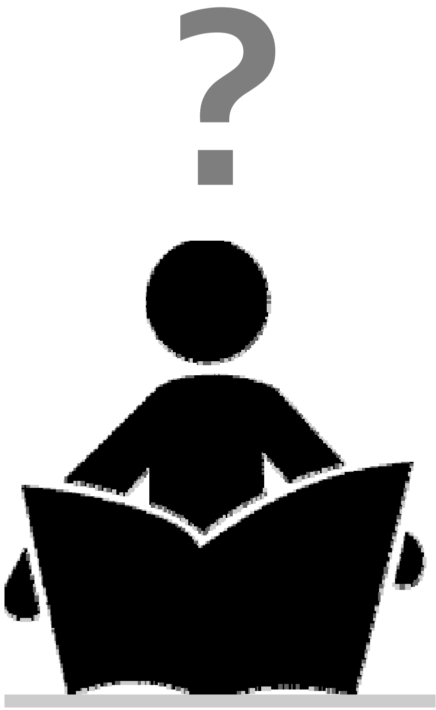
# Water: Algorithm Scalability



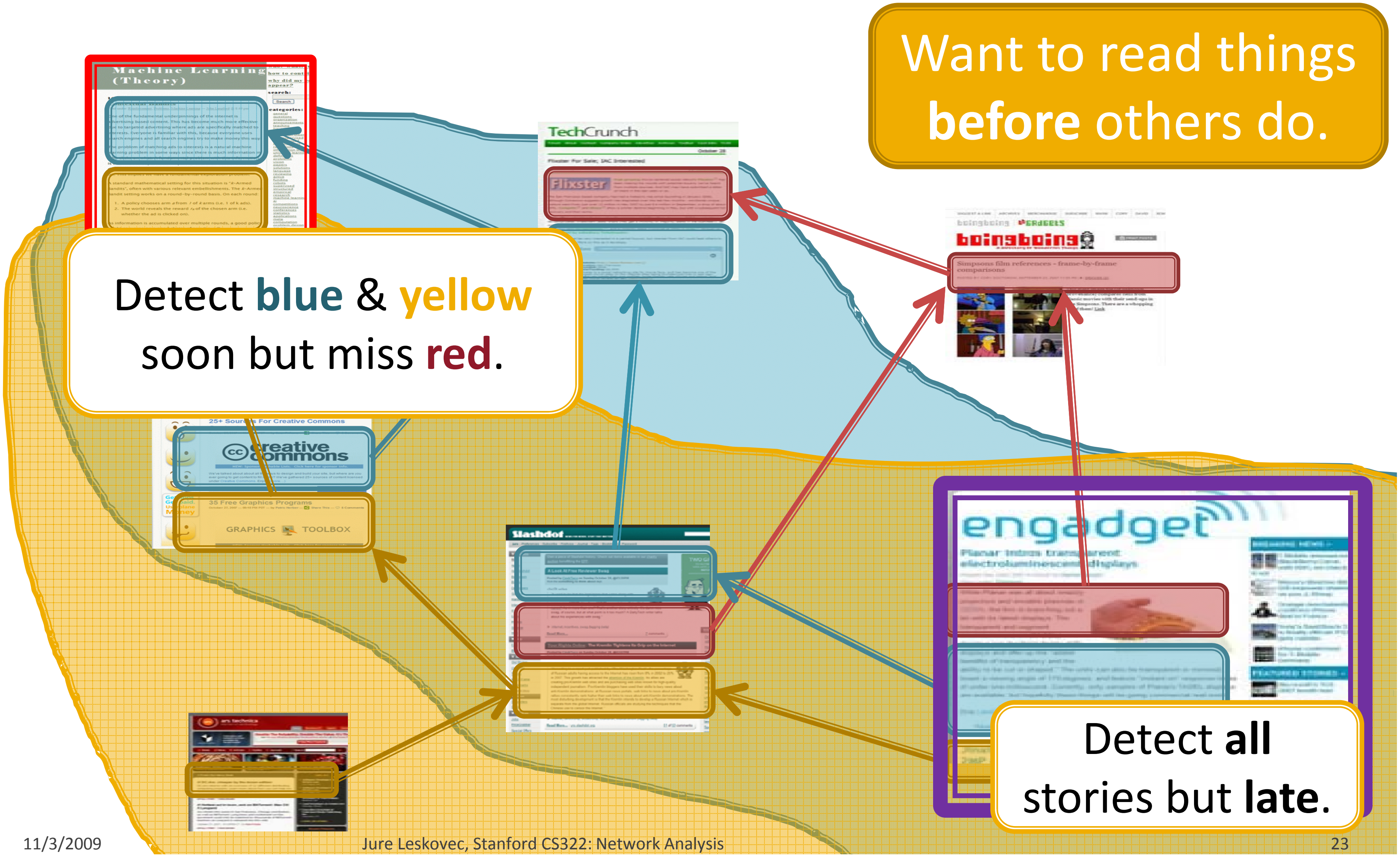

- CELF is an order of magnitude faster than hill-climbing

# Question...

- I have 10 minutes. Which blogs should I read to be most up to date?
- Who are the most influential bloggers?



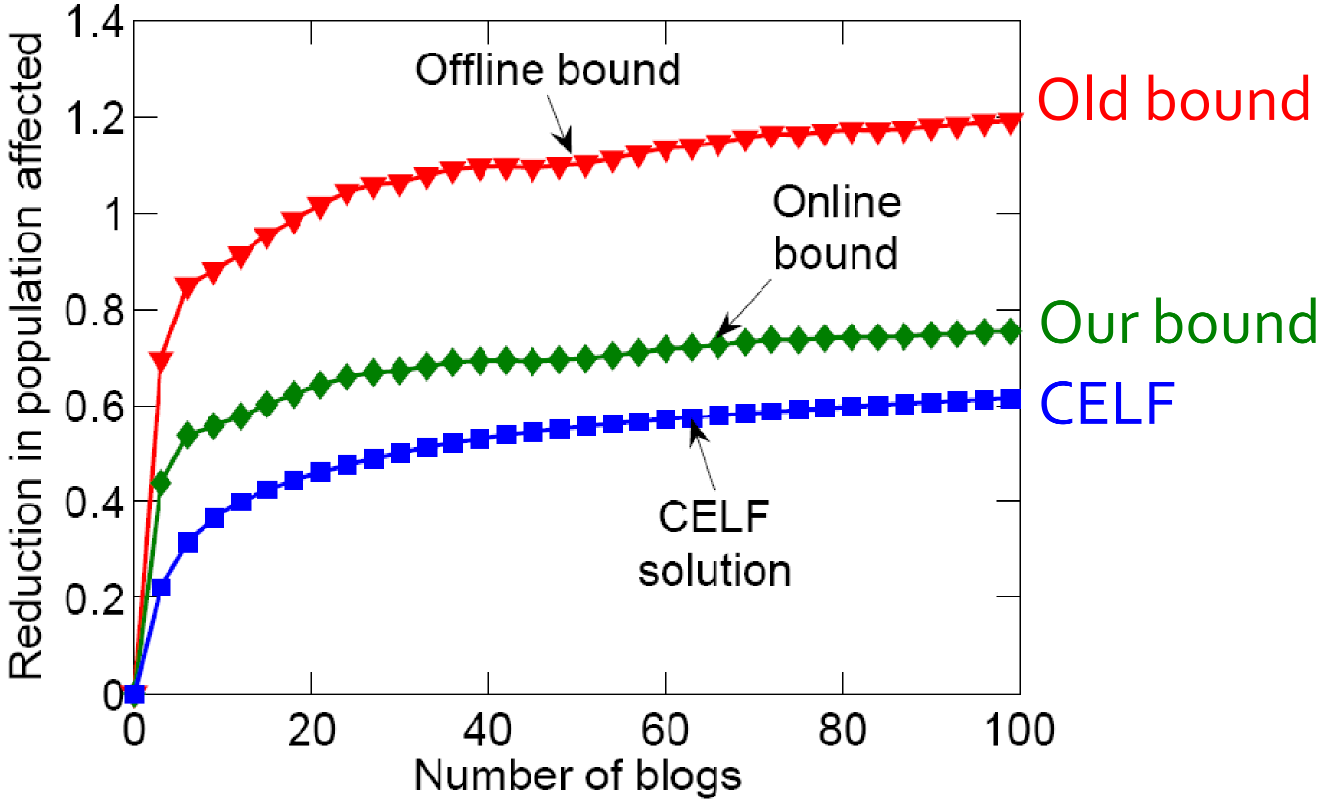
# Detecting information outbreaks



[Leskovec et al., KDD '07]

# Blogs: Solution Quality

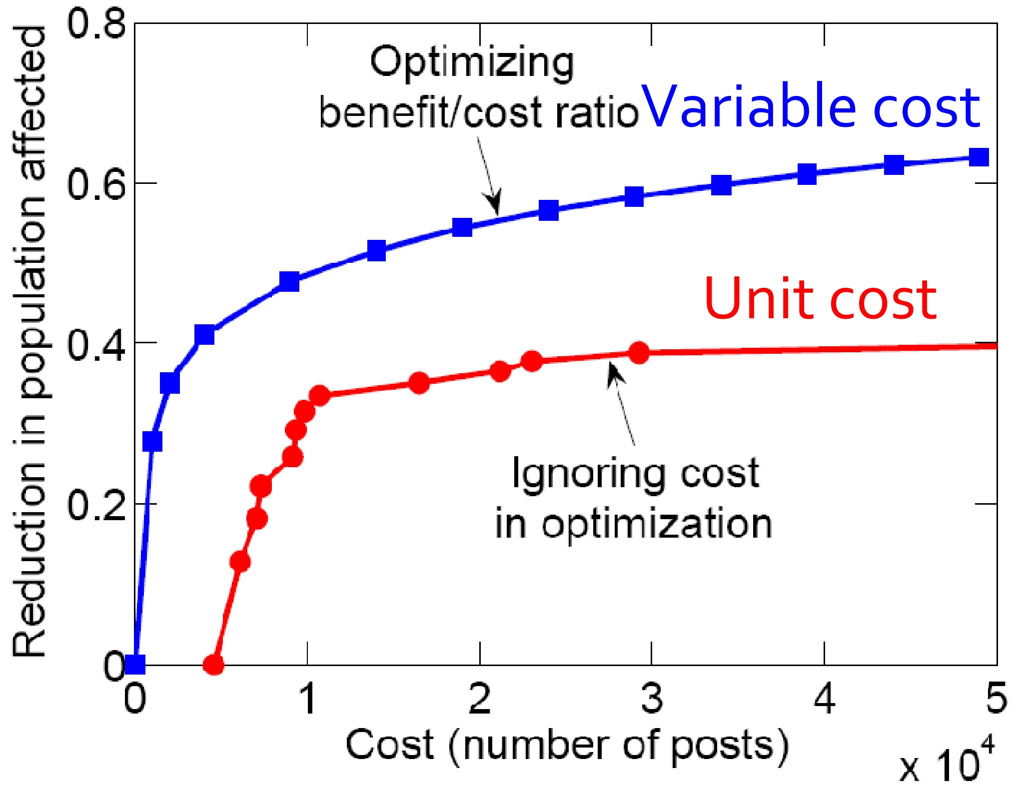
- Online bound [Leskovec et al., KDD '07] is much tighter
  - 13%



[Leskovec et al., KDD '07]

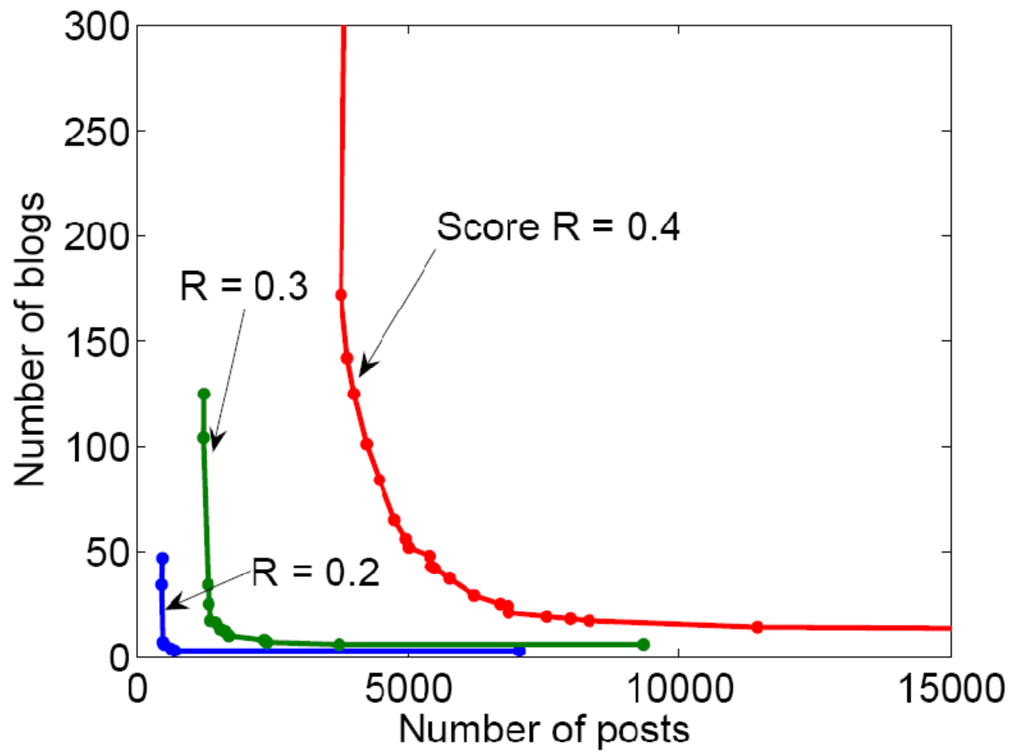
# Blogs: Cost of a Blog

- Unit cost:
  - algorithm picks large popular blogs: instapundit.com, michellemalkin.com
- Variable cost:
  - proportional to the number of posts
- We can do much better when considering costs



# Blogs: Cost of a Blog

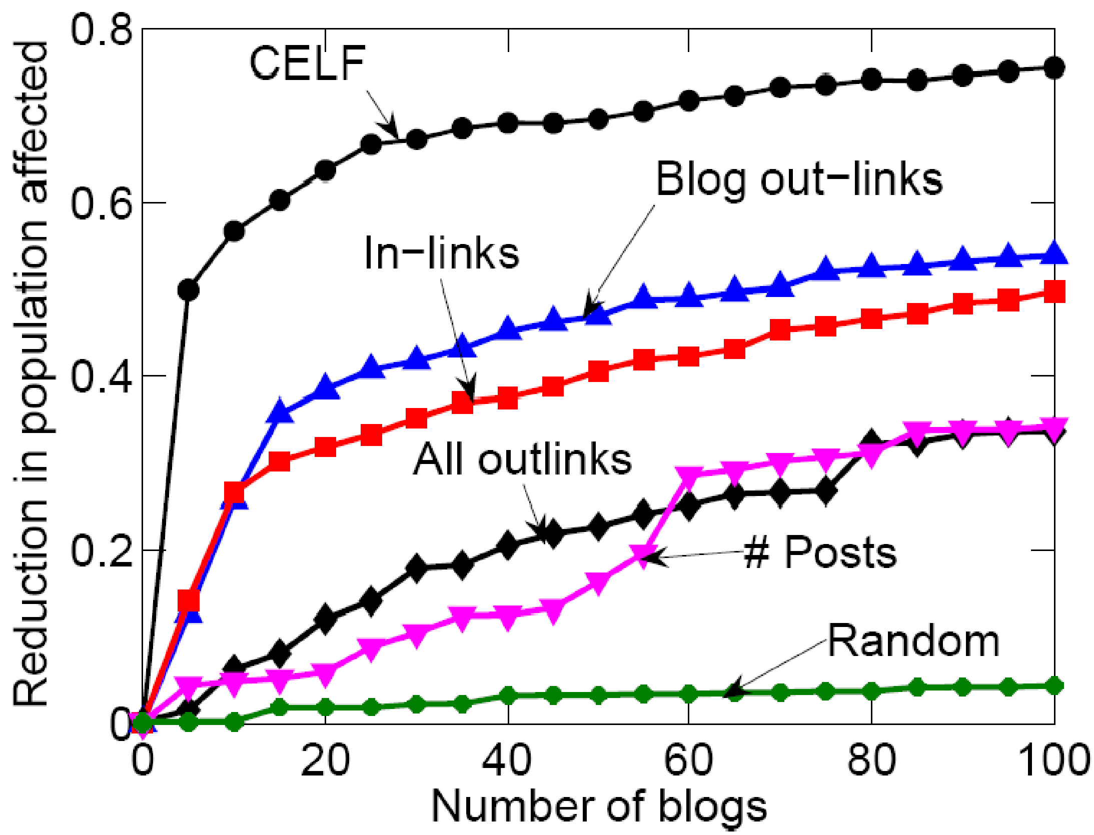
- But then algorithm picks lots of small blogs that participate in few cascades
- We pick best solution that interpolates between the costs
- We can get good solutions with few blogs and few posts




Each curve represents solutions with same final reward

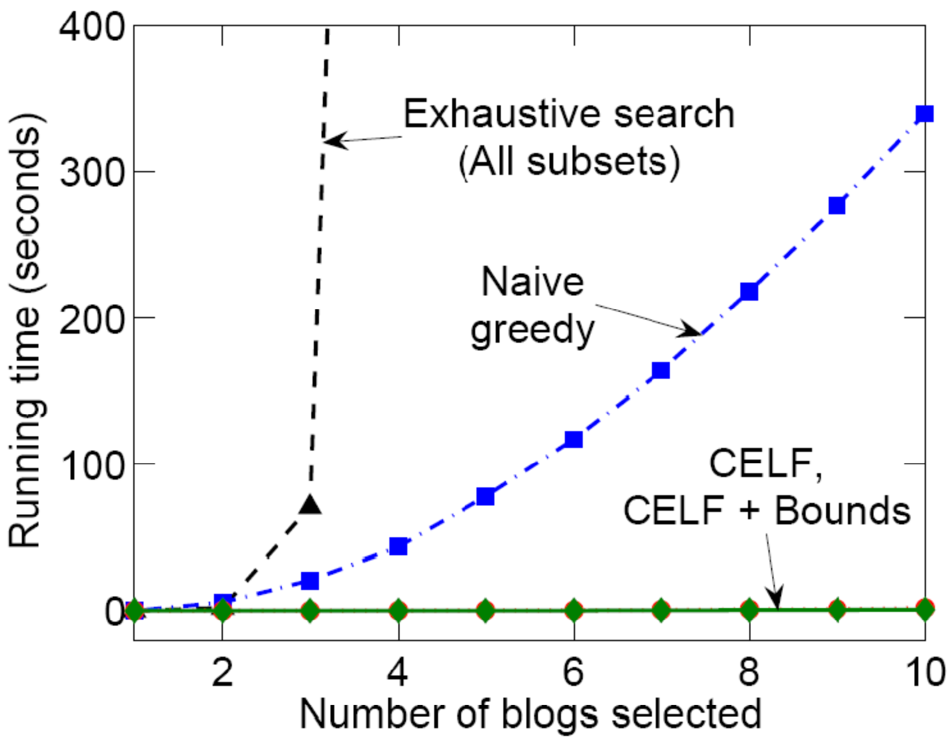
[Leskovec et al., KDD '07]

# Blogs: Heuristic Selection




- Heuristics perform much worse
- One really needs to perform optimization

# Blogs: Scalability

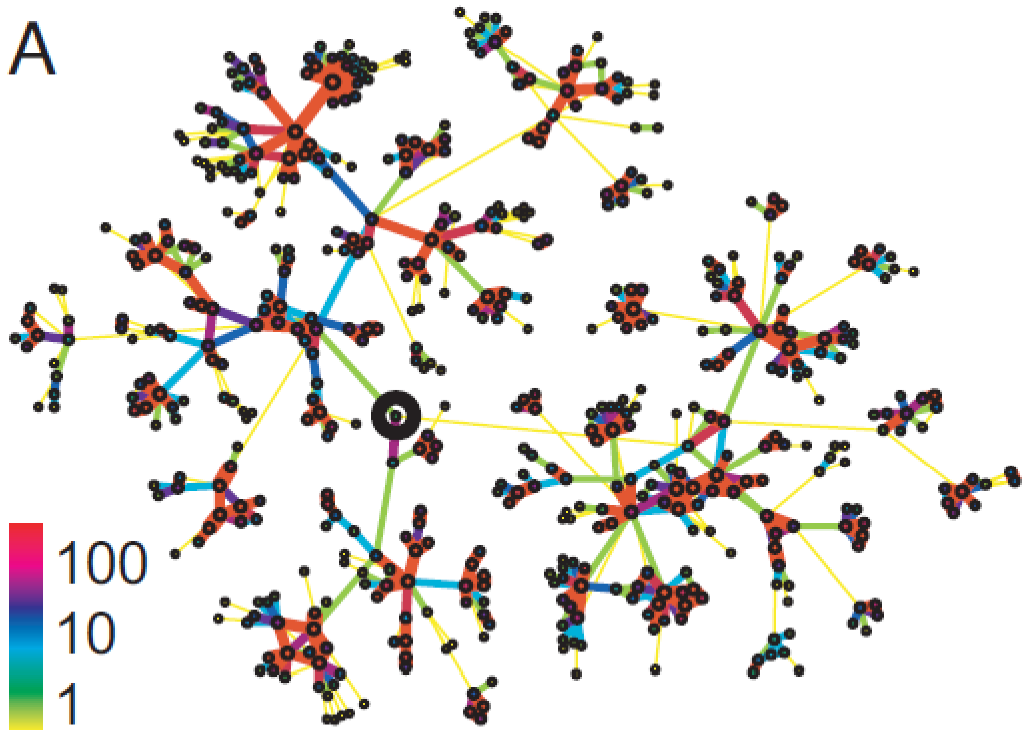



- CELF runs **700** times faster than simple hill-climbing algorithm

# Finding communities/clusters in networks

[Onnela et al. '07]

# Strength of weak ties

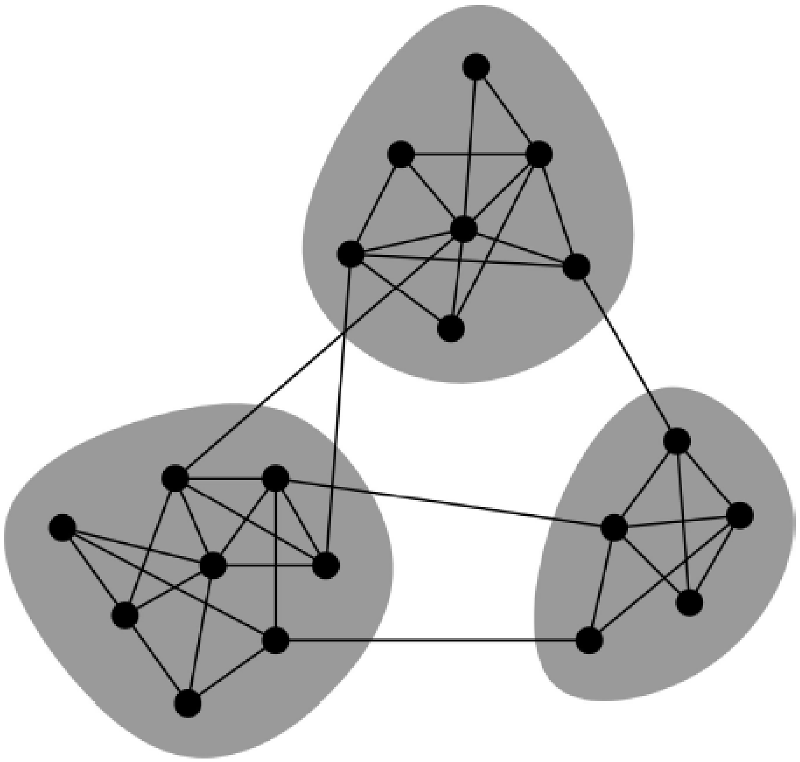



- Real edge strengths in mobile call graph

# Network communities

- Findings so far suggest that network groups are **tightly connected**

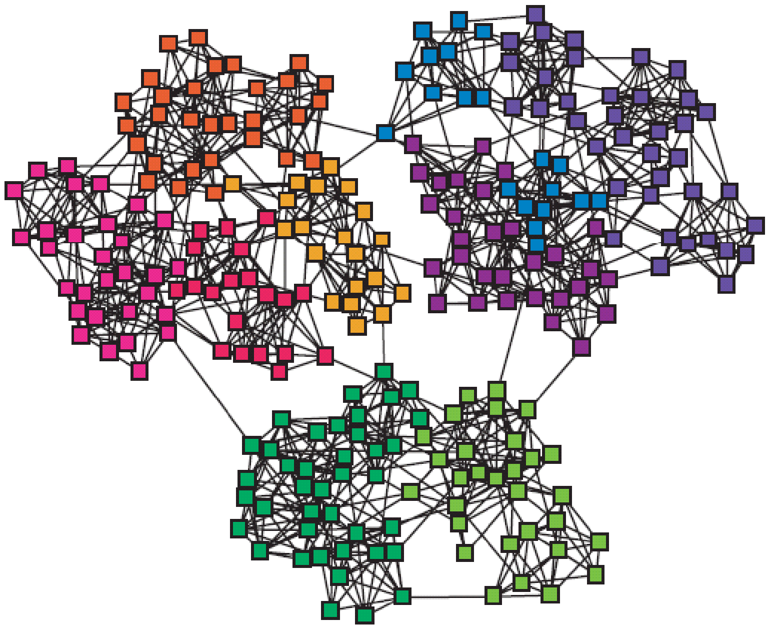
- **Network communities:**
  - Sets of nodes with **lots** of connections **inside** and **few** to **outside** (the rest of the network)



Communities, clusters, groups, modules

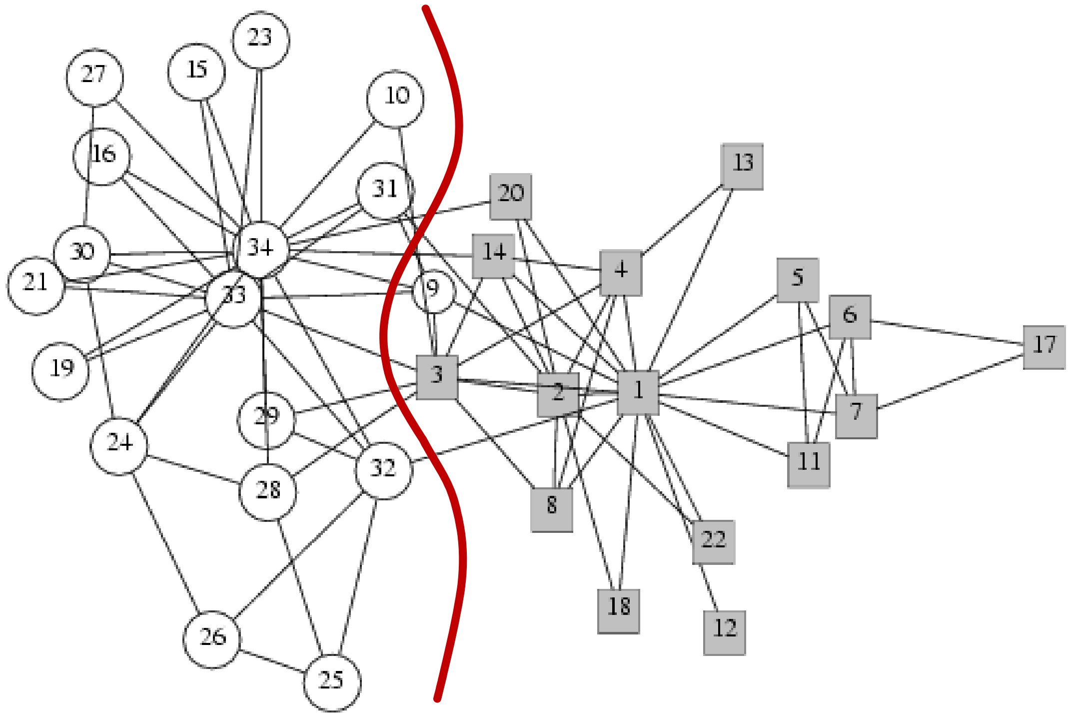
# Finding network communities

- How to automatically find such densely connected groups of nodes?
- Ideally such automatically detected clusters would then correspond to real groups
- For example:



Communities, clusters, groups, modules

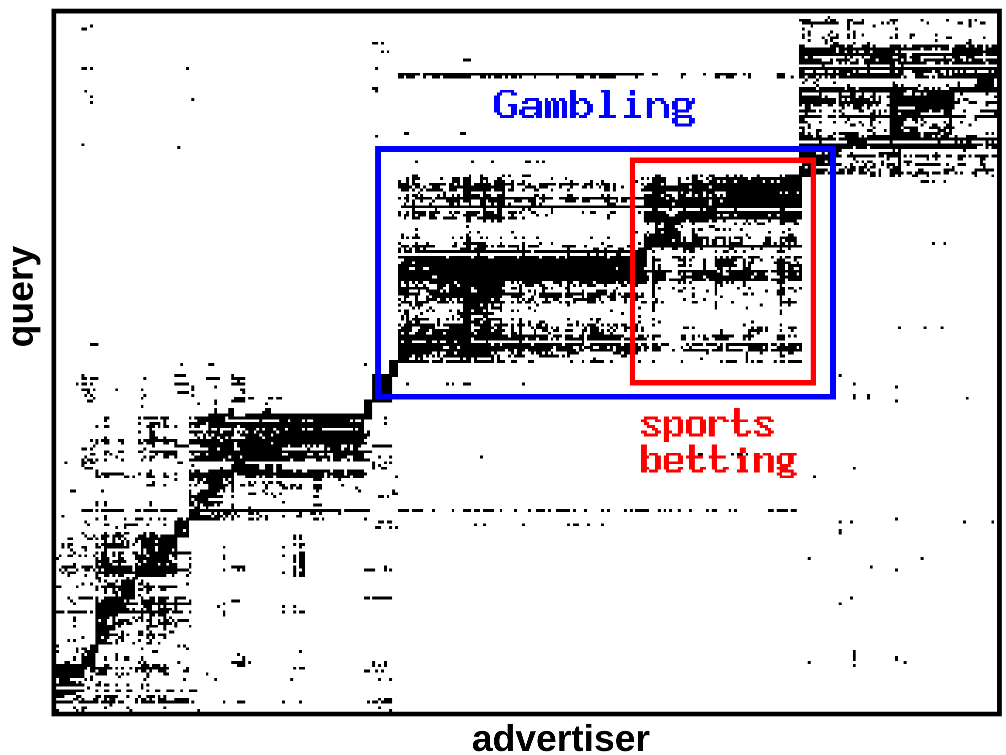
# Social Network Data



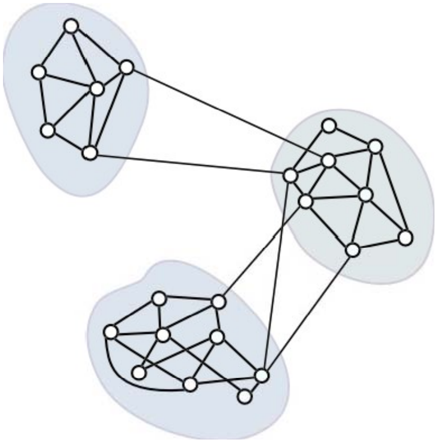

- Zachary's Karate club network:
  - Observe social ties and rivalries in a university karate club
  - During his observation, conflicts led the group to split
  - Split could be explained by a minimum cut in the network

# Micro-markets in sponsored search

Find micro-markets by partitioning the “query x advertiser” graph:



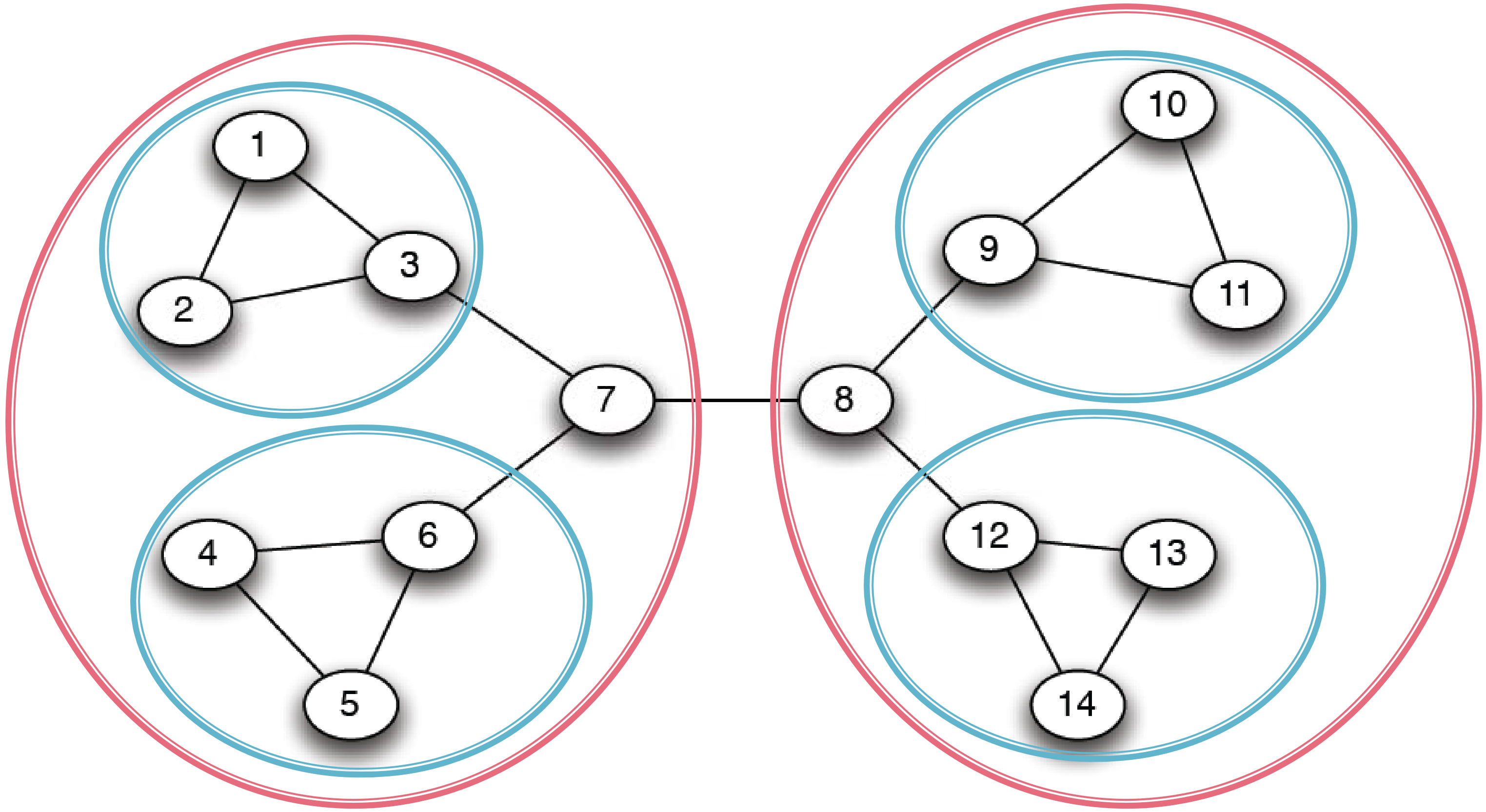
# Clustering and Community Finding



Many methods:

- Linear (low-rank) methods:
  - If Gaussian, then low-rank space is good
- Kernel (non-linear) methods:
  - If low-dimensional manifold, then kernels are good
- Hierarchical methods:
  - Top-down and bottom-up – common in social sciences
- Graph partitioning methods:
  - Define "edge counting" metric – conductance, expansion, modularity, etc. – and optimize!

# Hierarchically nested communites




What is a good notion that would extract such clusters?

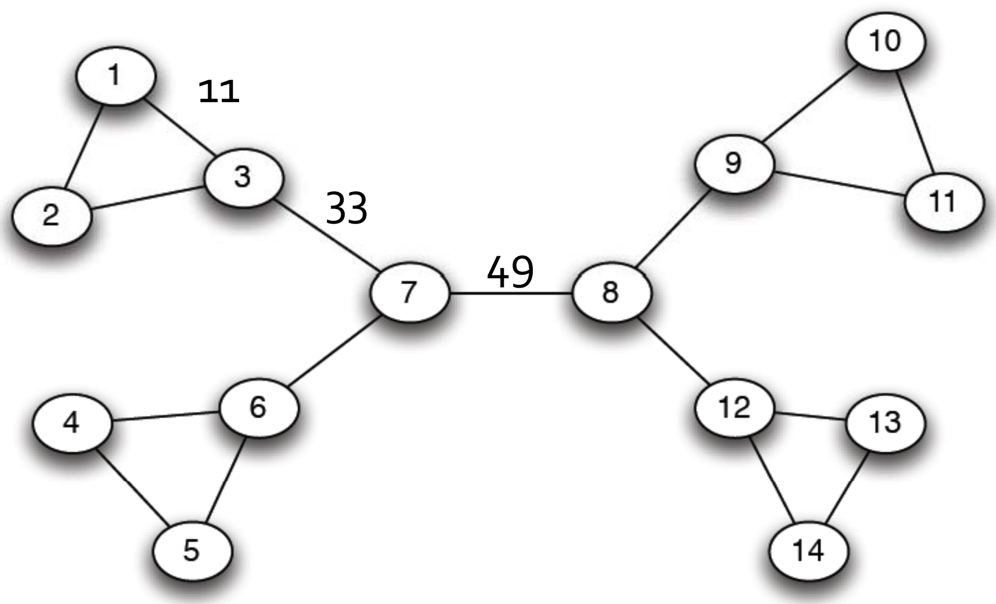
[Girvan-Newman PNAS '02]

# Algorithm of Girvan-Newman

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

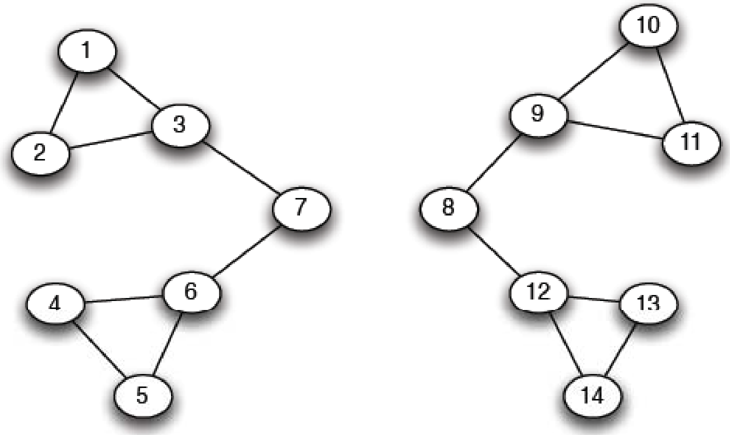
  Number of shortest paths passing through the edge

- Remove edges in decreasing betweenness

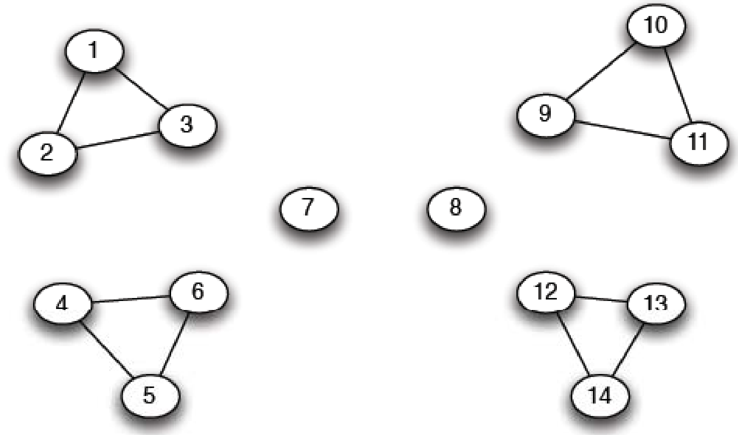


[Girvan-Newman PNAS '02]

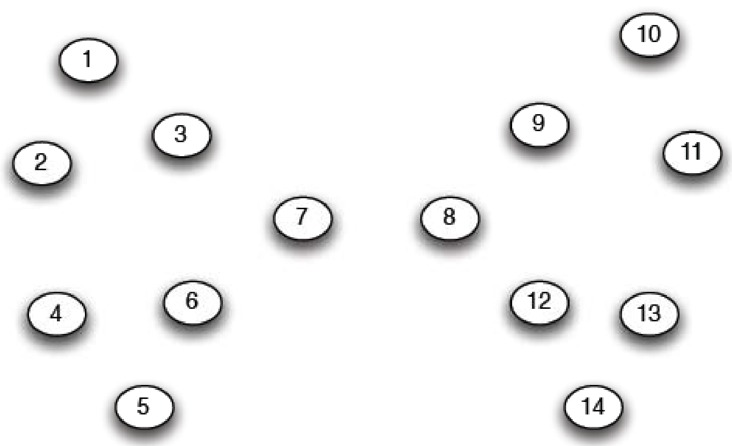
# Algorithm of Girvan-Newman




(a) *Step 1*




(b) *Step 2*

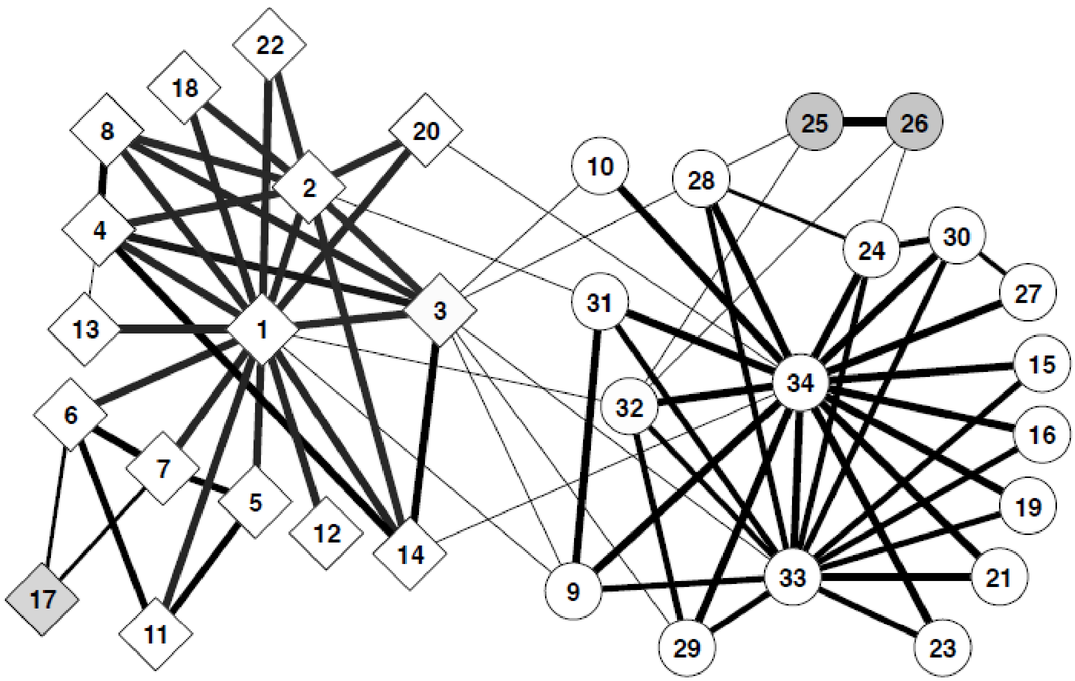


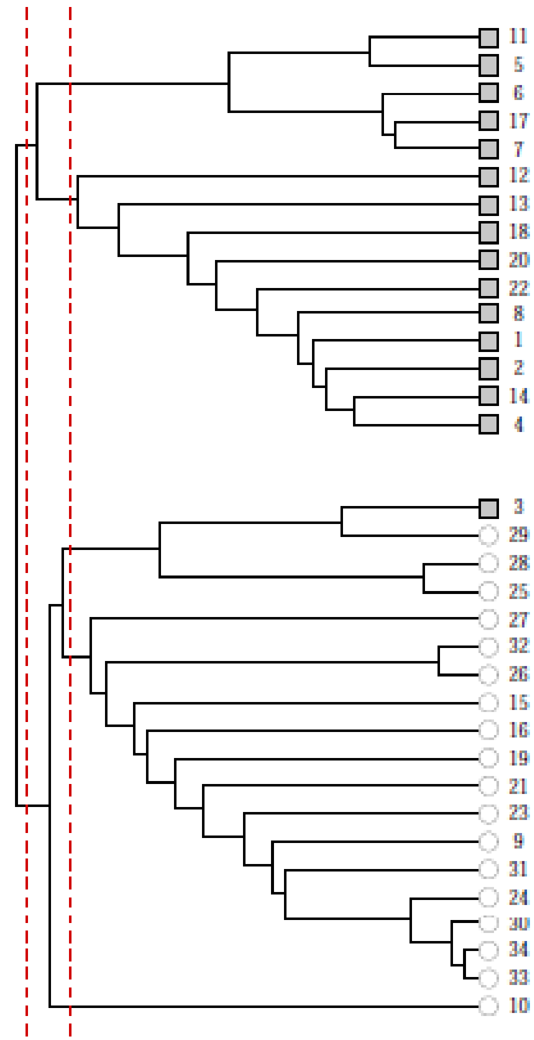

(c) *Step 3*

[Newman-Girvan PhysRevE '03]

# Girvan-Newman: Results

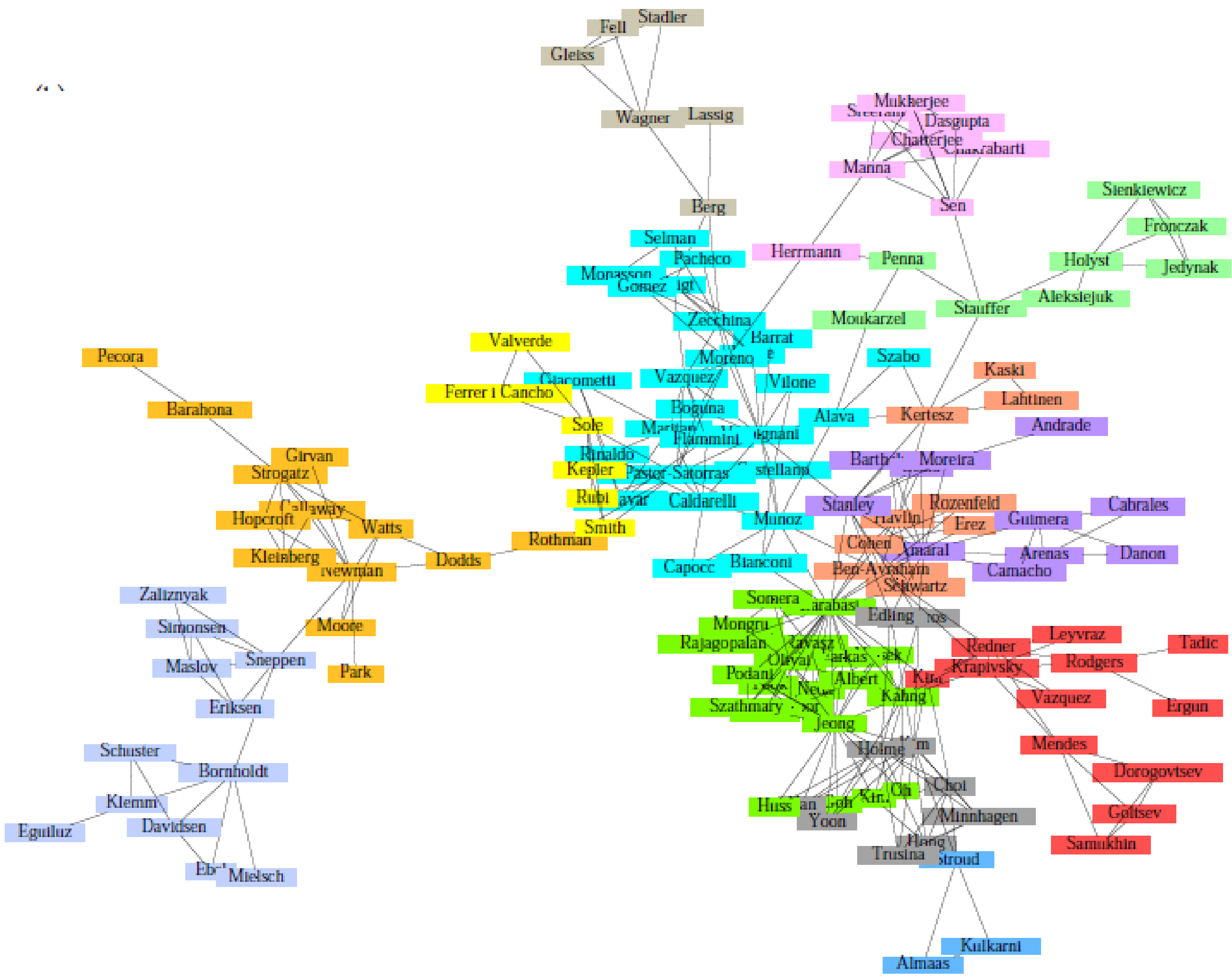
- Zachary's Karate club: hierarchic decomposition



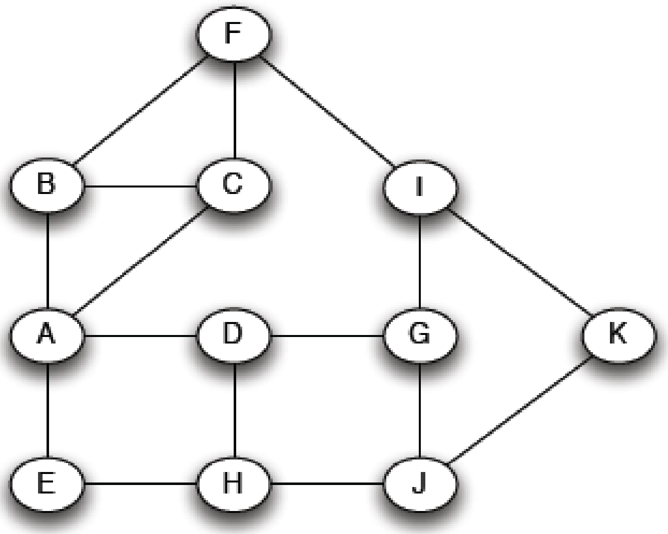
[Newman-Girvan PhysRevE '03]

# Girvan-Newman: Results



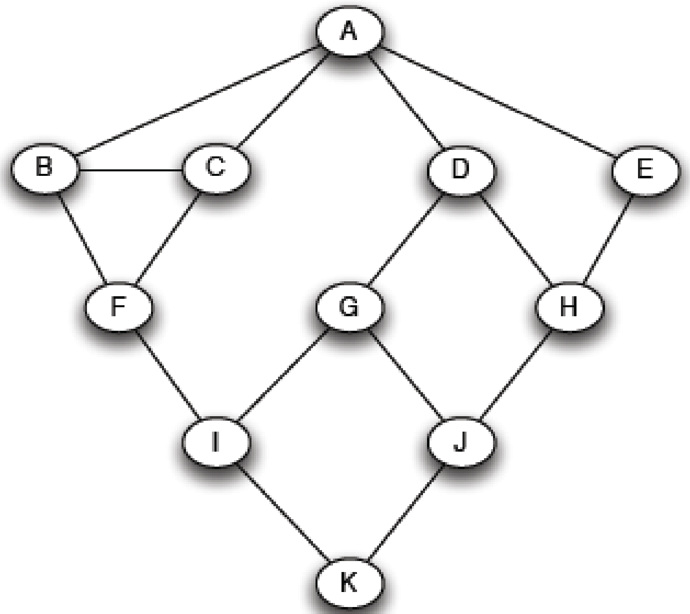
Communities in physics collaborations

# How to compute betweenness (1)



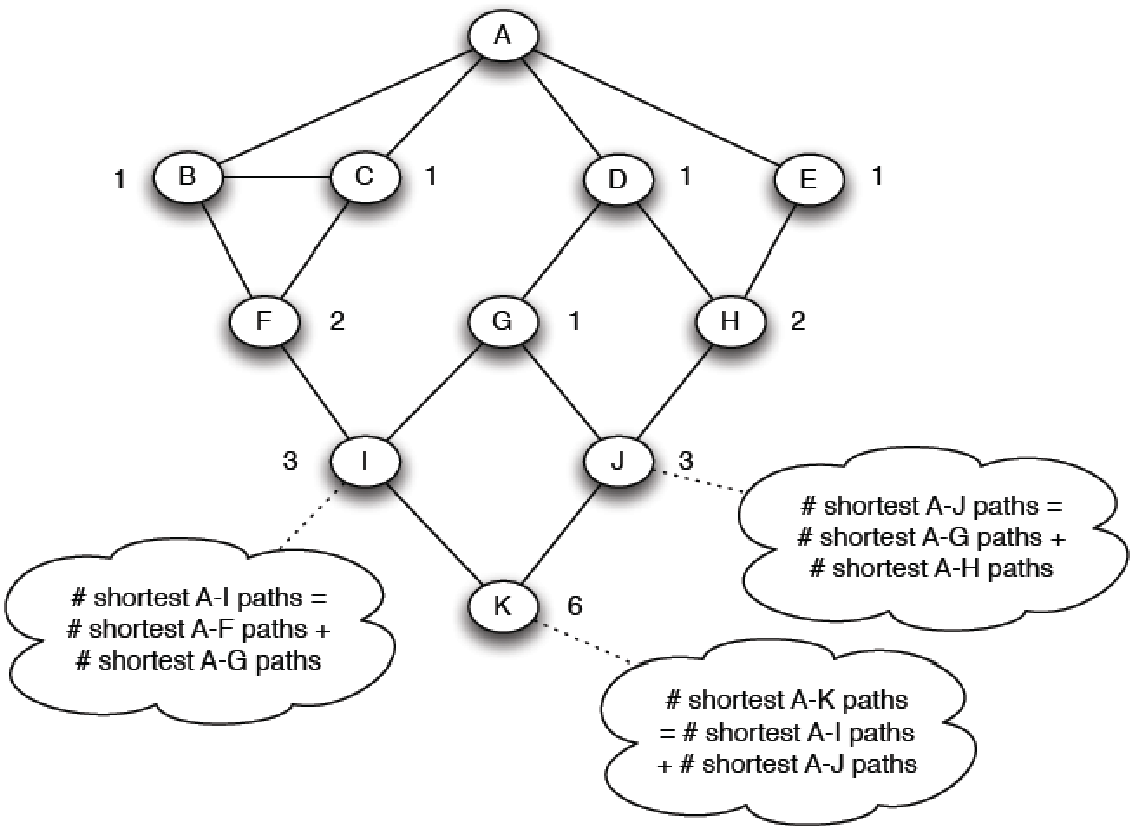

- Want to compute betweenness of paths starting at node A

- Breath first search starting from A:



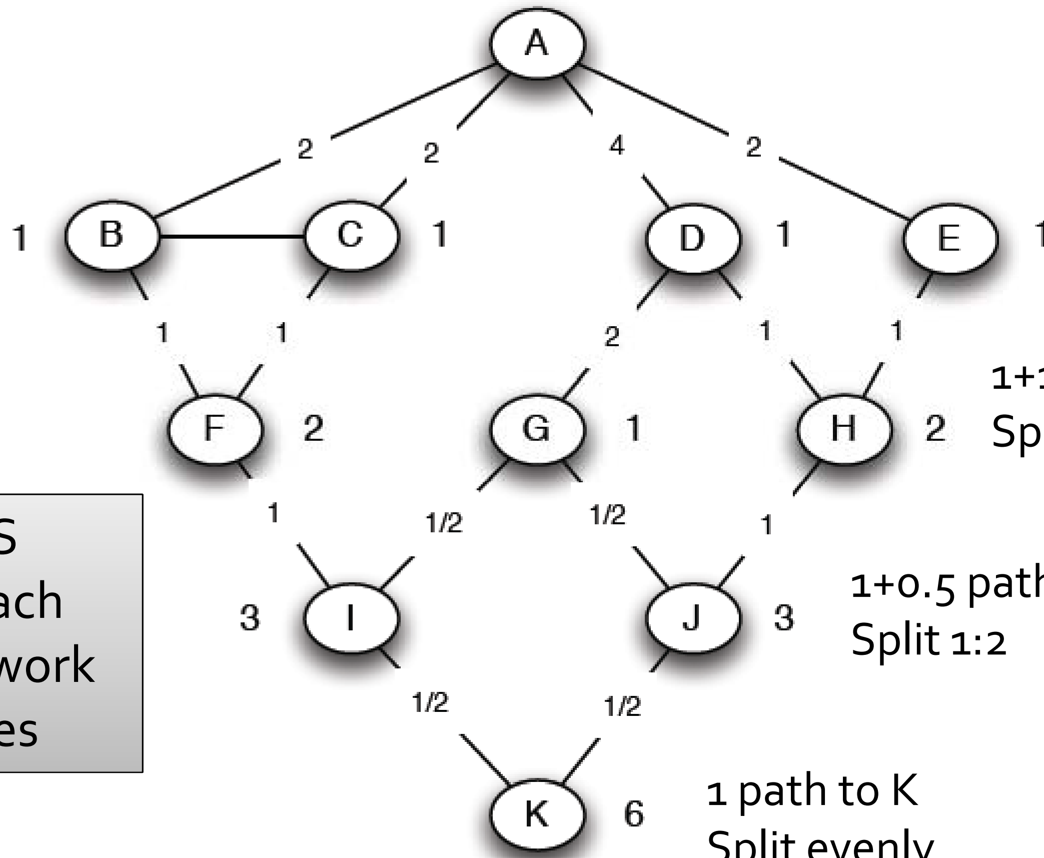
# How to compute betweenness (2)

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



# How to compute betweenness (3)

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




1+1 paths to H
Split evenly

1+0.5 paths to J
Split 1:2

1 path to K
Split evenly

- Repeat the BFS procedure for each node of the network
- Add edge scores