Link Prediction and Network Inference

CS224W: Social and Information Network Analysis

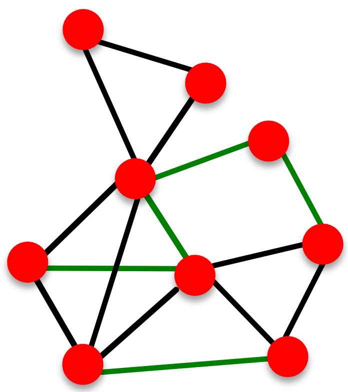
Jure Leskovec, Stanford University

http://cs224w.stanford.edu

[LibenNowell-Kleinberg '03]

Link Prediction in Networks

- **The link prediction task:**
  - Given $G[t_0, t'_0]$ a graph on edges up to time t'_0, **output a ranked list L** of links (not in $G[t_0, t'_0]$) that are predicted to appear in $G[t_1, t'_1]$



$G[t_0, t'_0]$
$G[t_1, t'_1]$

- **Evaluation:**
  - $n = |E_{new}|$: # new edges that appear during the test period $[t_1, t'_1]$
  - Take top n elements of L and count correct edges

[LibenNowell-Kleinberg '03]

Link Prediction via Proximity

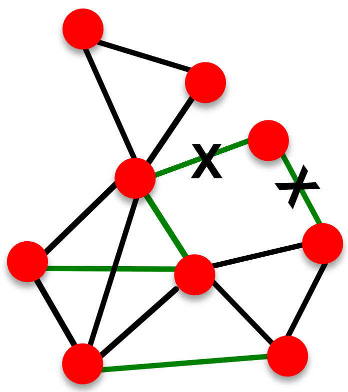
- **Predict links in a evolving collaboration network**

	training period			Core		
	authors	papers	collaborations[1]	authors	$\lvert E_{old} \rvert$	$\lvert E_{new} \rvert$
astro-ph	5343	5816	41852	1561	6178	5751
cond-mat	5469	6700	19881	1253	1899	1150
gr-qc	2122	3287	5724	486	519	400
hep-ph	5414	10254	47806	1790	6654	3294
hep-th	5241	9498	15842	1438	2311	1576

- **Core:** Because network data is very sparse
  - Consider only nodes with degree of at least 3
    - Because we don't know enough about nodes with less than 3 edges to make good inferences

Link Prediction via Proximity

- **Methodology:**
  - For each pair of nodes *(x,y)* compute score *c(x,y)*
    - For example, *c(x,y)* could be the # of common neighbors of *x* and *y*
  - Sort pairs *(x,y)* by the decreasing score *c(x,y)*
    - **Note:** Only consider/predict edges where both endpoints are in the core (*deg.* ≥ *3*)
  - **Predict top n pairs as new links**
  - **See which of these links actually appear in $G[t_1, t_1']$**



[LibenNowell-Kleinberg '03]

Link Prediction via Proximity

- **Different scoring functions** $c(x, y) =$
  - **Graph distance:** (negated) Shortest path length
  - **Common neighbors:** $|\Gamma(x) \cap \Gamma(y)|$
  - **Jaccard's coefficient:** $|\Gamma(x) \cap \Gamma(y)| / |\Gamma(x) \cup \Gamma(y)|$
  - **Adamic/Adar:** $\sum_{z \in \Gamma(x) \cap \Gamma(y)} 1 / \log |\Gamma(z)|$
  - **Preferential attachment:** $|\Gamma(x)| \cdot |\Gamma(y)|$
  - **PageRank:** $r_x(y) + r_y(x)$
    - $r_x(y)$ … stationary distribution score of *y* under the random walk:
      - with prob. 0.15, jump to *x*
      - with prob. 0.85, go to random neighbor of current node

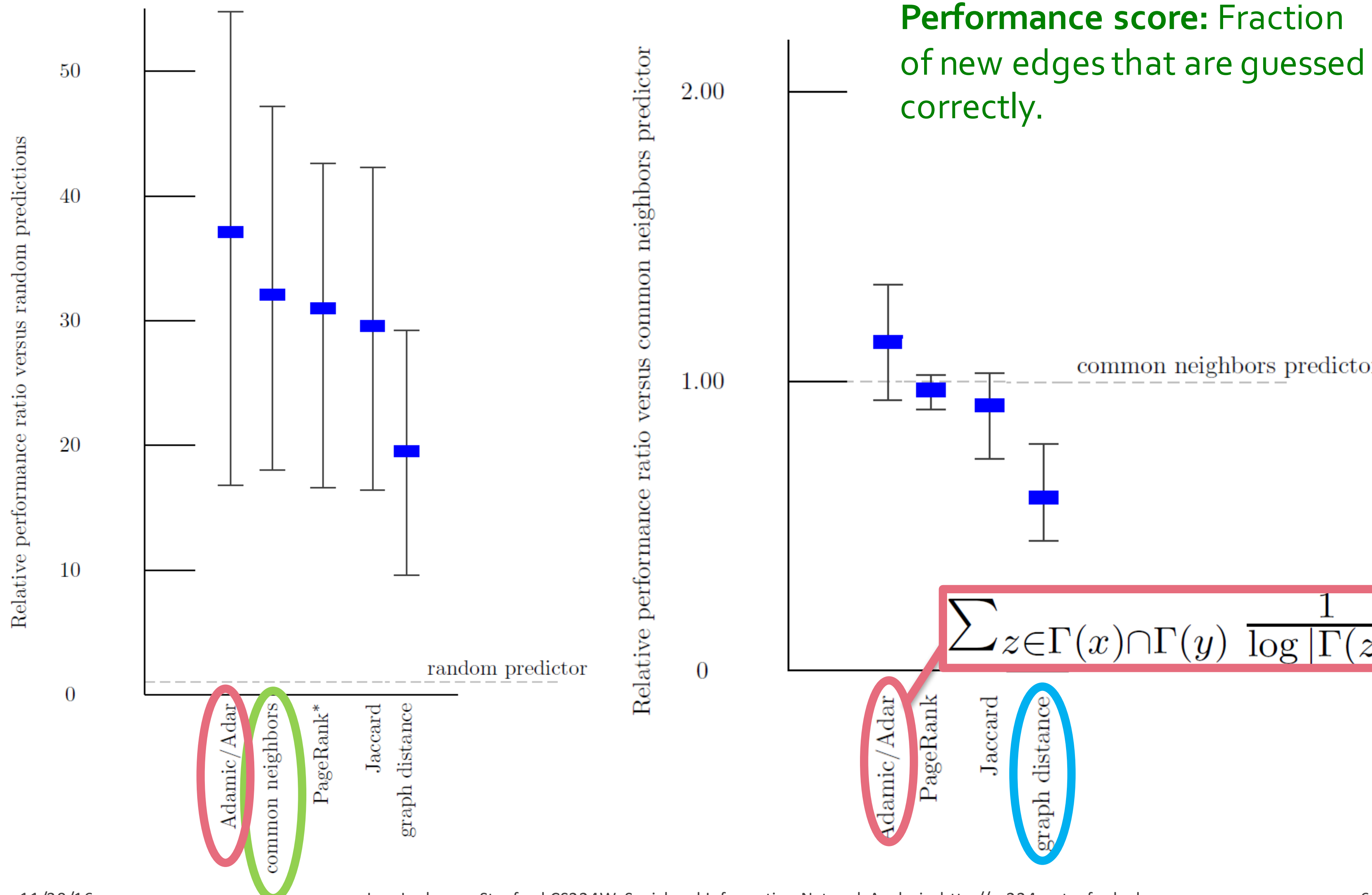
Γ(x) … neighbors of node x

- **Then, for a particular choice of *c(·)***
  - For every pair of nodes *(x,y)* compute *c(x,y)*
  - Sort pairs *(x,y)* by the decreasing score *c(x,y)*
  - **Predict top *n* pairs as new links**

[LibenNowell-Kleinberg ' 03]

Results: Improvement

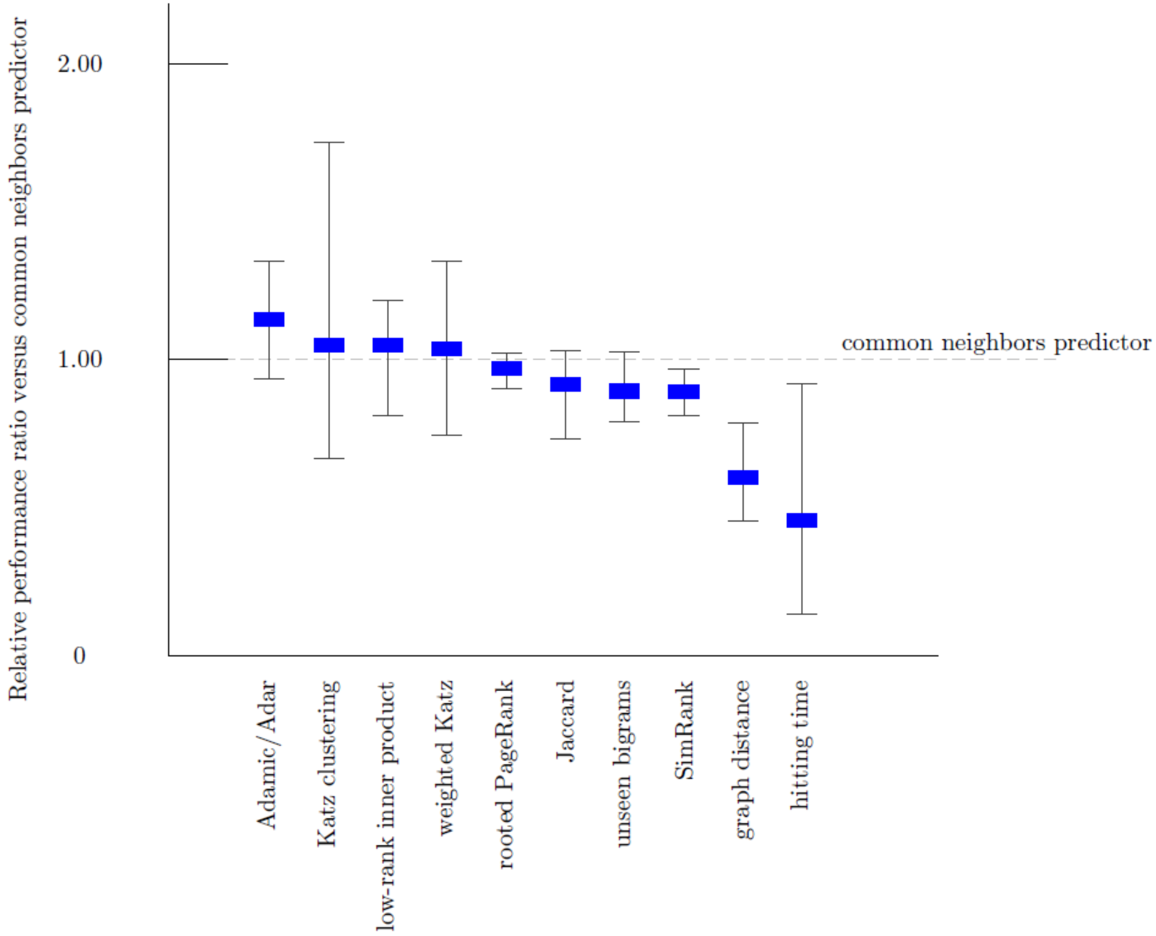
Performance score: Fraction of new edges that are guessed correctly.



$$\sum_{z\in\Gamma(x)\cap\Gamma(y)} \frac{1}{\log|\Gamma(z)|}$$

Results: Common Neighbors

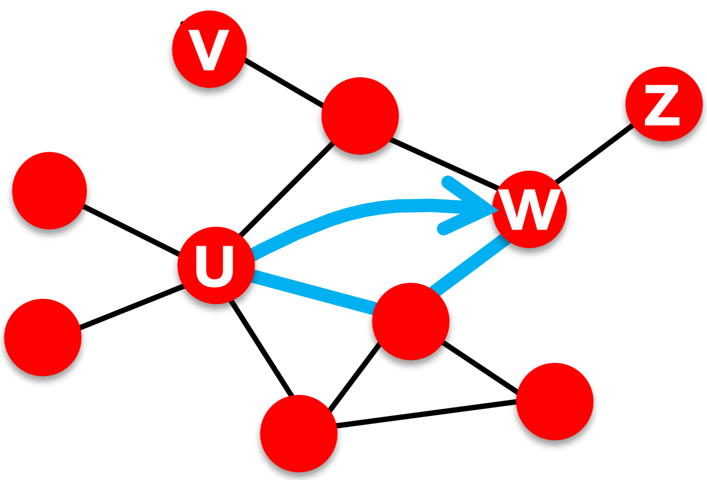
- **Improvement over #common neighbors**

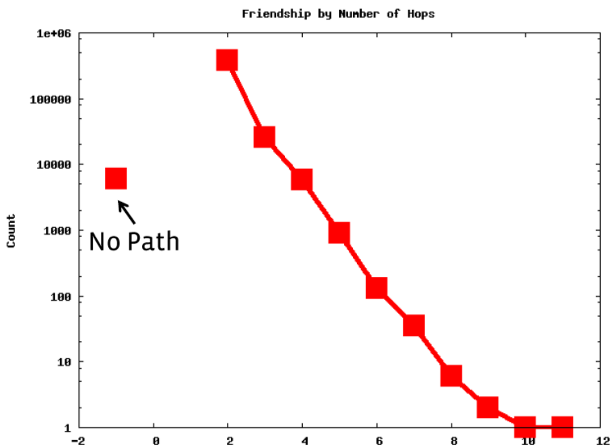


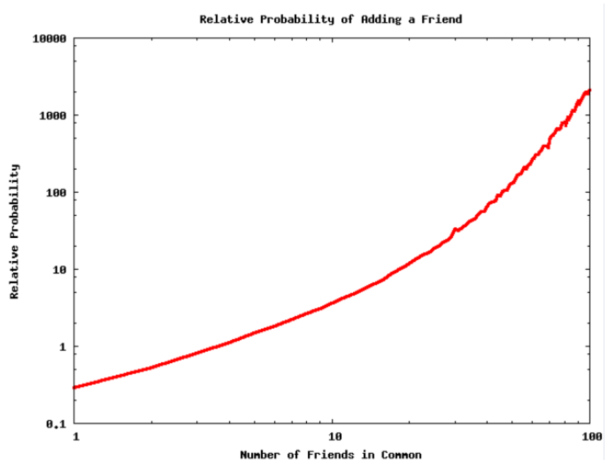
Supervised Random Walks for Link Prediction

Supervised Link Prediction

- **Can we learn to predict new friends?**
  - **Facebook's People You May Know**
  - **Let's look at the FB data:**
    - 92% of new friendships on FB are friend-of-a-friend
    - More mutual friends helps

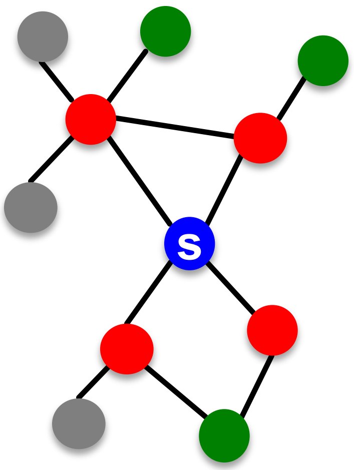






Supervised Link Prediction

- **Goal:** **Recommend a list of possible friends**
- **Supervised machine learning setting:**
  - **Labeled training examples:**
    - For every user s have a list of others she will create links to $\{d_1 \ \dots \ d_k\}$ **in the future**
      - *U*se FB network from May 2012 and $\{d_1 \ \dots \ d_k\}$ are the new friendships you created since then
      - These are the "positive" training examples
    - Use all other users as "negative" example
  - **Task:**
    - For a given node s, **score** nodes $\{d_1 \ \dots \ d_k\}$ **higher** than any other node in the network

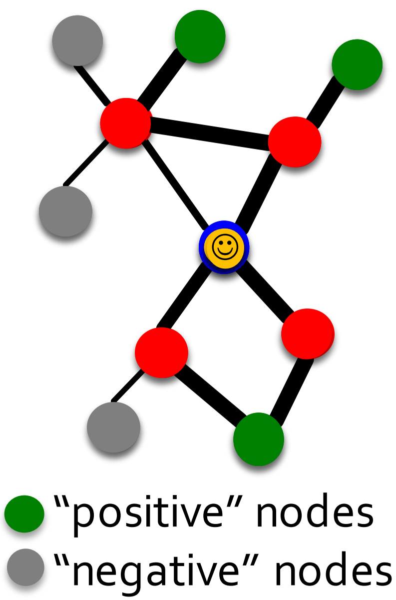


"positive" nodes
"negative" nodes

Green nodes are the nodes to which **s** creates links in the future

Supervised Link Prediction

- **How to combine node/edge features and the network structure?**
  - Estimate **strength** of each friendship (u, v) using:
    - Profile of user u, profile of user v
    - Interaction history of users u and v
  - This creates a **weighted graph**
  - Do **Personalized PageRank from s** and measure the "**proximity**" (the visiting prob.) of any other node w from s
  - Sort nodes w by decreasing **"proximity"**

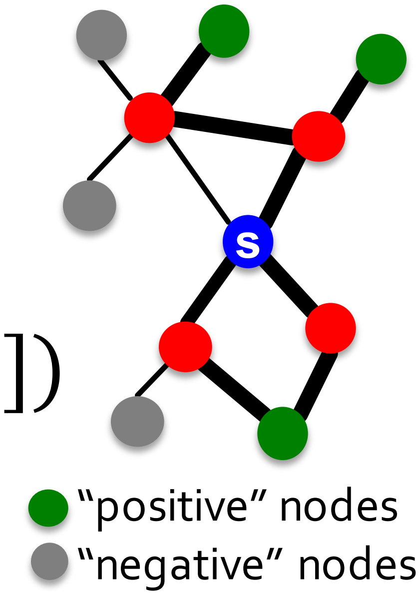


Supervised Random Walks

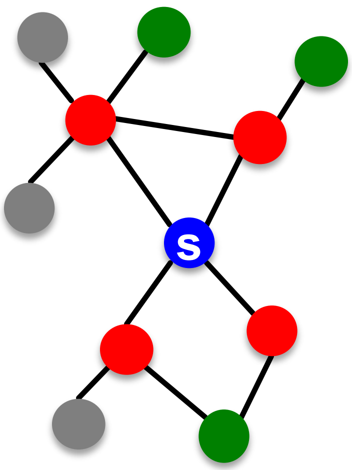
- Let s be the starting node
- Let $f_\beta(u,v)$ be a function that assigns **strength a_{uv} to edge (u,v)**

$$a_{uv} = f_\beta(u,v) = \exp\left(-\sum_i \beta_i \cdot \mathrm{x}_{uv}[i]\right)$$

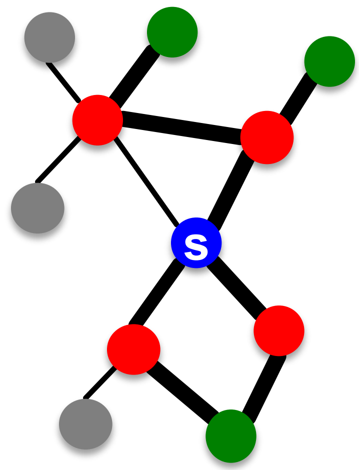
  - **x_{uv} is a feature vector of (u,v)**
    - Features of node u
    - Features of node v
    - Features of edge (u,v)
  - **Note: β is the weight vector we will later estimate!**
- Do **Random Walk with Restarts** from s where transitions are according to edge strengths a_{uv}



SRW: Prediction



Network



Set edge strengths
$a_{uv} = f_\beta(u,v)$

Random Walk with Restarts on the weighted graph.
Each node w has a PageRank proximity p_w

Sort nodes w by the decreasing PageRank score p_w

Recommend top k nodes with the highest proximity p_w to node s

- **How to estimate edge strengths?**
  - **How to set parameters β of $f_\beta(u,v)$?**
- **Idea:** Set β such that it (correctly) predicts known future links

Personalized PageRank

[WSDM '11]

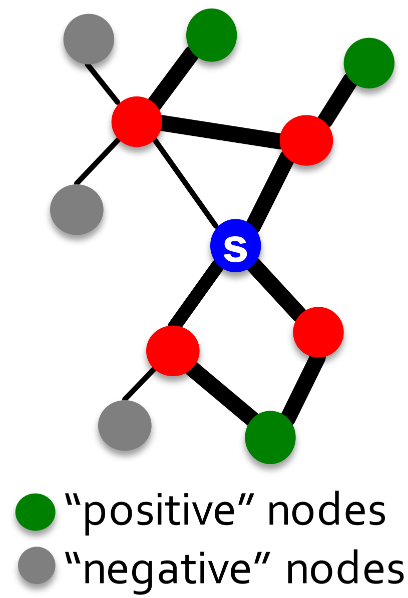
- a_{uv} .... Strength of edge (u, v)
- **Random walk transition matrix:**

$$Q'_{uv} = \begin{cases} \frac{a_{uv}}{\sum_w a_{uw}} & \text{if } (u,v) \in E, \\ 0 & \text{otherwise} \end{cases}$$

- **PageRank transition matrix:**

$$Q_{ij} = (1-\alpha)Q'_{ij} + \alpha \mathbf{1}(j = s)$$

  - Where with prob. α we jump back to node s
- **Compute PageRank vector:** $p = p^T Q$
- **Rank** nodes w by decreasing p_w

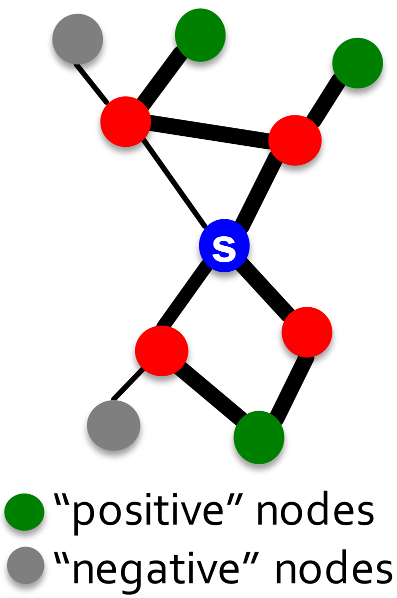


The Optimization Problem

- **Positive** examples
  $D = \{d_1, \ldots, d_k\}$
- **Negative** examples
  $L = \{other\ nodes\}$
- **What do we want?**

$$\min_{\beta} F(\beta) = ||\beta||^2$$

We prefer small weights β to prevent overfitting



such that

$$\forall\ d \in D, l \in L:\ p_l < p_d$$

Every positive example has to have higher PageRank score than every negative example

- **Note:**
  - Exact solution to this problem may not exist
  - So we make the constraints "soft" (i.e., optional)

Making Constraints "Soft"

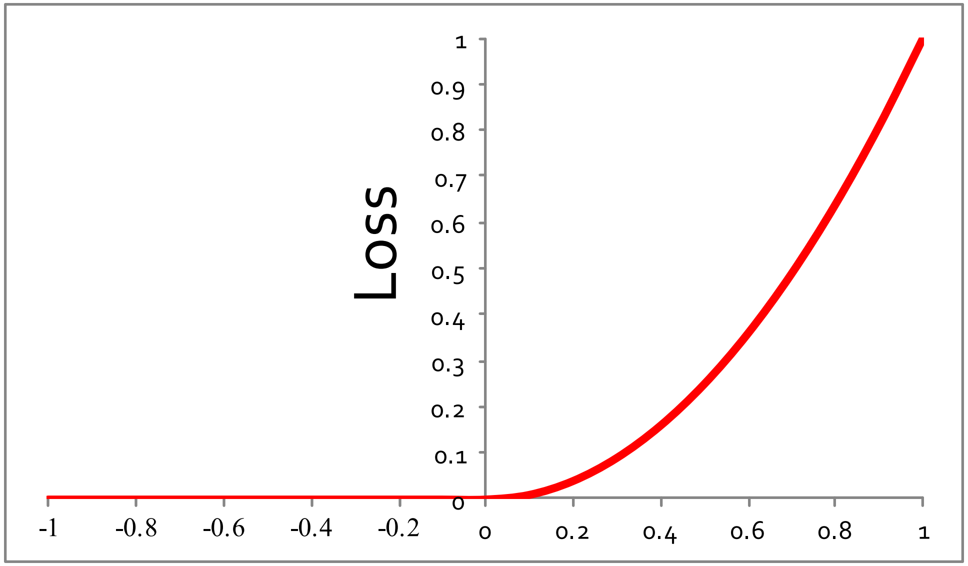
[WSDM '11]

- **Want to minimize:**

$$\min_{\beta} F(\beta) = \sum_{d \in D, l \in L} h(p_l - p_d) + \lambda ||\beta||^2$$

Penalty for violating the constraint that $p_d > p_l$

- **Loss:** $h(x) = 0$ if $x < 0$, or x^2 else

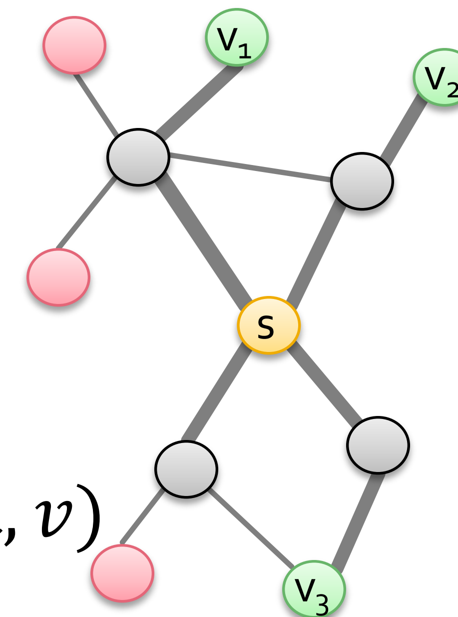


$p_l < p_d$ $p_l = p_d$ $p_l > p_d$

Solving the problem: Intuition

- **Want to minimize *F(β)***

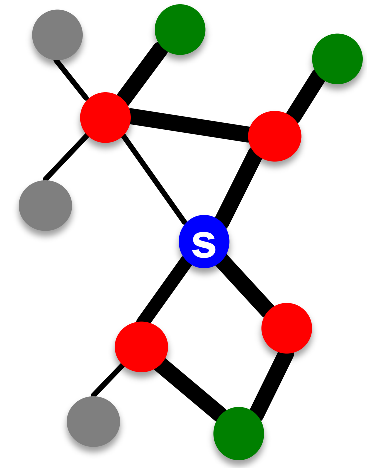
$$\min_{\beta} F(\beta) = \sum_{d\in D, l\in L} h(p_l - p_d) + \lambda ||\beta||^2$$



- **Both p_l and p_d depend on β**
  - Given β assign edge weights $a_{uv} = f_\beta(u, v)$
  - Using $Q = [a_{uv}]$ compute PageRank scores p_β
  - Rank nodes by the decreasing score

- **Goal: Want to find β such that $p_l < p_d$**

Solving the Problem: Intuition



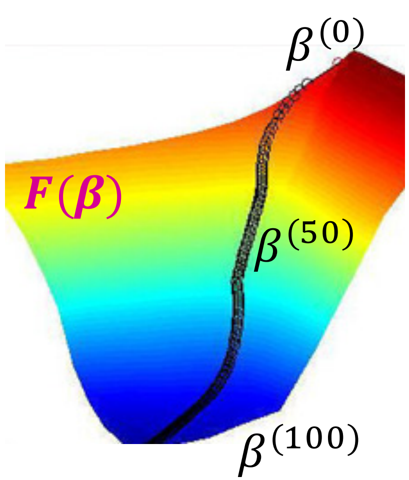
- **How to minimize $F(\beta)$?**

$$\min_{\beta} F(\beta) = \sum_{d \in D, l \in L} h(p_l - p_d) + \lambda ||\beta||^2$$

- **Idea:**
  - Start with some random $\beta^{(0)}$
  - Evaluate the derivative of $F(\beta)$ and do a small step in the opposite direction

$$\beta^{(t+1)} = \beta^{(t)} - \eta \frac{\partial F(\beta^{(t)})}{\partial \beta}$$

  - Repeat until convergence



Gradient Descent

Details!

$$F(\beta) = \sum_{d \in D, l \in L} h(p_l - p_d) + \lambda ||\beta||^2$$

- **What's the derivative $\frac{\partial F(\beta^{(t)})}{\partial \beta}$?**

$$\frac{\partial F(\beta)}{\partial \beta} = \sum_{l,d} \frac{\partial h(p_l - p_d)}{\partial \beta} + 2\lambda\beta$$

$$= \sum_{l,d} \frac{\partial h(p_l - p_d)}{\partial (p_l - p_d)} \left(\frac{\partial p_l}{\partial \beta} - \frac{\partial p_d}{\partial \beta}\right) + 2\lambda\beta$$

$h(x) = \max\{x, 0\}^2$

Easy!

- **We know:**

$p = p^T Q$ *that is* $p_u = \sum_j p_j Q_{ju}$

- **So:**

$$\boxed{\frac{\partial p_u}{\partial \beta} = \sum_j Q_{ju} \frac{\partial p_j}{\partial \beta} + p_j \frac{\partial Q_{ju}}{\partial \beta}}$$

Gradient Descent

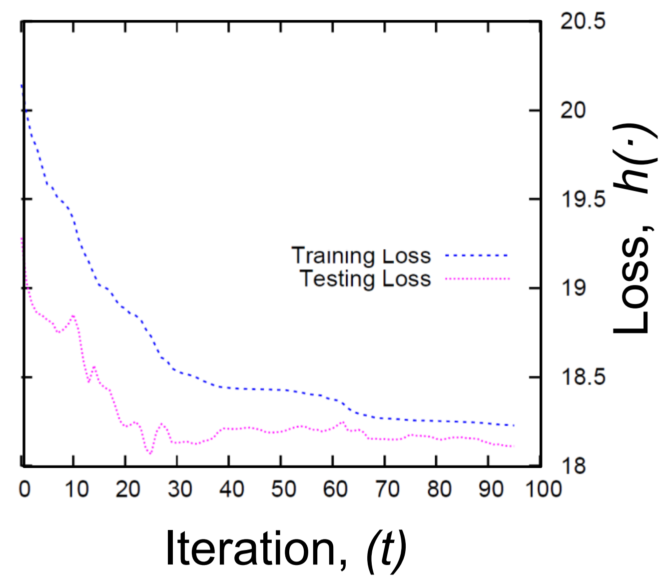
Details!

- **We just got:** $$\frac{\partial p_u}{\partial \beta} = \sum_j Q_{ju} \frac{\partial p_j}{\partial \beta} + p_j \frac{\partial Q_{ju}}{\partial \beta}$$
  - **Few details:**
    - Computing $\partial Q_{ju}/\partial \beta$ is easy. **Remember:** $Q'_{uv} = \begin{cases} \frac{a_{uv}}{\sum_w a_{uw}} & \text{if } (u,v) \in E, \\ 0 & \text{otherwise} \end{cases}$

      $$a_{uv} = f_\beta(u,v)$$
      $$= \exp\left(-\sum_i \beta_i \cdot \mathrm{x}_{uv}[i]\right)$$
    - We want $\frac{\partial p_j}{\partial \beta}$ but it appears on both sides of the equation. Notice the whole thing looks like a PageRank equation: $x = Q \cdot x + z$
- **As with PageRank we can use the power-iteration to solve it:**
  - Start with a random $\frac{\partial p}{\partial \beta}^{(0)}$
  - Then iterate: $\frac{\partial p}{\partial \beta}^{(t+1)} = Q \cdot \frac{\partial p}{\partial \beta}^{(t)} + \frac{\partial Q_{ju}}{\partial \beta} \cdot p$

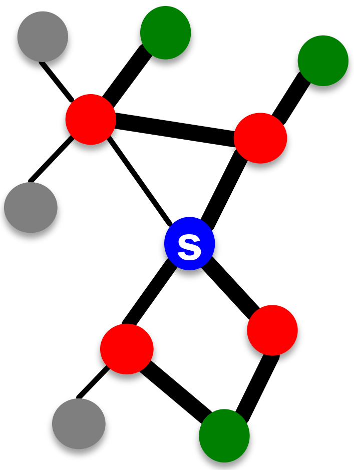
Optimizing $F(\beta)$

- **To optimize $F(\beta)$, use gradient descent:**
  - Pick a random starting point $\beta^{(0)}$
  - Using current $\beta^{(t)}$ compute edge strenghts and the transition matrix Q
  - Compute PageRank scores p
  - Compute the gradient with respect to weight vector $\beta^{(t)}$
  - Update $\beta^{(t+1)}$



Data: Facebook

- **Facebook Iceland network**
  - 174,000 nodes (55% of population)
  - Avg. degree 168
  - Avg. person added 26 friends/month
- **For every node s:**
  - **Positive examples:**
    - $D = \{$ new friendships s created in Nov '09 $\}$
  - **Negative examples:**
    - $L = \{$ other nodes s did not create new links to $\}$
  - **Limit to friends of friends:**
    - On avg. there are 20,000 FoFs (maximum is 2 million)!



Experimental setting

- **Node and Edge features for learning:**
  - **Node:** Age, Gender, Degree
  - **Edge:** Age of an edge, Communication, Profile visits, Co-tagged photos
- **Evaluation:**
  - **Precision at top 20**
    - We produce a list of 20 candidates
      - By taking top 20 nodes x with highest PageRank score p_x
    - Measure to what fraction of these nodes s actually links to

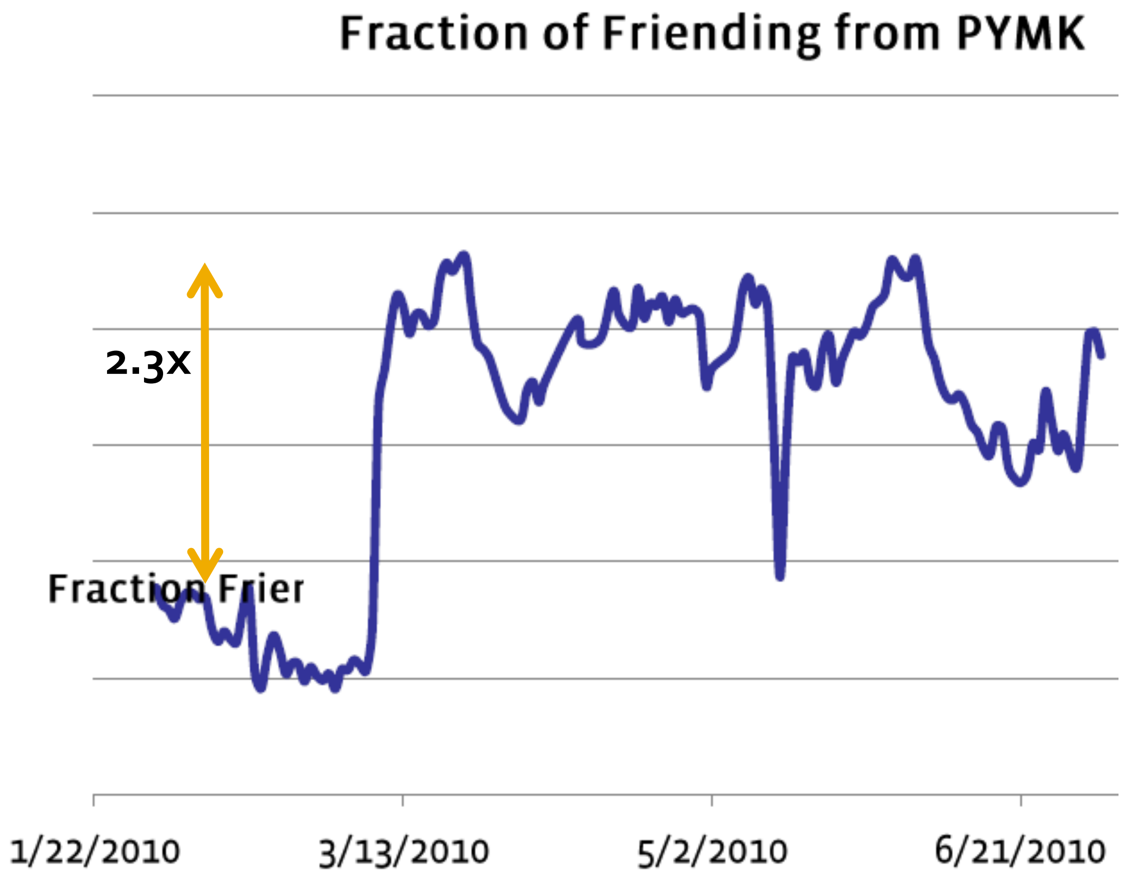
Results: Facebook Iceland

- **Facebook:** Predict future friends
  - Adamic-Adar already works great
  - Supervised Random Walks (SRW) gives slight improvement

Learning Method	Prec@Top20
Random Walk with Restart	6.80
Adamic-Adar	7.35
Common Friends	7.35
Degree	3.25
SRW: one edge type	6.87
SRW: multiple edge types	**7.57**

Results: Facebook

- **2.3x improvement over previous FB-PYMK (People You May Know)**



Results: Co-Authorship

- **Arxiv Hep-Ph collaboration network:**
  - Poor performance of unsupervised methods
  - SRW gives a boost of 25%!

Learning Method	Prec@Top20
Random Walk with Restart	3.41
Adamic-Adar	3.13
Common Friends	3.11
Degree	3.05
SRW: one edge type	4.24
SRW: multiple edge types	**4.25**

Network Inference

CS224W: Social and Information Network Analysis

Jure Leskovec, Stanford University

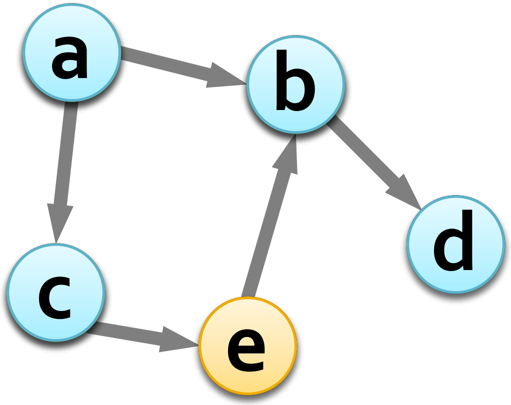
http://cs224w.stanford.edu

Hidden and Implicit Networks

- **Many networks are implicit or hard to observe:**
  - Hidden/hard-to-reach populations:
    - Network of needle sharing between drug injection users
  - Implicit connections:
    - Network of information propagation in online news media
- **But we can observe results of the processes taking place on such (invisible) networks:**
  - **Virus propagation:**
    - Drug users get sick, and we observe when they see the doctor
  - **Information networks:**
    - We observe when media sites mention information
- **Question: Can we infer the hidden networks?**

Inferring the Diffusion Networks

- **There is a hidden diffusion network:**

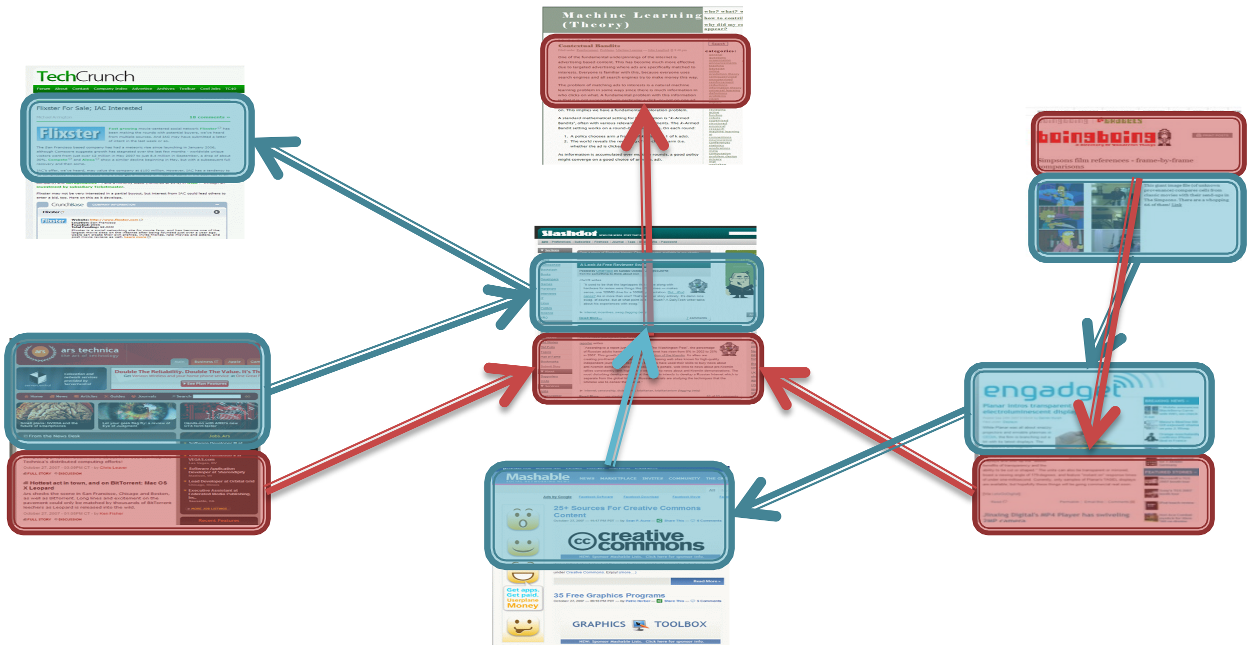


- We only see times when nodes get "infected":
  - Cascade c_1: (a,1), (c,2), (b,3), (e,4)
  - Cascade c_2: (c,1), (a,4), (b,5), (d,6)
- **Want to infer who-infects-whom network!**

[KDD, '09]

Examples and Applications

- **Information diffuses through the blogosphere**



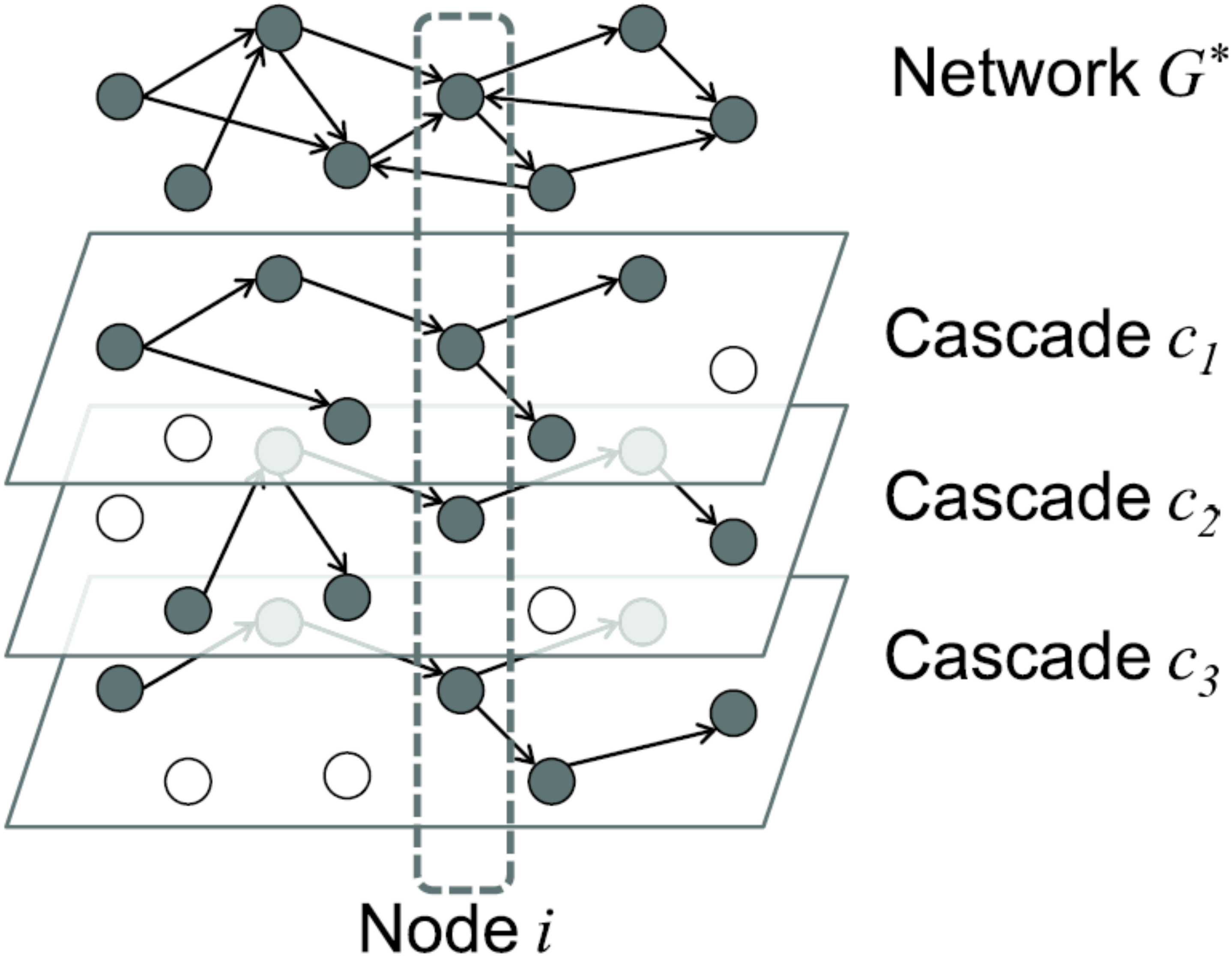
- We only see the mention but not the source
- Can we reconstruct (hidden) **diffusion network**?

Examples and Applications

	Virus propagation	Word of mouth & Viral marketing
Process	Viruses propagate through the network	Recommendations and influence propagate
We observe	We only observe when people get sick	We only observe when people buy products
It's hidden	But NOT who **infected** whom	But NOT who **influenced** whom

Can we infer the underlying network?

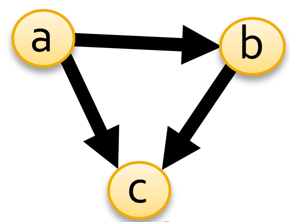
Inferring the Diffusion Network



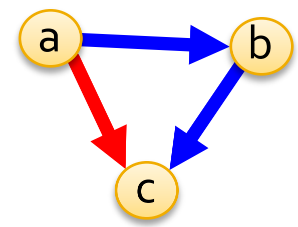
Network Inference: The Task

- **Goal:** Find a graph G that best explains the observed infection times
  - **Given a graph G, define the likelihood P(C|G):**

$P_c(a,b)$: How likely is *a* to infect *b*

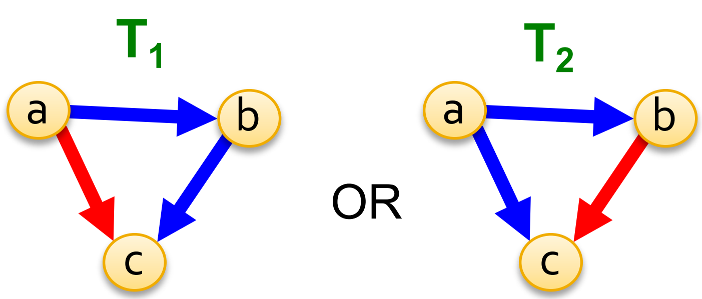


Graph G



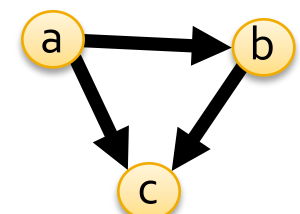
P(c|T): How likely is *c* to propagate via cascade-tree T

Here: T={a → b → c}



P(c|G): How likely is *c* to propagate in graph G

In both T_1,T_2 the order of infections is the same: a,b,c



P(C|G): How likely is a set of $c \in C$ to propagate in G

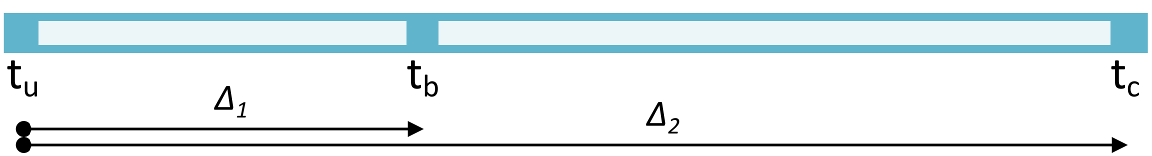
Network Inference: The Task

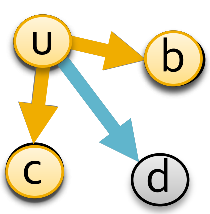
- **Goal: Find a graph G that best explains the observed infection times**
  - **Given a graph G, define the likelihood P(C|G):**

    Define a model of information diffusion over a graph
    - **$P_c(a,b)$** … prob. that *a* infects *b* in contagion *c*
    - **P(c|T)** … prob. that *c* spread in particular cascade-tree *T*
    - **P(c|G)** … prob. that cascade *c* occurred in *G*
    - **P(C|G)** … prob. that a set of cascades *C* occurred in *G*
- **Questions:**
  - How to efficiently compute P(G|C)? (given a single G)
  - How to efficiently find G^* that maximizes P(G|C)? (over $O(2^{N*N})$ graphs)

Cascade Diffusion Model

- **Continuous time cascade diffusion model:**
  - Cascade *c* reaches node *u* at t_u and spreads to *u*'s neighbors:
    - With probability β cascade propagates along edge *(u, v)* and we determine the infection time of node v
      $t_v = t_u + \Delta$
      e.g.: $\Delta \sim$ *Exponential*



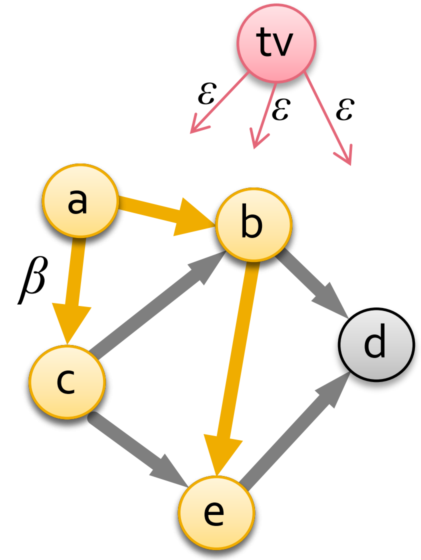


We assume each node v has only one parent!

Cascade Diffusion Model

- **The model for one cascade:**
  - Cascade reaches node u at time t_u, and spreads to u's neighbors v:

    With prob. β cascade propagates along edge (u,v) and $t_v = t_u + \Delta$



- **Transmission probability:**

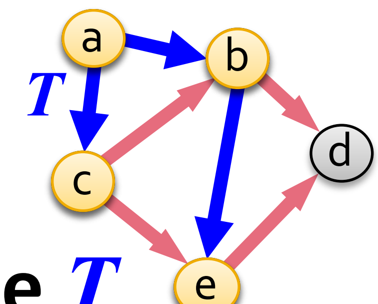
$$P_c(u,v) \propto P(t_v - t_u) \text{ if } t_v > t_u \text{ else } \varepsilon$$

$$e.g.: P_c(u,v) \propto e^{-\Delta t}$$

  - ε captures influence external to the network
    - At any time a node can get infected from outside with small probability ε, equal for all nodes

Cascade Probability

- **Given node infection times & cascade-tree *T*:**
  - $c = \{ (a,1), (c,2), (b,3), (e,4) \}$
  - $T = \{ a \to b, a \to c, b \to e \}$



Graph G

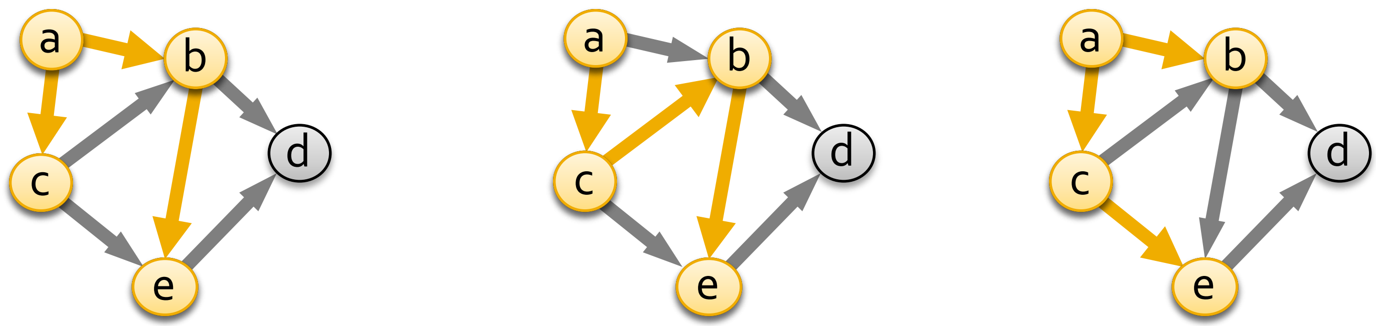
- **Prob. that *c* propagates in cascade-tree *T***

$$P(c|T) = \underbrace{\prod_{(u,v) \in E_T} \beta P_c(u,v)}_{\text{Edges that "propagated"}} \underbrace{\prod_{u \in V_T, (u,x) \in E \setminus E_T} (1-\beta)}_{\text{Edges that failed to "propagate"}}$$

- **Approximate it as:** $P(c|T) \approx \prod_{(u,v) \in E_T} P_c(v,u)$

Complication: Too Many Trees

- **How likely is cascade *c* to spread in graph G?**
  - $c = \{(a,1), (c,2), (b,3), (e,4)\}$



- Need to consider **all possible ways for *c* to spread over *G*** (i.e., all spanning trees *T*):

$$P(c|G) = \sum_{T \in \mathcal{T}_c(G)} P(c|T) \approx \max_{T \in \mathcal{T}_c(G)} P(c|T)$$

Consider only the most likely propagation tree

The Optimization Problem

- **Score of a graph G for a set of cascades C:**

$$P(C|G) = \prod P(c|G)$$

$$F_C(G) = \sum_{c \in C} \log P(c|G)$$

- **Want to find the "best" graph:**

$$G^* = \underset{|G| \leq k}{\operatorname{argmax}} F_C(G)$$

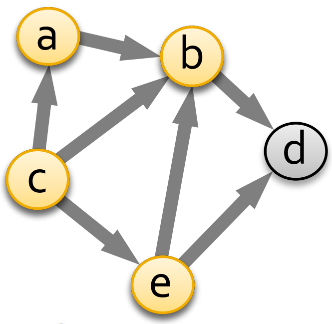
The problem is NP-hard: MAX-k-COVER [KDD '10]

How to Find the Best Tree?

- **Given a cascade c, what is the most likely propagation tree?**

$$\max_{T \in \mathcal{T}_c(G)} P(c|T) = \max_{T \in \mathcal{T}(G)} \sum_{(i,j) \in T} w_c(i,j)$$

  - Maximum **directed** spanning tree
    - Edge *(i,j)* in G has weight $w_c(i,j) = \log P_c(i,j)$
    - The maximum weight spanning tree on infected nodes: Each node picks an in-edge of max weight: $= \sum_{i \in V} \max_{Par_T(i)} w(Par_T(i), i)$



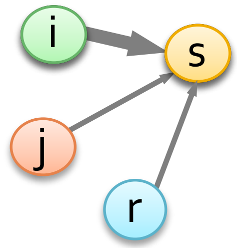
Parent of node *i* in tree *T*

Local greedy selection gives optimal tree!

Great News: Submodularity!

- **Theorem:**
  $F_c(G)$ is monotonic, and submodular
- Proof:
  - Single cascade c, some edge $e=(r,s)$ of weight. w_{rs}
  - Show $F_c(G \cup \{e\}) - F_c(G) \geq F_c(G' \cup \{e\}) - F_c(G')$
  - Let $w_{.s}$ be max weight in-edge of s in G
  - Let $w'_{.s}$ be max weight in-edge of s in G'
  - Since $G \subseteq G'$: $w_{.s} \leq w'_{.s}$ and $w_{rs} = w'_{rs}$
  - $F_c(G \cup \{(r,s)\}) - F_c(G)$

$$
\begin{aligned}
&= \max(w_{.s}, w_{rs}) - w_{.s} \\
&\geq \max(w'_{.s}, w_{rs}) - w'_{.s} \\
&= F_c(G' \cup \{(r,s)\}) - F_c(G')
\end{aligned}
$$



s picks in-edge of max weight

NetInf: The Algorithm

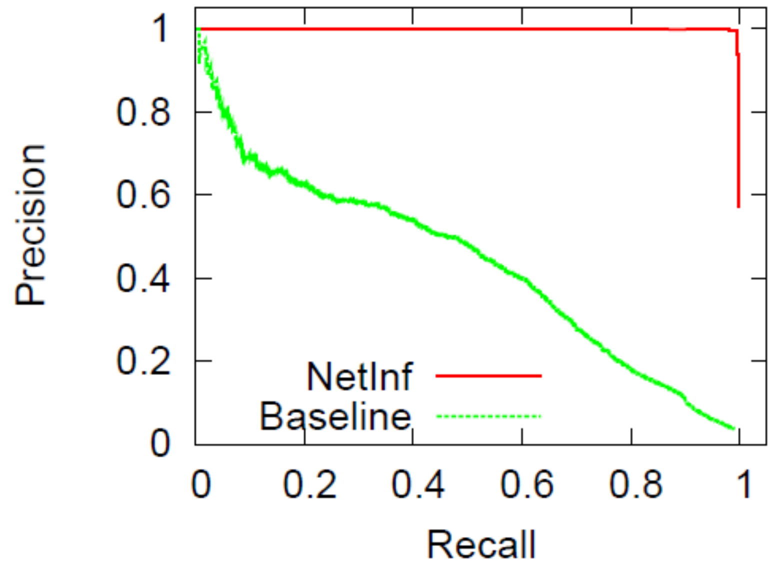
- **The NetInf algorithm:**
  Use **greedy hill-climbing** to maximize $F_C(G)$:
  - Start with empty G_0 (G with no edges)
  - Add k edges (k is parameter)
  - At every step i add an edge to the graph G_i that maximizes the marginal improvement

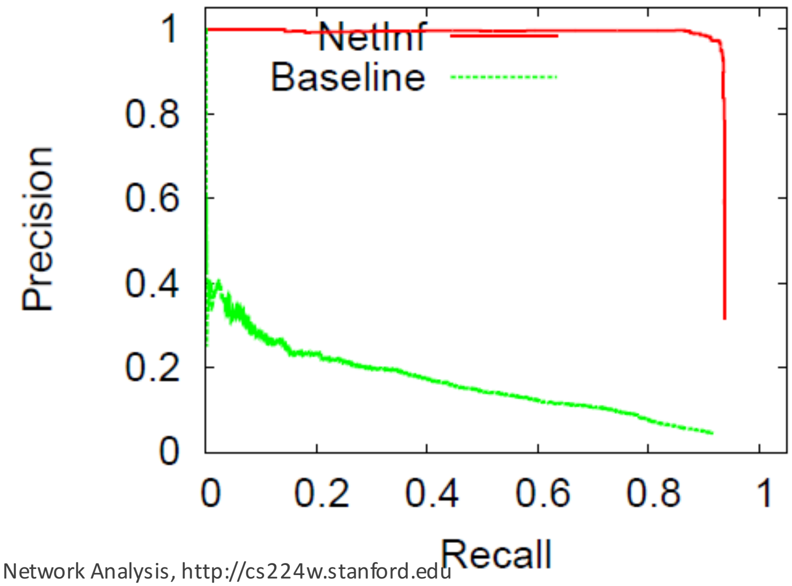
$$e_i = \operatorname*{argmax}_{e \in G \setminus G_{i-1}} F_C(G_{i-1} \cup \{e\}) - F_C(G_{i-1})$$

Note: This is the same algorithm we used for influence maximization

Experiments: Synthetic data

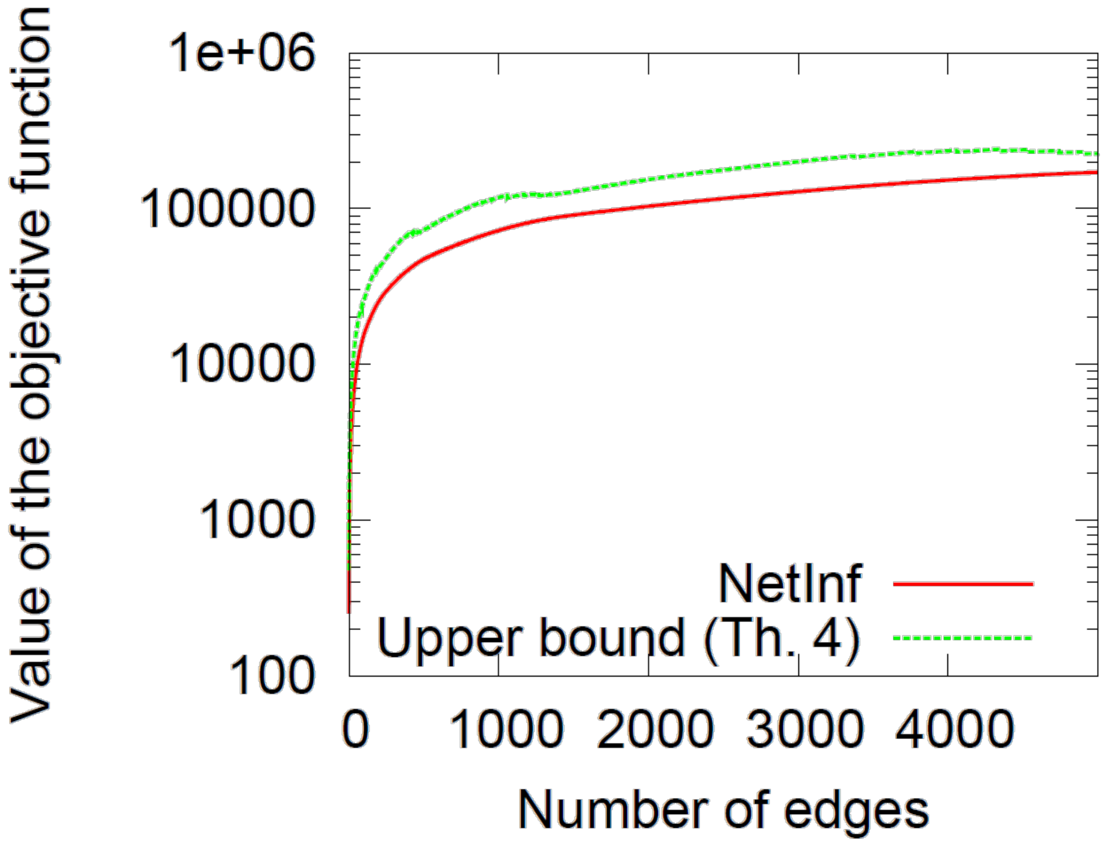
- **Synthetic data:**
  - Take a graph G on *k* edges
  - Simulate info. diffusion
  - Record node infection times
  - Reconstruct G
- **Evaluation:**
  - How many edges of G can NetInf find?
    - Break-even point (precision=recall): 0.95
    - Performance is independent of the structure of G!





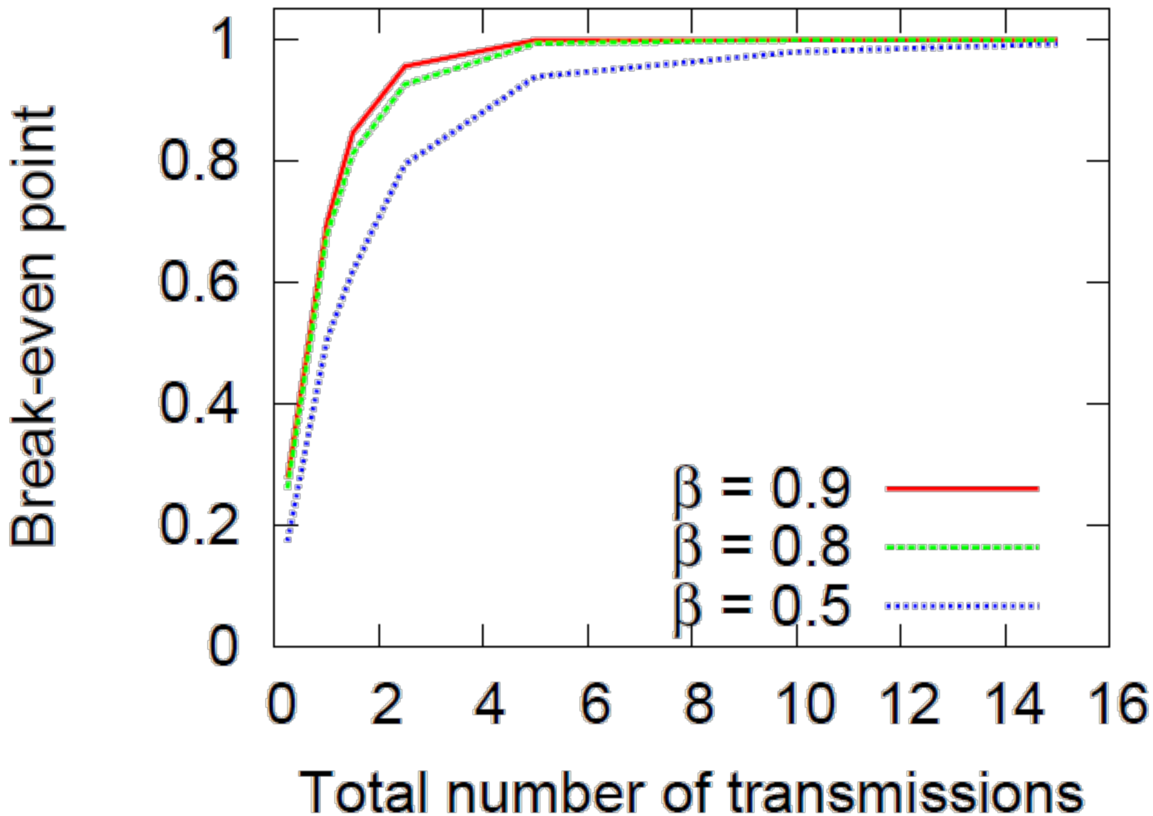
How Good is Our Graph?

- **NetInf achieves ≈ 90 % of the best possible network!**



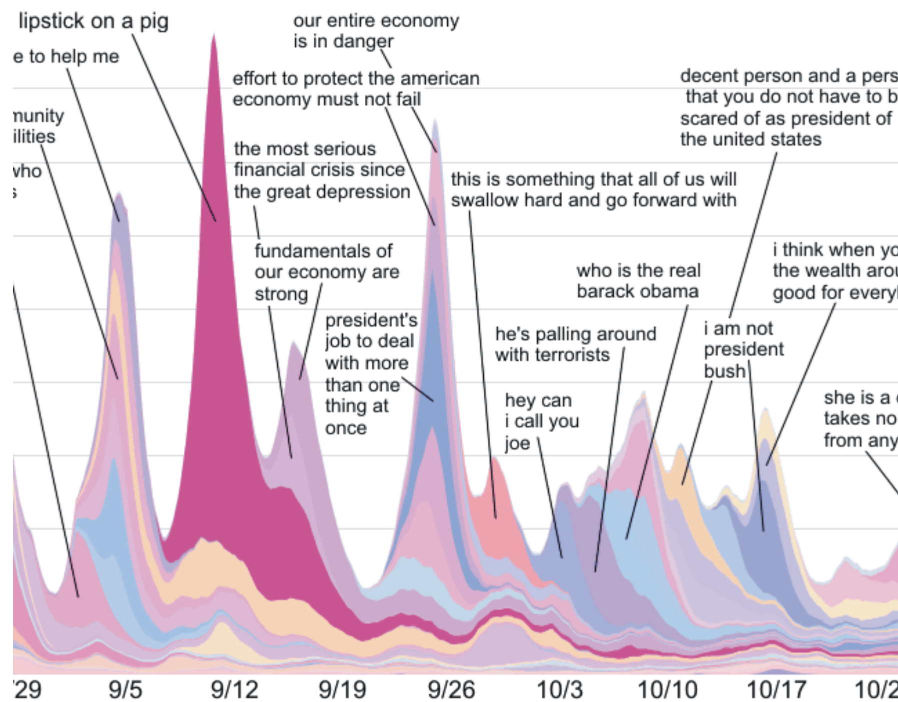
How Many Cascades Do We Need?

- **With 2x as many infections as edges, the break-even point is already 0.8 - 0.9!**



Experiments: Real data

- **Memetracker dataset:**
  - 172m news articles
  - Aug '08 – Sept '09
  - 343m textual phrases
  - Times $t_c(w)$ when site w mentions phrase c

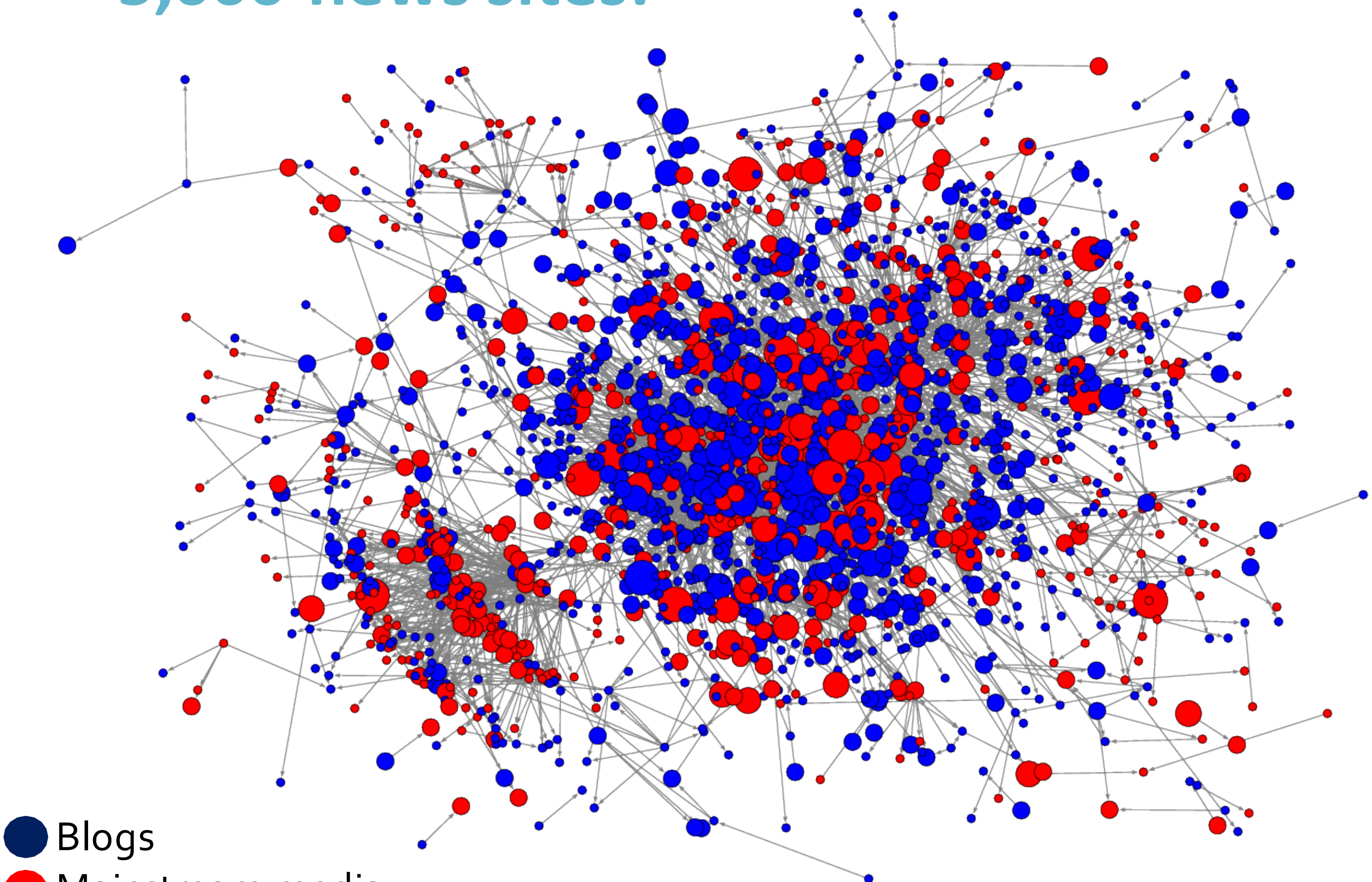


http://memetracker.org

- Given times when sites mention phrases
- Infer the network of information diffusion:
  - Who tends to copy (repeat after) whom

Example: Diffusion Network

- 5,000 news sites:



Diffusion Network (small part)