

Link Prediction and Network Inference

CS224W: Social and Information Network Analysis
Jure Leskovec, Stanford University

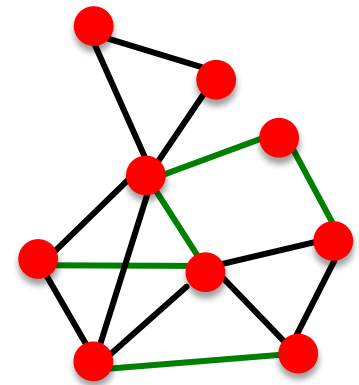
<http://cs224w.stanford.edu>



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 $G[t_1, t_1']$



■ 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

Link Prediction via Proximity

- Predict links in a evolving collaboration network

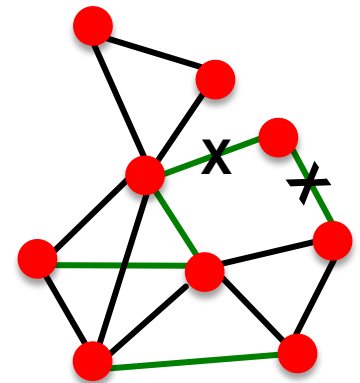
	training period			Core		
	authors	papers	collaborations ¹	authors	$ E_{old} $	$ E_{new} $
astro-ph	5343	5816	41852	1561	6178	5751
cond-mat	5469	6700	19881	1253	1899	1150
gr-qc	2122	3287	5724	486	519	400
hep-ph	5414	10254	47806	1790	6654	3294
hep-th	5241	9498	15842	1438	2311	1576

- **Core:** Since network data is very sparse
 - Consider only nodes with in-degree and out-degree of at least 3

Link Prediction via Proximity

■ Methodology:

- For every pair of nodes (x,y) compute proximity c
 - # of common neighbors $c(x,y)$ of x and y
- Sort pairs by the decreasing score
- $E_{new}^* := E_{new} \cap (\text{Core} \times \text{Core})$
 - (only consider/predict edges where both endpoints are in the core)
- Predict top n pairs as new links



Link Prediction via Proximity

- For every pair of nodes (x,y) compute:

graph distance	(negated) length of shortest path between x and y
common neighbors	$ \Gamma(x) \cap \Gamma(y) $
Jaccard's coefficient	$\frac{ \Gamma(x) \cap \Gamma(y) }{ \Gamma(x) \cup \Gamma(y) }$
Adamic/Adar	$\sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\log \Gamma(z) }$
preferential attachment	$ \Gamma(x) \cdot \Gamma(y) $
Katz $_{\beta}$	$\sum_{\ell=1}^{\infty} \beta^{\ell} \cdot \text{paths}_{x,y}^{\langle \ell \rangle} $

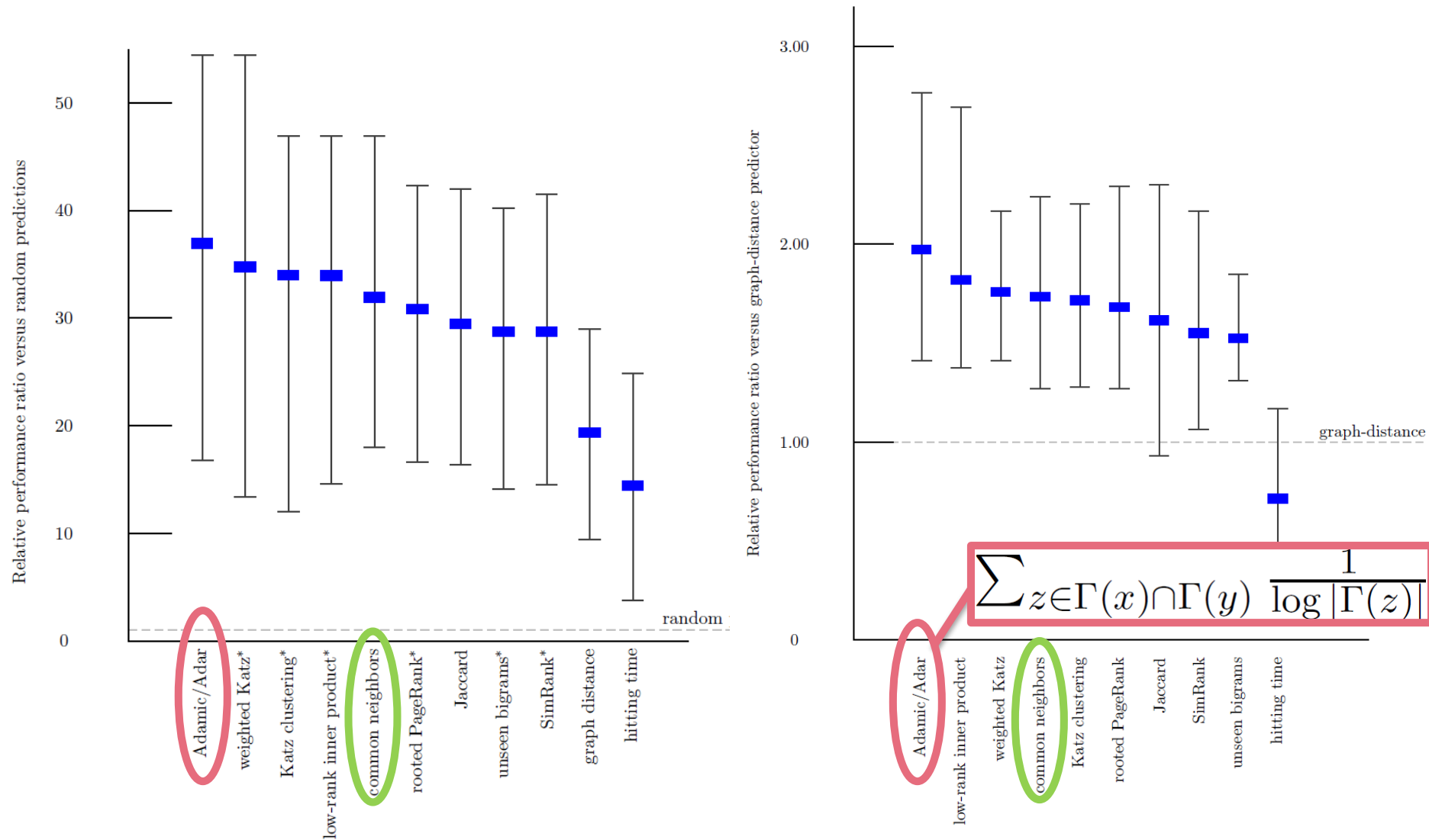
where $\text{paths}_{x,y}^{\langle \ell \rangle} := \{\text{paths of length exactly } \ell \text{ from } x \text{ to } y\}$

weighted: $\text{paths}_{x,y}^{\langle 1 \rangle} := \text{number of collaborations between } x, y.$

unweighted: $\text{paths}_{x,y}^{\langle 1 \rangle} := 1$ iff x and y collaborate.

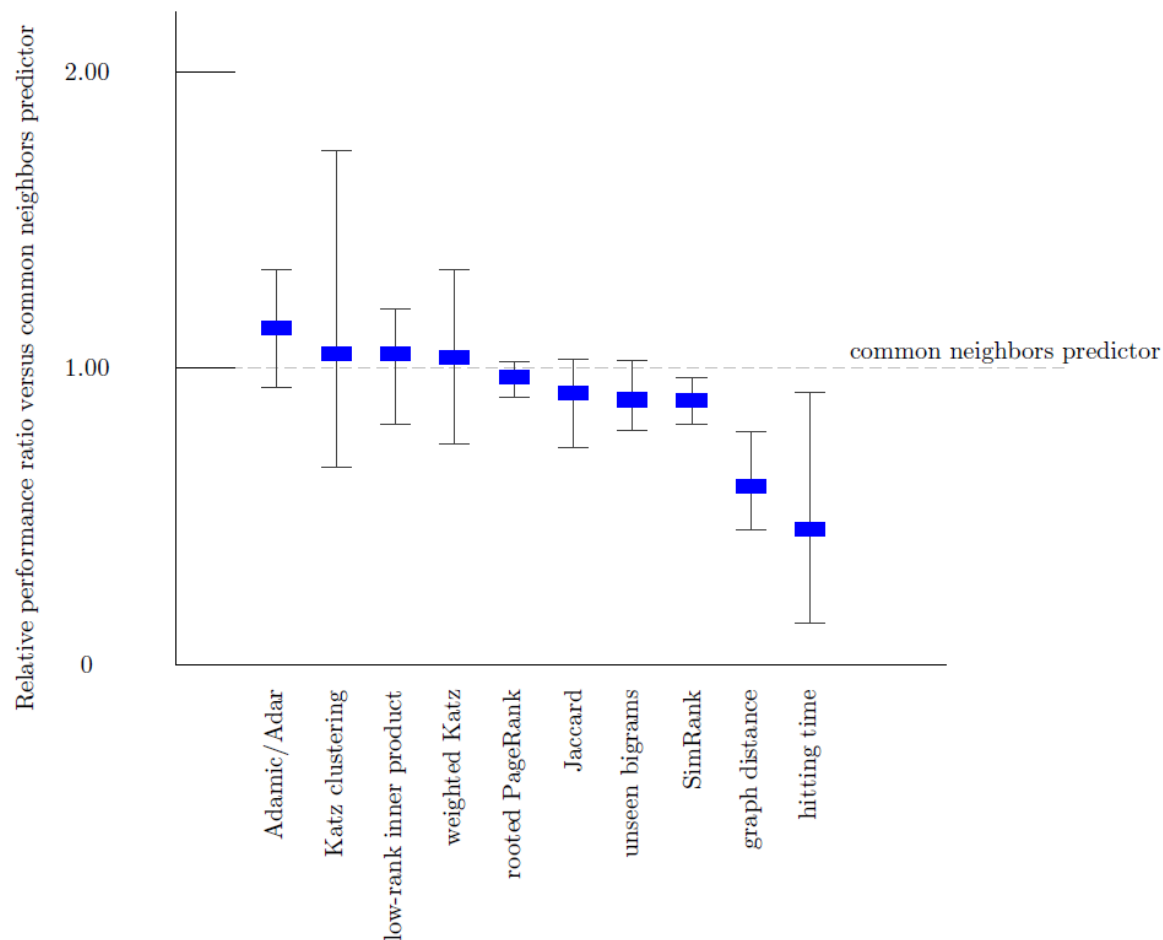
$\Gamma(x)$... degree of node x

Results: Improvement over Random



Results: Common Neighbors

■ Improvement over #common neighbors



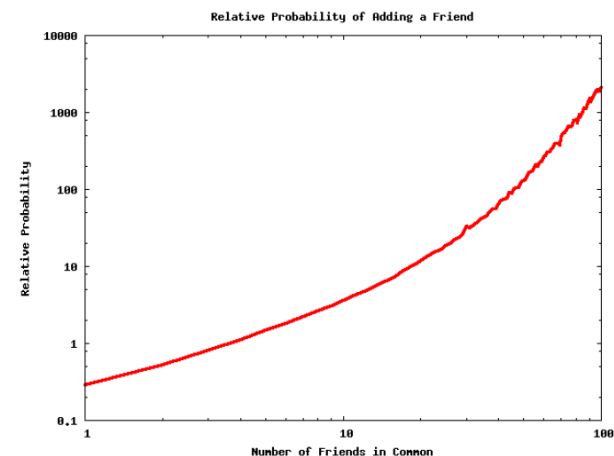
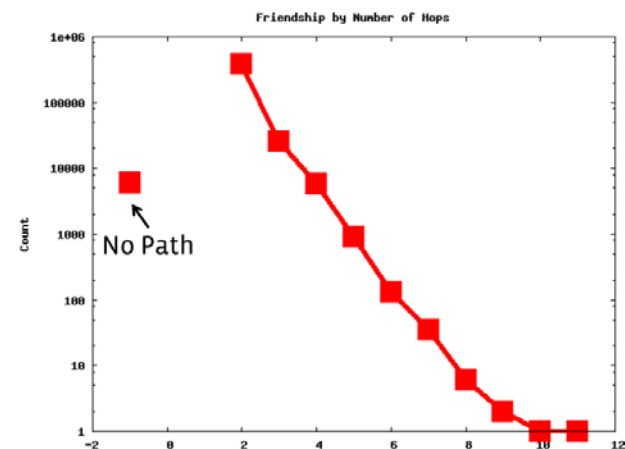
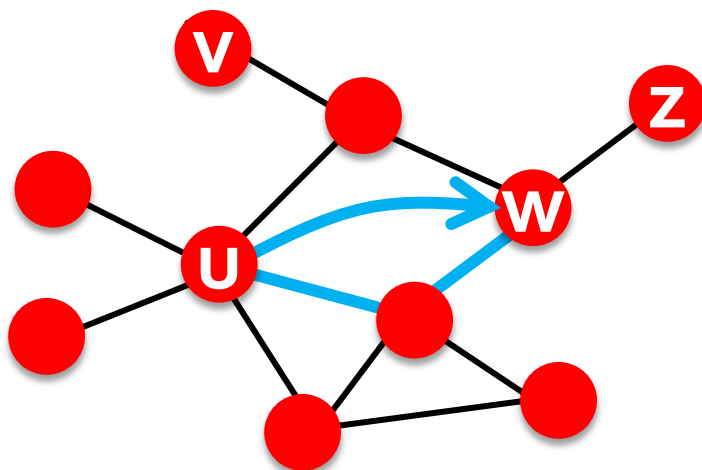
Supervised Link Prediction

■ How to learn to predict new friends?

■ Facebook's People You May Know

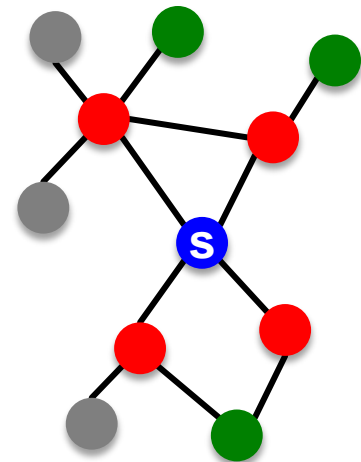
■ Let's look at the data:

- 92% of new friendships on FB are friend-of-a-friend
- More common friends helps



Supervised Link Prediction

- Recommend a list of possible friends
- Supervised machine learning setting:
 - Training example:
 - For every node s have a list of nodes she will create links to $\{v_1 \dots v_k\}$
 - Use FB network from May 2011 and $\{v_1 \dots v_k\}$ are the new friendships you created since then
 - Task:
 - For a given node s **rank** nodes $\{v_1 \dots v_k\}$ **higher** than other nodes in the network
- Supervised Random Walks based on work by Agarwal&Chakrabarti



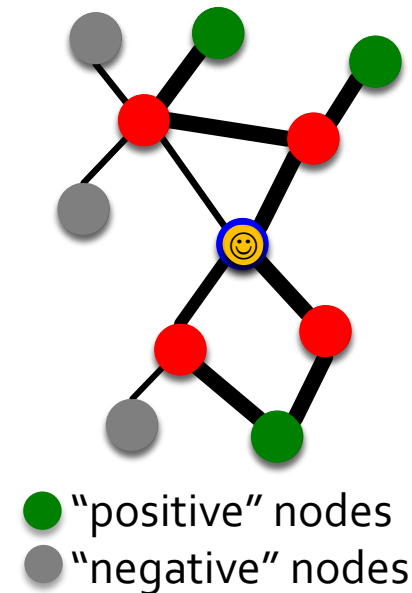
● "positive" nodes
● "negative" nodes

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

Supervised Link Prediction

■ How to combine node/edge attributes and the network structure?

- Learn a **strength** of each edge based on:
 - Profile of user u , profile of user v
 - Interaction history of u and v
- Do a **PageRank-like random walk** from s to measure the “**proximity**” between s and other nodes
- Rank nodes by their “**proximity**” (i.e., visiting prob.)

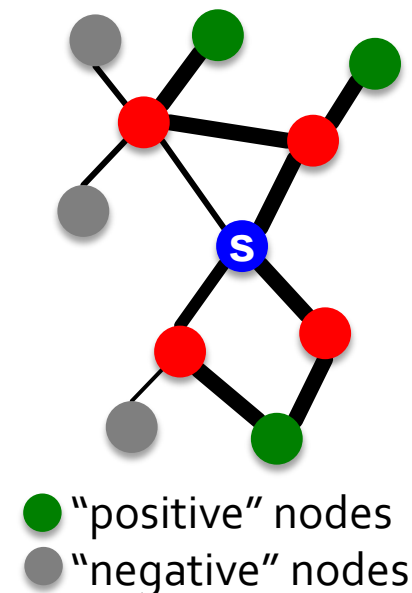


Supervised Random Walks

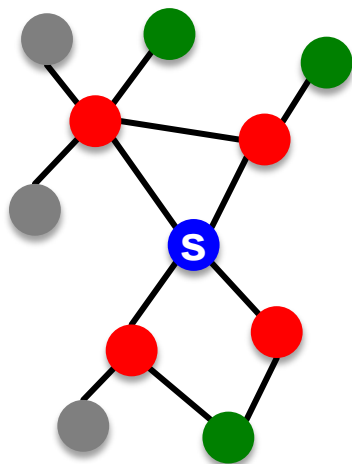
- Let s be the center node
- Let $f_w(u, v)$ be a function that assigns a **strength to each edge**:

$$a_{uv} = f_{\beta}(u, v) = \exp(-\beta^T \Psi_{uv})$$

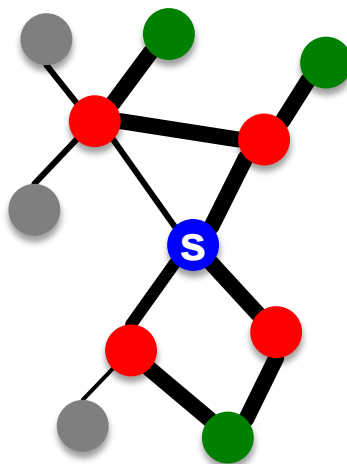
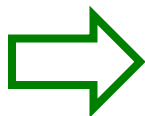
- Ψ_{uv} is a feature vector
 - Features of node u
 - Features of node v
 - Features of edge (u, v)
- (β is the parameter vector we want to learn!)
- Do Random Walk with Restarts from s where transitions are according to edge strengths a_{uv}



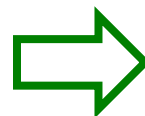
SRW: Prediction



Network



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



*Personalized
PageRank* on the
weighted graph.
Each node u gets a
PageRank proximity p_u



Sort nodes by the
decreasing PageRank
proximity p_u



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

- How to estimate edge strengths?
 - How to set parameters β of $f_{\beta}(u, v)$?

Personalized PageRank

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

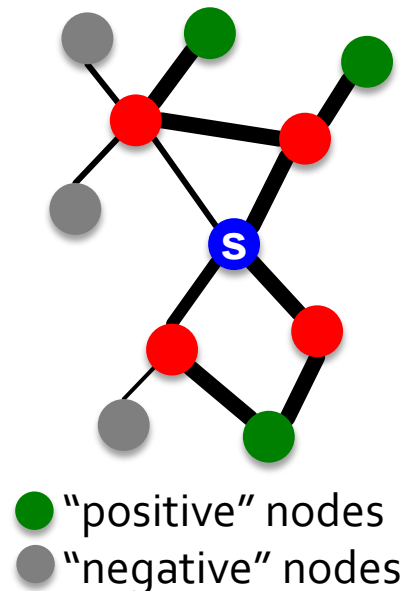
$$Q'_{uv} = \begin{cases} \frac{a_{uv}}{\sum_w a_{uw}} & \text{if } (u,v) \in E, \\ 0 & \text{otherwise} \end{cases}$$

- PageRank transition matrix:

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

- with prob. α jump back to s

- Compute PageRank vector: $p = p^T Q$
- Rank nodes u by p_u



The Optimization Problem

- Each node u has a score p_u
- Positive nodes $D = \{d_1, \dots, d_k\}$
- Negative nodes $L = \{\text{the rest}\}$
- What do we want?

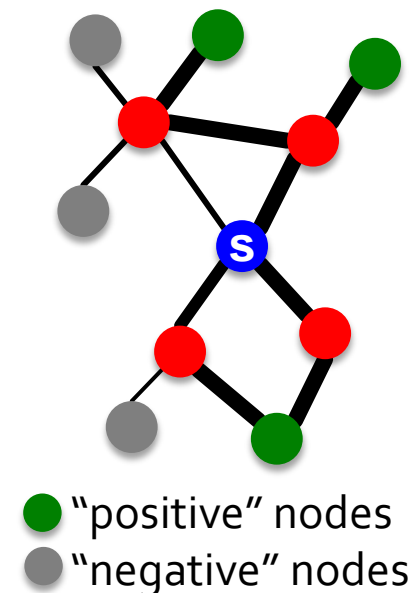
Want to find β such that $p_l < p_d$

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

such that

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

- The exact solution to the above problem may not exist
- So we make the constraints “soft” (i.e., optional)

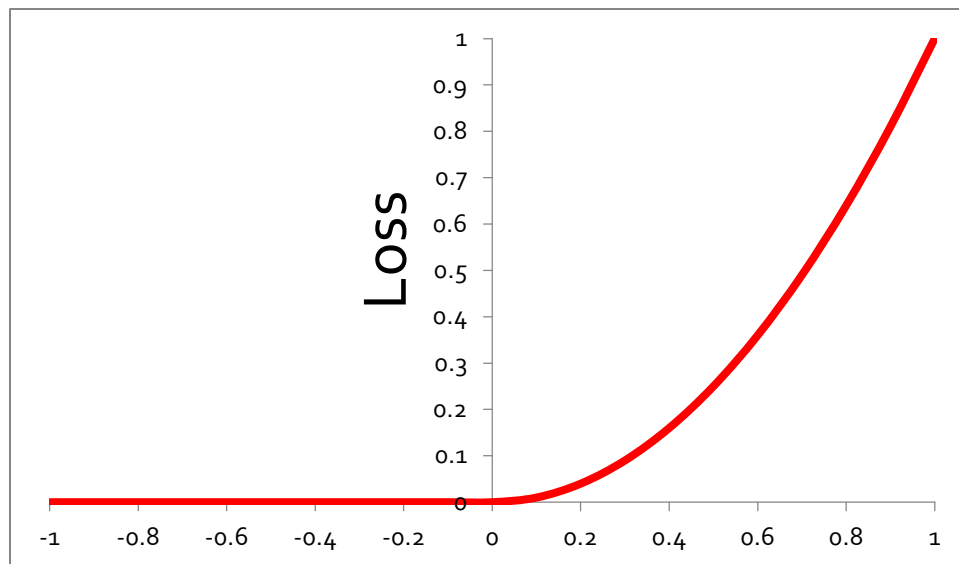


Making Constraints “Soft”

- Want to minimize:

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

- **Loss:** $h(x) = 0$ if $x < 0$, x^2 else $h(x) = \max\{x, 0\}^2$



$p_l < p_d$

$p_l = p_d$

$p_l > p_d$

Solving the Problem: Intuition

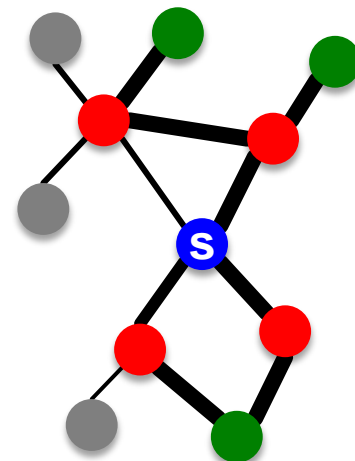
■ How to minimize F ?

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

■ Both p_l and p_d depend on β

- Given β assign edge weights $a_{uv} = f_{\beta}(u, v)$
- Using transition matrix $Q = [a_{uv}]$ compute PageRank scores p_u
- Rank nodes by the PageRank score

■ Want to find β such that $p_l < p_d$



Gradient Descent

- How to minimize F ?

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

- Take the derivative!

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

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

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

Easy

- We know:

$$p = p^T Q \quad i.e. \quad p_u = \sum_j p_j Q_{ju}$$

- So:

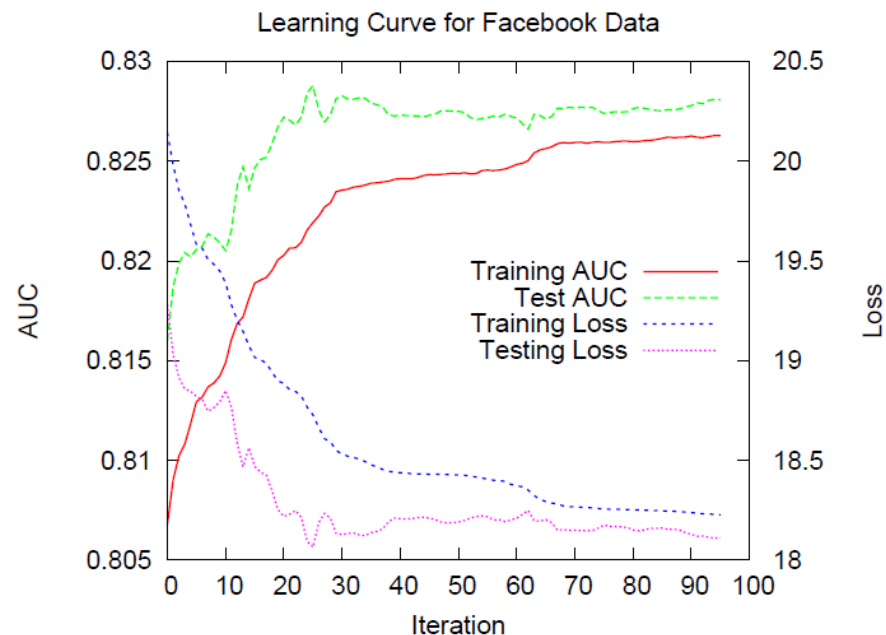
$$\frac{\partial p_u}{\partial \beta} = \sum_j Q_{ju} \frac{\partial p_j}{\partial \beta} + p_j \frac{\partial Q_{ju}}{\partial \beta}$$

- Looks like the PageRank equation!

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

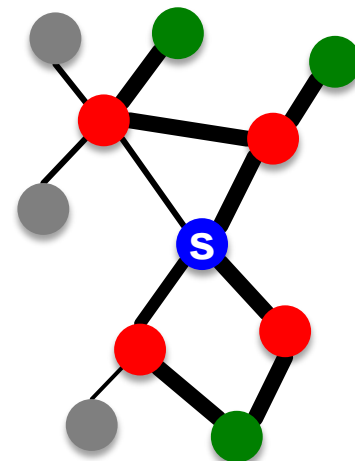
Optimizing F

- To optimize F, use gradient based method:
 - Pick a random starting point β_0
 - Compute the personalized PageRank vector p
 - Compute the gradient with respect to the weight vector β
 - Update β
 - Optimize using quasi-Newton method



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:**
 - $D = \{ \text{new friendships of } s \text{ created in Nov '09} \}$
 - **Negative examples:**
 - $L = \{ \text{other nodes } s \text{ did not create new links to} \}$
 - **Limit to friends of friends:**
 - on avg. there are 20k FoFs (max 2M)!



Experimental setting

- **Node and Edge features for learning:**
 - **Node:** Age, Gender, Degree
 - **Edge:** Age of an edge, Communication, Profile visits, Co-tagged photos
- **Baselines:**
 - Decision trees and logistic regression:
 - Above features + 10 network features (PageRank, common friends)
- **Evaluation:**
 - AUC and Precision at Top20

Results: Facebook Iceland

- Facebook:
predict future friends
 - Adamic-Adar already works great
 - Logistic regression also strong
 - SRW gives slight improvement

Learning Method	AUC	Prec@20
Random Walk with Restart	0.81725	6.80
Adamic-Adar	0.81586	7.35
Common Friends	0.80054	7.35
Degree	0.58535	3.25
DT: Node features	0.59248	2.38
DT: Network features	0.76979	5.38
DT: Node+Network	0.76217	5.86
DT: Path features	0.62836	2.46
DT: All features	0.72986	5.34
LR: Node features	0.54134	1.38
LR: Network features	0.80560	7.56
LR: Node+Network	0.80280	7.56
LR: Path features	0.51418	0.74
LR: All features	0.81681	7.52
SRW: one edge type	0.82502	6.87
SRW: multiple edge types	0.82799	7.57

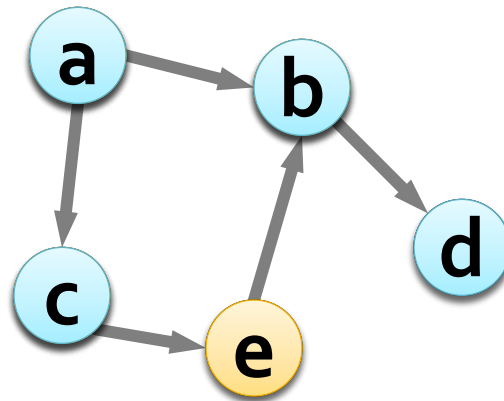
Network Inference

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!**

Examples and Applications

Virus propagation

Process

Viruses propagate through the network

We observe

We only observe when people get sick

It's hidden

But NOT who **infected** whom

Word of mouth & Viral marketing

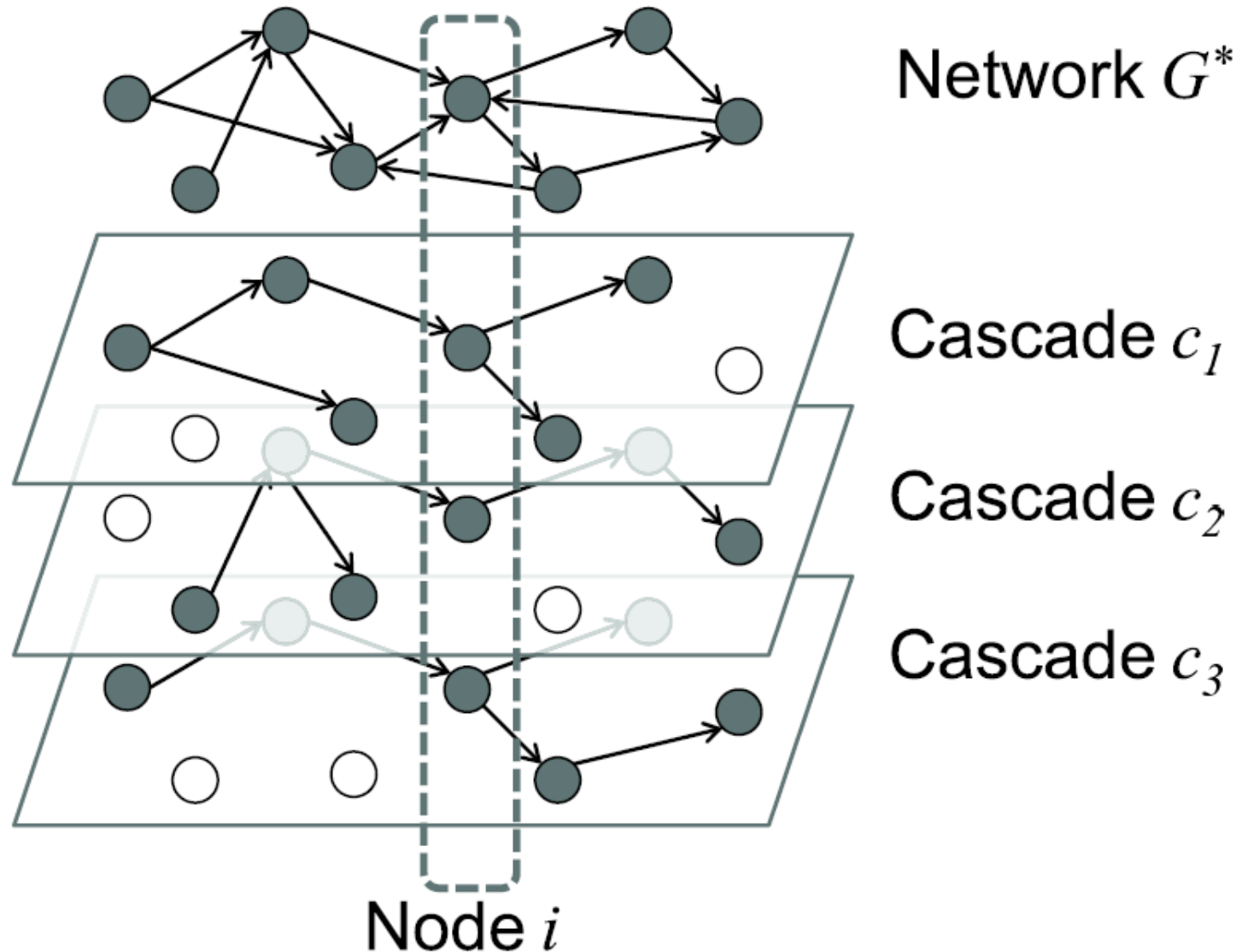
Recommendations and influence propagate

We only observe when people buy products

But NOT who **influenced** whom

Can we infer the underlying network?

Inferring the Diffusion Network

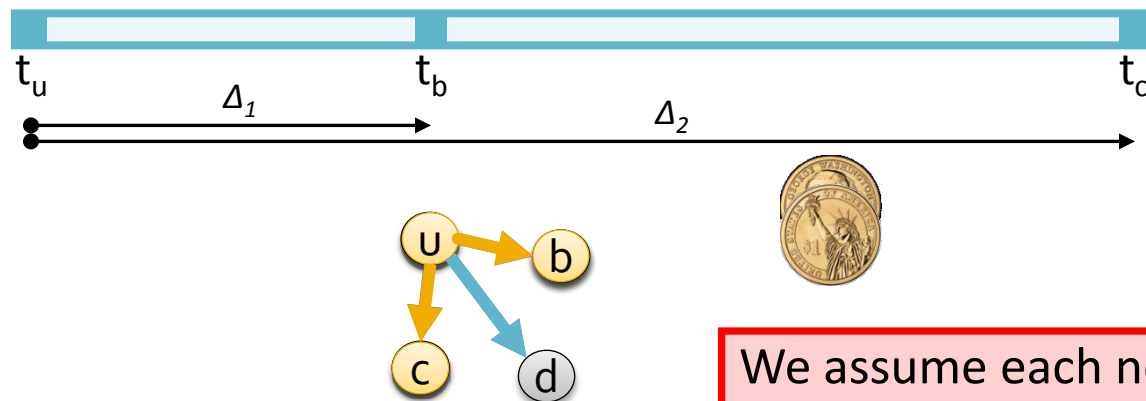


Network Inference: The Task

- **Goal:** Find a graph G that best explains the observed information times
 - Given a graph G , define the likelihood $P(C|G)$:
 - Define a model of information diffusion over a graph
 - $P_c(u,v)$... prob. that u infects v in cascade c
 - $P(c|T)$... prob. that c spread in particular pattern T
 - $P(c|G)$... prob. that cascade c occurred in G
 - $P(G|C)$... 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 $O(2^{N \times N})$ 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.: $\Delta \sim \text{Exponential or Power-law}$



We assume each node v has only one parent!

Cascade Diffusion Model

- **The model for 1 cascade:**

- Cascade reaches node u at time t_u , and spreads to u 's neighbors v :

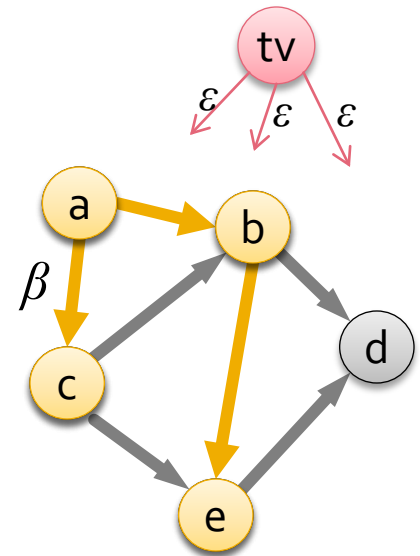
With prob. β cascade propagates along edge (u,v) and $t_v = t_u + \Delta$

- **Transmission probability:**

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

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

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



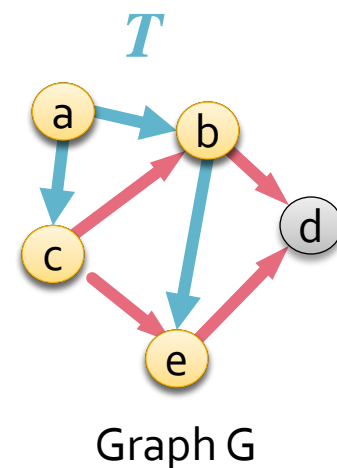
Cascade Probability

- Given node infection times and pattern T :

- $c = \{ (a,1), (c,2), (b,3), (e,4) \}$
- $T = \{ a \rightarrow b, a \rightarrow c, b \rightarrow e \}$

- Prob. that c propagates in pattern T

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

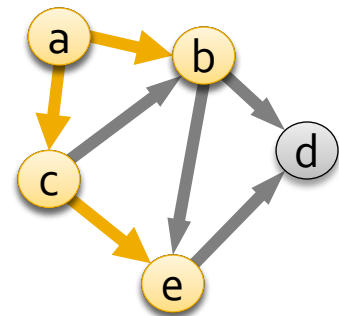
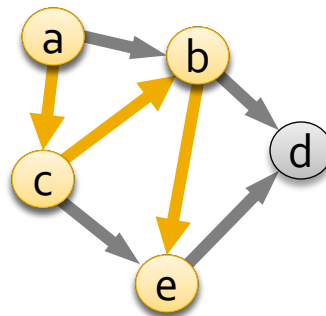
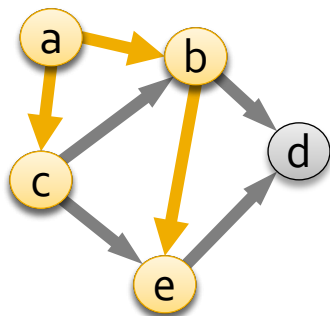


- Approximate it as: $P(c|T) \approx \prod_{(u,v) \in E_T} P_c(v,u)$

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:

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

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

How to Find the Best Tree?

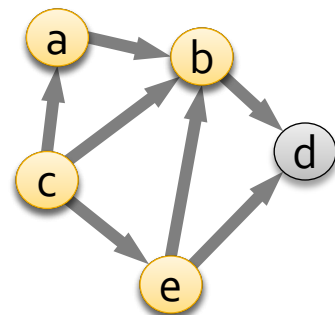
- Given a cascade c , what is the most likely propagation tree?

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

- A 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:

$$\text{max weight} = \sum_{i \in V} \max_{Par_T(i)} w(Par_T(i), i)$$



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}

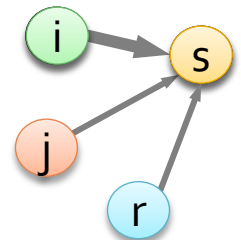
- Show $F_c(G \cup \{e\}) - F_c(G) \geq F_c(G' \cup \{e\}) - F_c(G')$

- 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 $w_{rs} = w'_{rs}$

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



s picks in-edge of max weight

NetInf: The Algorithm

■ The 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 G_i that **maximizes the marginal improvement**

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

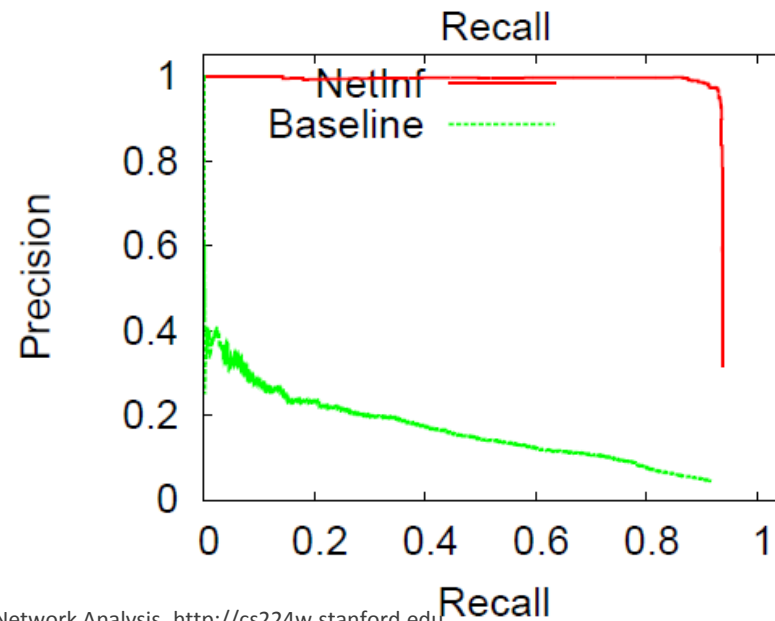
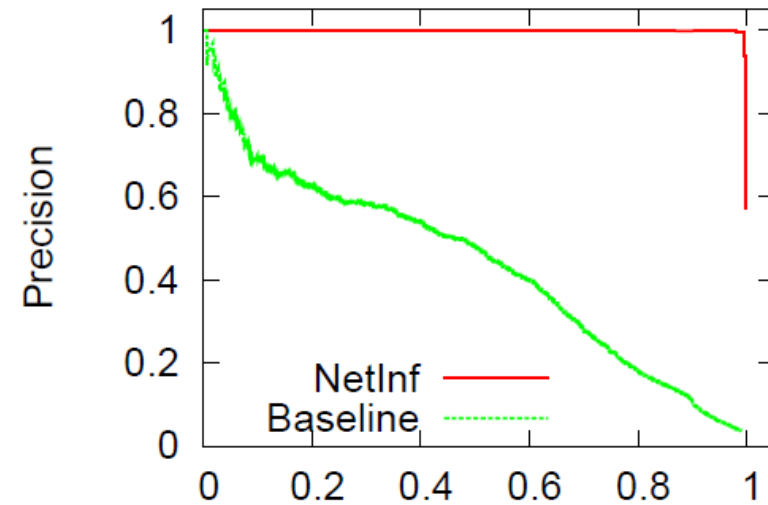
Experiments: Synthetic data

■ Synthetic data:

- Take a graph G on k edges
- Simulate info. diffusion
- Record node infection times
- Reconstruct G

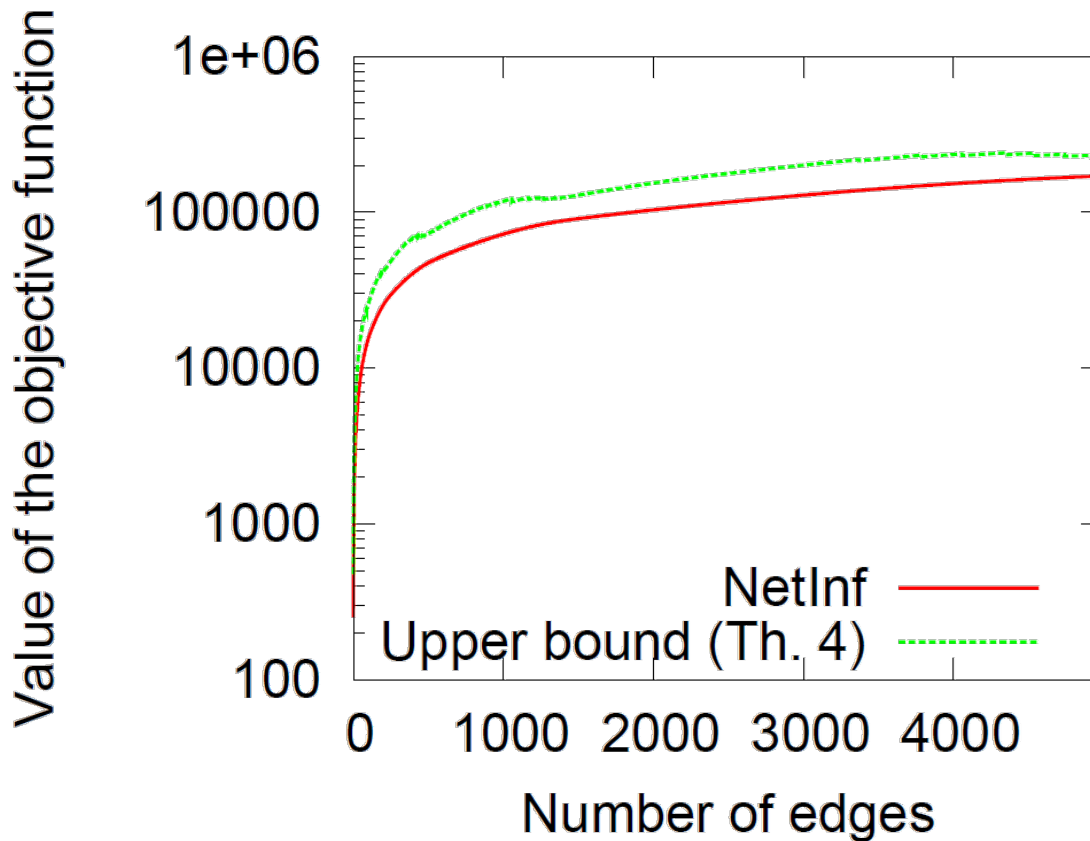
■ Evaluation:

- How many edges of G can NetInf find?
 - Break-even point: 0.95
 - Performance is independent of the structure of G !



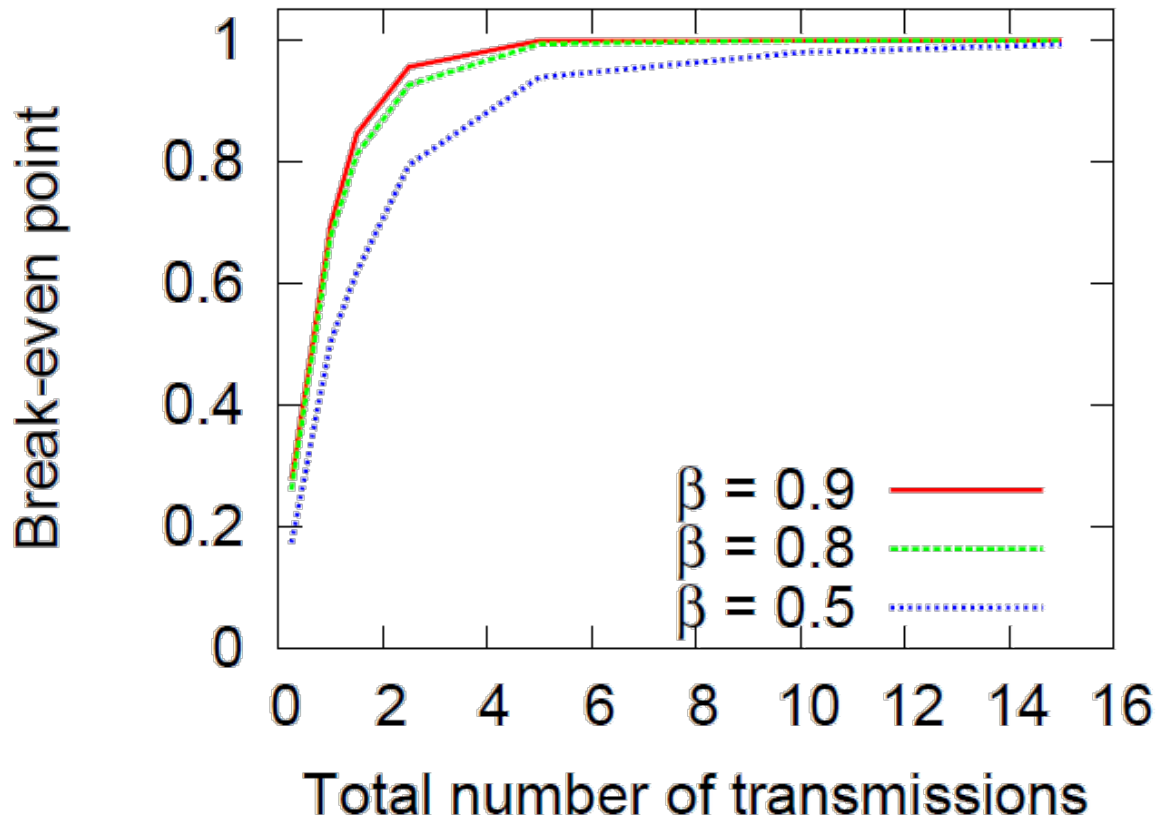
How Good is Our Graph?

- We achieve $\approx 90\%$ of the best possible network!



How Many Cascades Do We Need?

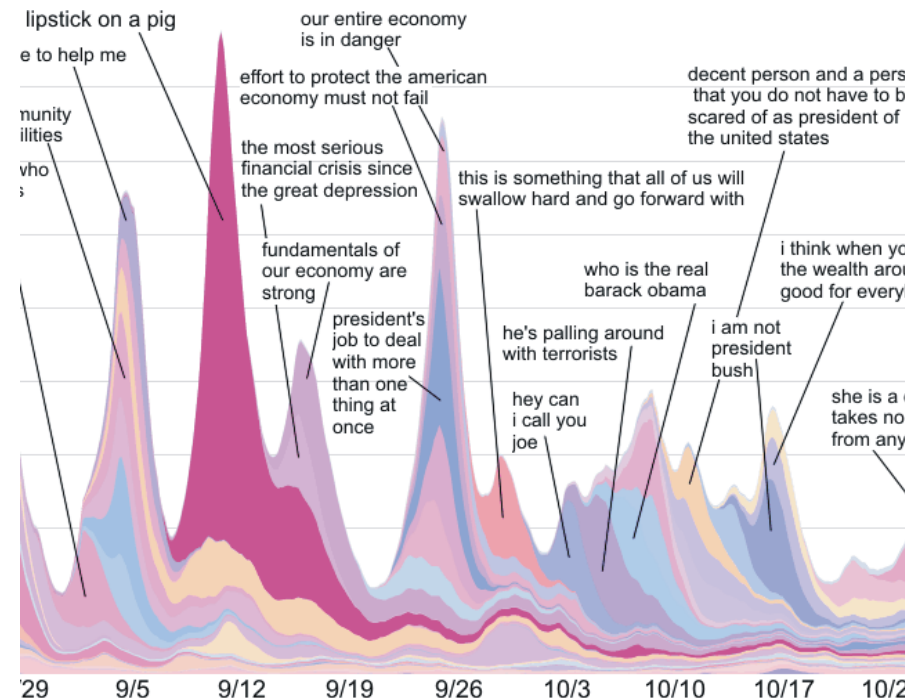
- With 2x as many infections as edges, the break-even point is already 0.8 - 0.9!



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

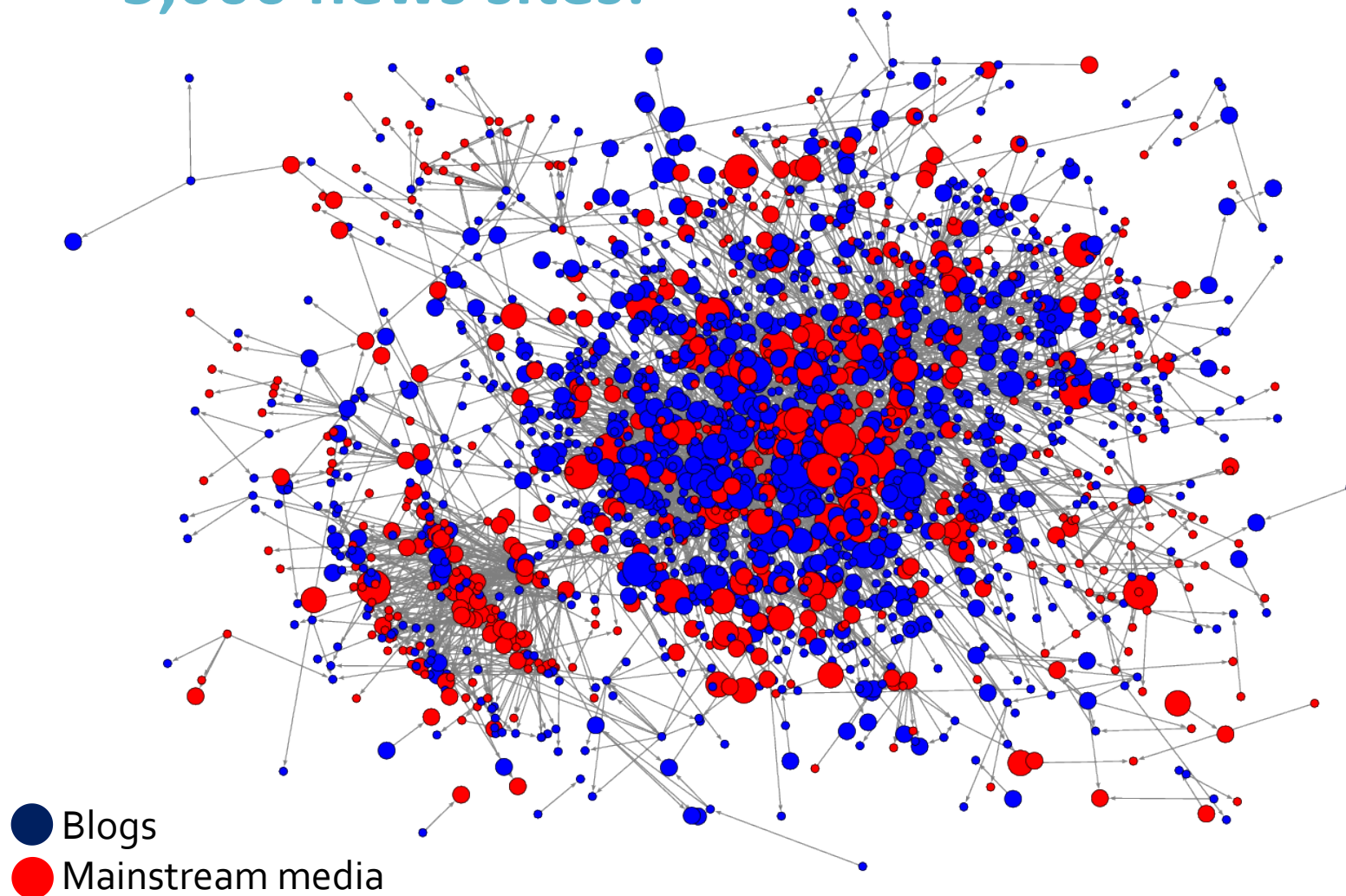


<http://memetracker.org>

- 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)

