Outbreak Detection in Networks

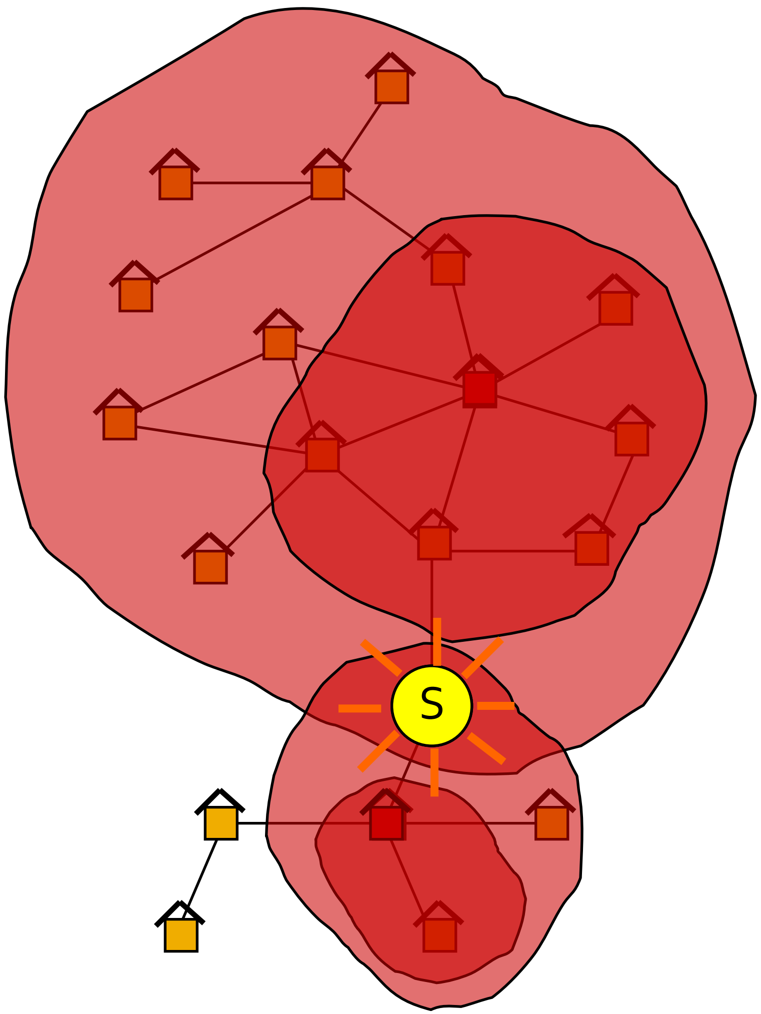
CS224W: Social and Information Network Analysis
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

Plan for Today

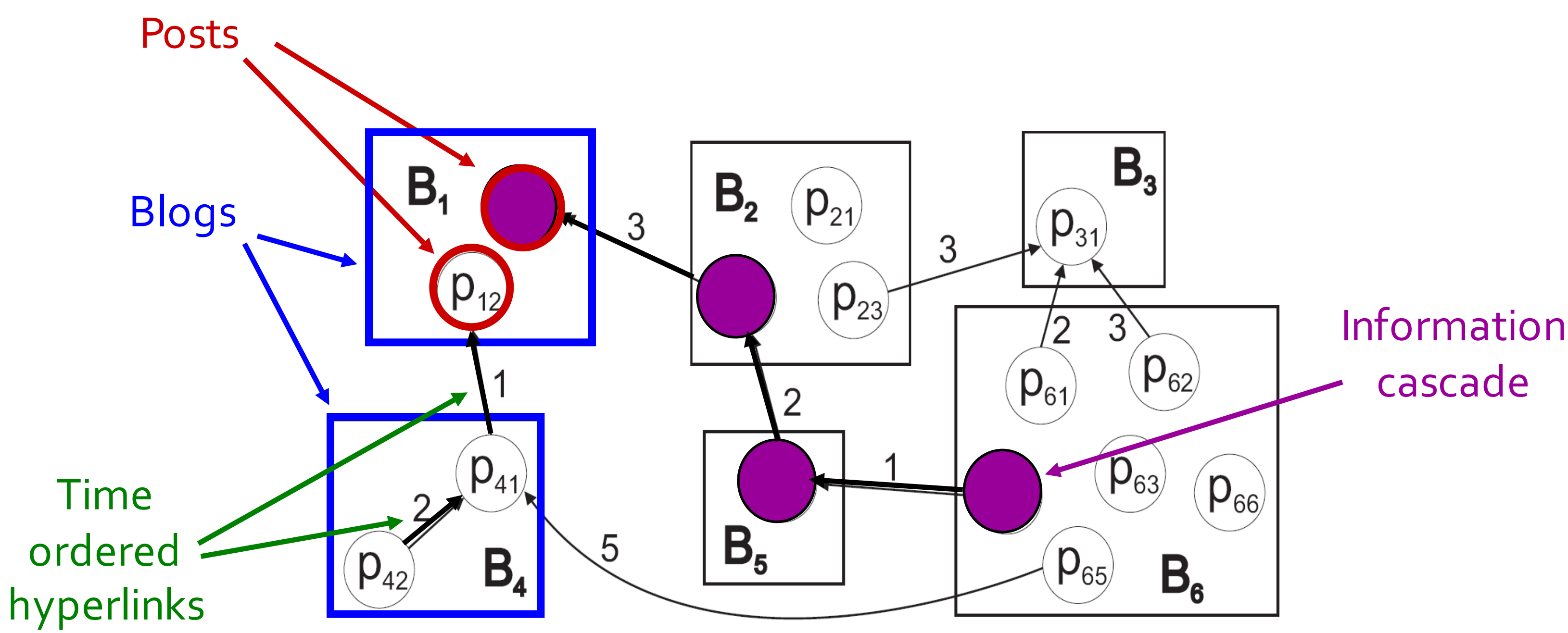
- **(1) New problem: Outbreak detection**
- **(2) Develop an approximation algorithm**
  - It is a submodular opt. problem!
- **(3) Speed-up greedy hill-climbing**
  - Valid for optimizing general submodular functions (i.e., also works for influence maximization)
- **(4) Prove a new "data dependent" bound on the solution quality**
  - Valid for optimizing any submodular function (i.e., also works for influence maximization)

Detecting Contamination Outbreaks

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

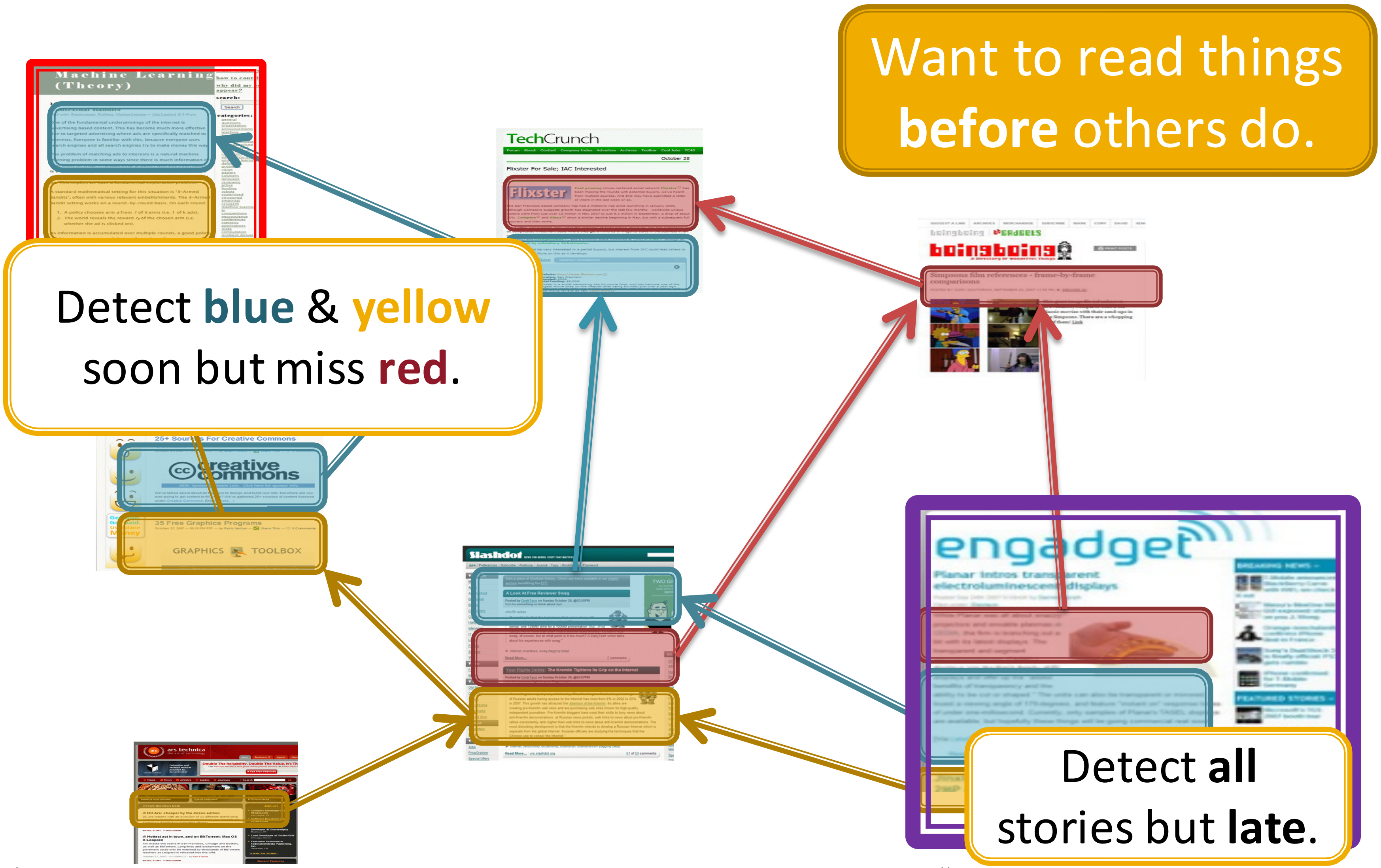


Detecting Information Outbreaks



Which blogs should one read to **detect cascades** as **effectively** as possible?

Detecting Information Outbreaks

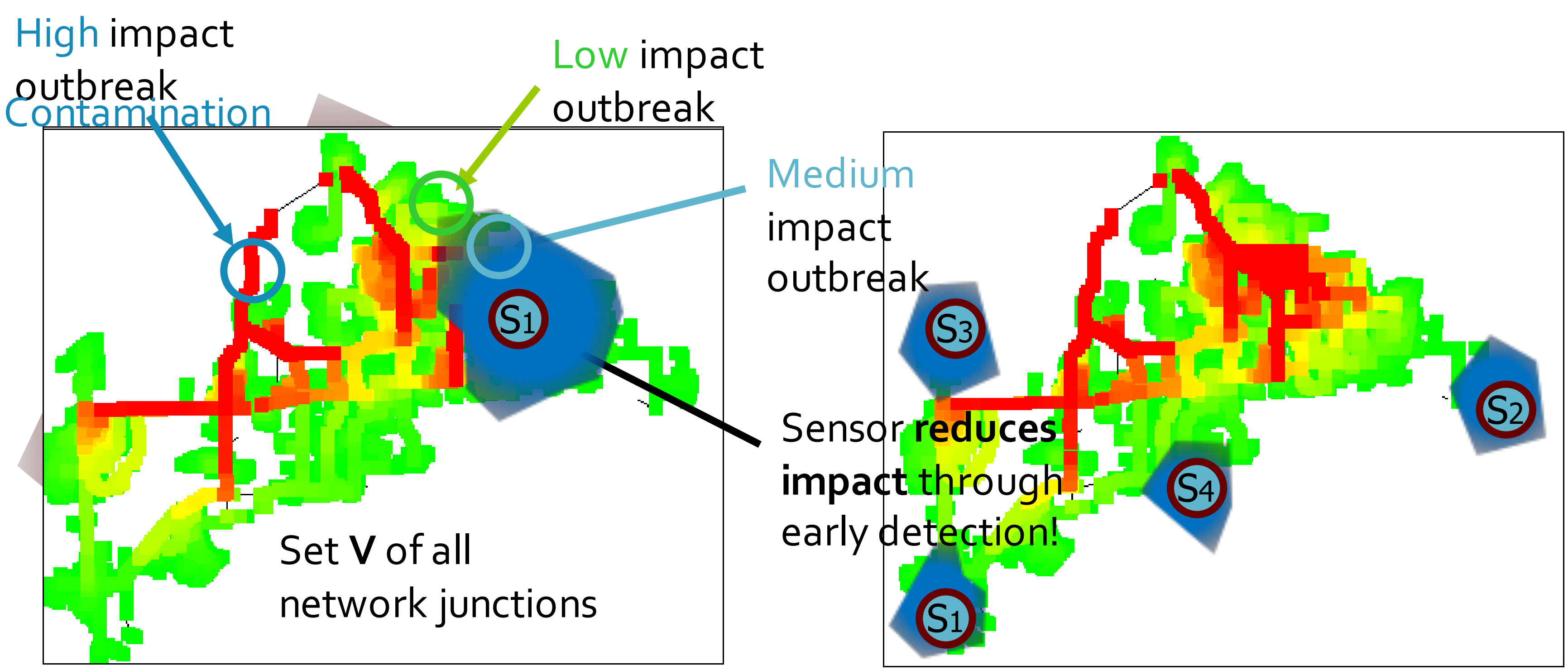


General Problem

- **Both of these two are an instance of the same underlying problem!**
- **Given a dynamic process spreading over a network we want to select a set of nodes to detect the process effectively**
- **Many other applications:**
  - Epidemics
  - Influence propagation
  - Network security

Water Network: Utility

- **Utility of placing sensors:**
  - Water flow dynamics, demands of households, …
- **For each subset S ⊆ V compute utility *f(S)***



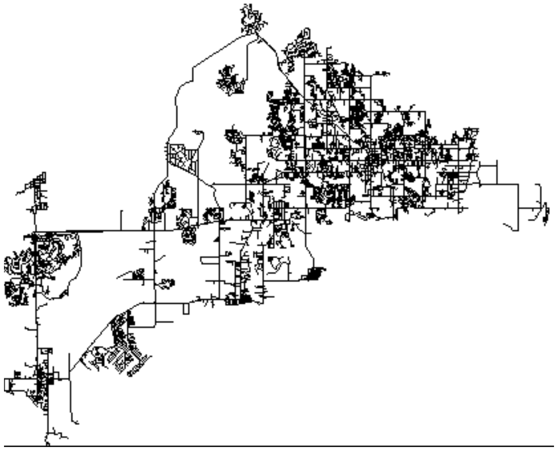
High sensing quality f(S) = 0.9

Low sensing quality f(S)=0.01

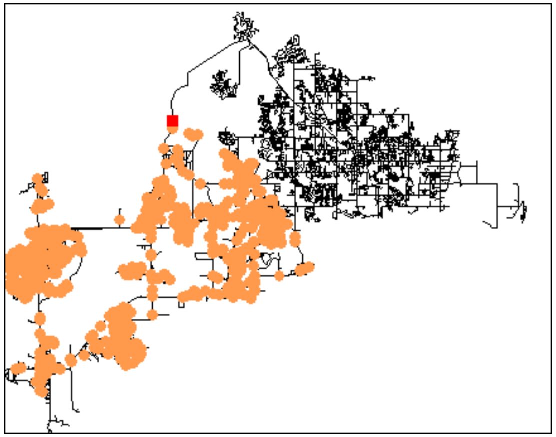
Problem Setting: Contamination

Given:

- Graph $G(V, E)$
- Data on **how outbreaks spread over the G**:
  - For each outbreak i we know the time $T(u, i)$ when outbreak i contaminates node u



Water distribution network
(physical pipes and junctions)

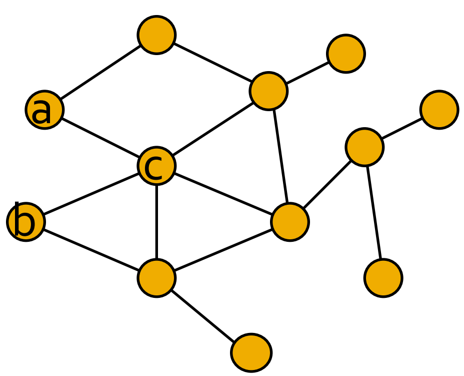


Simulator of water consumption&flow
(built by Mech. Eng. people)
We simulate the contamination spread for every possible location.

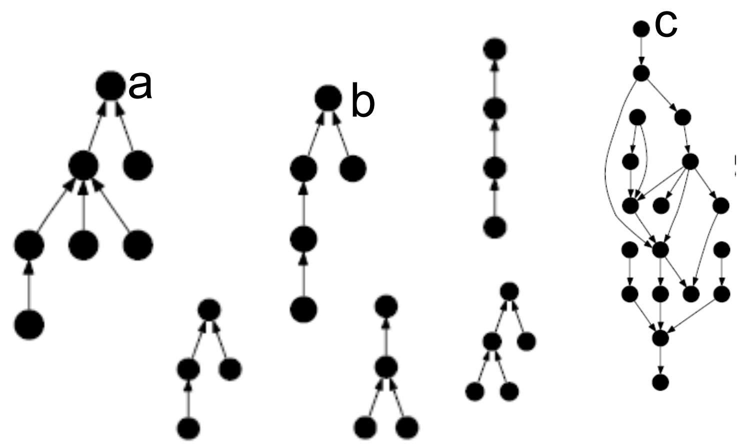
Problem Setting: Blogospehere

Given:

- Graph $G(V, E)$
- Data on **how outbreaks spread over the G**:
  - For each outbreak i we know the time $T(u, i)$ when outbreak i contaminates node u



The network of the blogosphere



Traces of the information flow and identify influence sets

Collect lots of blogs posts and trace hyperlinks to obtain data about information flow from a given blog.

Problem Setting

Given:

- Graph $G(V,E)$
- Data on **how outbreaks spread over the G**:
  - For each outbreak i we know the time $T(u,i)$ when outbreak i contaminates node u

- **Goal:** Select a subset of nodes ***S*** that maximizes the expected **reward**:

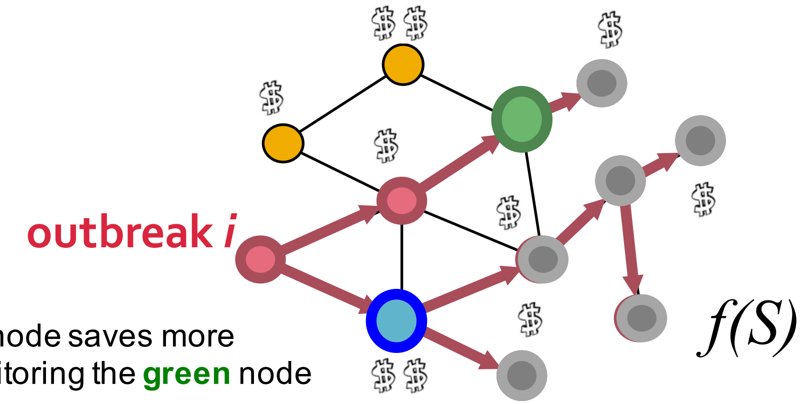
$$\max_{S \subseteq V} f(S) = \sum_i \underbrace{P(i)\ f_i(S)}_{\text{Expected reward for detecting outbreak } i}$$

subject to: $cost(S) < B$

P(i)… probability of outbreak *i* occurring.
f(i)… reward for detecting outbreak *i* using sensors *S*.

Two Parts to the Problem

- **Reward (one of the following three):**
  - **(1)** Minimize time to detection
  - **(2)** Maximize number of detected propagations
  - **(3)** Minimize number of infected people
- **Cost** (context dependent):
  - Reading big blogs is more time consuming
  - Placing a sensor in a remote location is expensive



Monitoring **blue** node saves more people than monitoring the **green** node

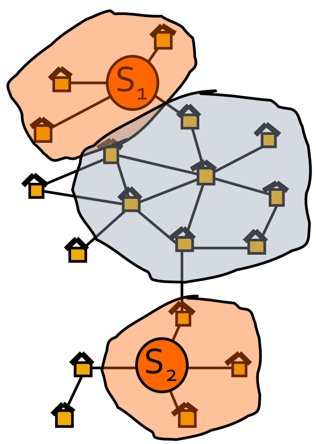
Objective functions are Submodular

$f_i(S)$ is penalty reduction:
$f_i(S) = \pi_i(\emptyset) - \pi_i(S)$

- **Objective functions:**
  - 1) **Time to detection** (**DT**)
    - How long does it take to detect a contamination?
    - **Penalty for detecting at time t:** $\pi_i(t) = \min\{t, T_{max}\}$
  - 2) **Detection likelihood** (**DL**)
    - How many contaminations do we detect?
    - **Penalty for detecting at time t:** $\pi_i(t) = 0, \pi_i(\infty) = 1$
      - Note, this is binary outcome: we either detect or not
  - 3) **Population affected** (**PA**)
    - How many people drank contaminated water?
    - **Penalty for detecting at time t:** $\pi_i(t) =$ {# of infected nodes in outbreak i by time t}.
- **Observation:**
  In all cases detecting sooner does not hurt!

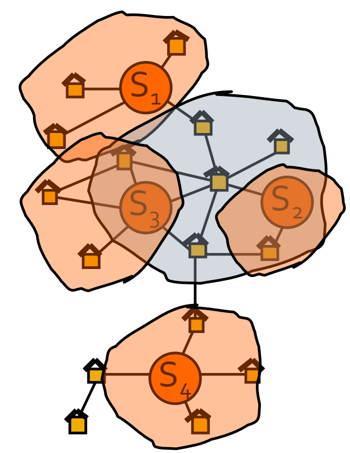
Structure of the Problem

- **Observation: Diminishing returns**



Placement S={s_1, s_2}

New sensor:



Placement S'={s_1, s_2, s_3, s_4}

Adding s' helps a lot

Adding s' helps very little

Objective functions are Submodular

- **Claim:** For all $\boldsymbol{A} \subseteq \boldsymbol{B} \subseteq \boldsymbol{V}$ and sensors $\boldsymbol{s} \in \boldsymbol{V} \backslash \boldsymbol{B}$
  $$\boldsymbol{f}(\boldsymbol{A} \cup \{\boldsymbol{s}\}) - \boldsymbol{f}(\boldsymbol{A}) \geq \boldsymbol{f}(\boldsymbol{B} \cup \{\boldsymbol{s}\}) - \boldsymbol{f}(\boldsymbol{B})$$
- **Proof: All our objectives are submodular**
  - Fix cascade/outbreak $\boldsymbol{i}$
  - **Show** $\boldsymbol{f_i}(\boldsymbol{A}) = \boldsymbol{\pi_i}(\infty) - \boldsymbol{\pi_i}(\boldsymbol{T}(\boldsymbol{A}, \boldsymbol{i}))$ is submodular
  - **Consider** $\boldsymbol{A} \subseteq \boldsymbol{B} \subseteq \boldsymbol{V}$ and sensor $\boldsymbol{s} \in \boldsymbol{V} \backslash \boldsymbol{B}$
  - **When does node $\boldsymbol{s}$ detect cascade $\boldsymbol{i}$?**
    - **We analyze 3 cases based on when $\boldsymbol{s}$ detects outbreak *i***
    - **(1)** $\boldsymbol{T}(\boldsymbol{s}, \boldsymbol{i}) \geq \boldsymbol{T}(\boldsymbol{A}, \boldsymbol{i})$**:** $\boldsymbol{s}$ detects late, nobody benefits:
      $f_i(A \cup \{s\}) = f_i(A)$, also $f_i(B \cup \{s\}) = f_i(B)$ and so
      $f_i(A \cup \{s\}) - f_i(A) = 0 = f_i(B \cup \{s\}) - f_i(B)$

Objective functions are Submodular

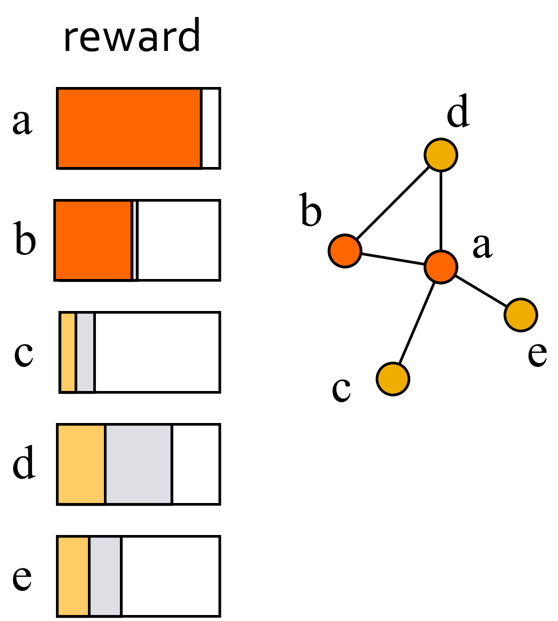
Remember $A \subseteq B$

- **Proof (contd.):**
  - **(2)** $T(B, i) \leq T(s, i) < T(A, i)$: s detects after **B** but before **A**
    s detects sooner than any node in A but after all in B.
    So s only helps improve the solution A (but not B)
    $f_i(A \cup \{s\}) - f_i(A) \geq 0 = f_i(B \cup \{s\}) - f_i(B)$
  - **(3)** $T(s, i) < T(B, i)$: s detects early
    $f_i(A \cup \{s\}) - f_i(A) = \left[\pi_i(\infty) - \pi_i\big(T(s,i)\big)\right] - f_i(A) \geq$
    $\left[\pi_i(\infty) - \pi_i\big(T(s,i)\big)\right] - f_i(B) = f_i(B \cup \{s\}) - f_i(B)$
    - Ineqaulity is due to non-decreasingness of $f_i(\cdot)$, i.e., $f_i(A) \leq f_i(B)$
  - **So, $f_i(\cdot)$ is submodular!**
- **So, $f(\cdot)$ is also submodular**

$$f(S) = \sum_i P(i)\, f_i(S)$$

Background: Submodular functions

Hill-climbing



Add sensor with highest marginal gain

- **What do we know about optimizing submodular functions?**
  - A hill-climbing (i.e., greedy) is near optimal: $(\mathbf{1}-\frac{\mathbf{1}}{\boldsymbol{e}}) \cdot \boldsymbol{OPT}$
- **But:**
  - **(1)** This only works for **unit cost case!** (each sensor costs the same)
    - **For us each sensor $\boldsymbol{s}$ has cost $\boldsymbol{c(s)}$**
  - **(2)** Hill-climbing algorithm is slow
    - At each iteration we need to re-evaluate marginal gains of all nodes
    - Runtime $\boldsymbol{O}(|\boldsymbol{V}| \cdot \boldsymbol{K})$ for placing $\boldsymbol{K}$ sensors

CELF: Algorithm for optimizing submodular functions under cost constraints

Towards a New Algorithm

- **Consider the following algorithm to solve the outbreak detection problem: Hill-climbing that ignores cost**
  - Ignore sensor cost $c(s)$
  - Repeatedly select sensor with highest marginal gain
  - Do this until the budget is exhausted
- **Q: How well does this work?**
- **A: It can fail arbitrarily badly! ☹**
  - Next we come up with an example where Hill-climbing solution is arbitrarily away from OPT

Problem 1: Ignoring Cost

- **Bad example when we ignore cost:**
  - $\boldsymbol{n}$ sensors, budget $\boldsymbol{B}$
  - $\boldsymbol{s_1}$: reward $\boldsymbol{r}$, cost $\boldsymbol{B}$, $\boldsymbol{s_2} \ldots \boldsymbol{s_n}$: reward $\boldsymbol{r - \varepsilon}$,
  - All sensors have the same cost: $\mathrm{c}(\boldsymbol{s_i}) = \boldsymbol{1}$
  - Hill-climbing always prefers more expensive sensor $\boldsymbol{s_1}$ with reward $\boldsymbol{r}$ (and exhausts the budget). It never selects cheaper sensors with reward $\boldsymbol{r - \varepsilon}$
    → **For variable cost it can fail arbitrarily badly!**
- Idea: What if we optimize **benefit-cost ratio**?

$$s_i = \arg\max_{s \in V} \frac{f(A_{i-1} \cup \{s\}) - f(A_{i-1})}{\boldsymbol{c(s)}}$$

Greedily pick sensor $\boldsymbol{s_i}$ that maximizes benefit to cost ratio.

Problem 2: Benefit-Cost

- **Benefit-cost ratio can also fail arbitrarily badly!**
- **Consider**: budget $\boldsymbol{B}$:
  - **2 sensors $\boldsymbol{s_1}$ and $\boldsymbol{s_2}$:**
    - **Costs: $\boldsymbol{c(s_1) = \varepsilon,\ c(s_2) = B}$**
    - **Only 1 cascade: $\boldsymbol{f(s_1) = 2\varepsilon,\ f(s_2) = B}$**
  - **Then benefit-cost ratio is:**
    - $\boldsymbol{B/c(s_1) = 2}$ and $\boldsymbol{B/c(s_2) = 1}$
  - So, we first select $\boldsymbol{s_1}$ and then can not afford $\boldsymbol{s_2}$

→We get reward $\boldsymbol{2\varepsilon}$ instead of $\boldsymbol{B}$! Now send $\boldsymbol{\varepsilon \to 0}$ and we get **arbitrarily bad solution!**

This algorithm incentivizes choosing nodes with very low cost, even when slightly more expensive ones can lead to much better global results.

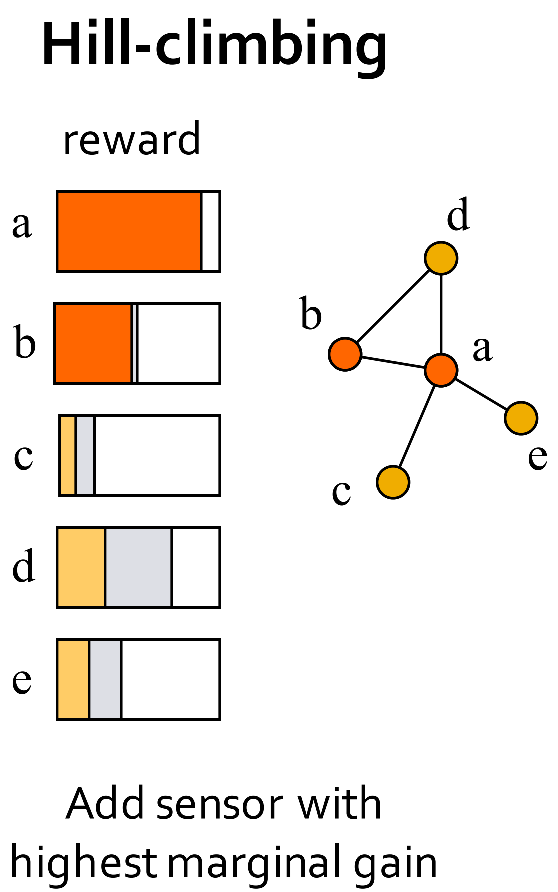
Solution: CELF Algorithm

- **CELF** (**Cost-Effective Lazy Forward-selection**)
  A two pass greedy algorithm:
  - **Set (solution)** S': Use **benefit-cost greedy**
  - **Set (solution)** S'': Use **unit-cost greedy**
  - **Final solution:** $S = \arg\max(f(S'), f(S''))$
- **How far is CELF from (unknown) optimal solution?**
- **Theorem: CELF is near optimal** [Krause&Guestrin, '05]
  - CELF achieves $\frac{1}{2}(1-1/e)$ factor approximation!

This is surprising: We have two clearly suboptimal solutions, but taking best of the two is guaranteed to give a near-optimal solution.

Speeding-up Hill-Climbing: Lazy Evaluations

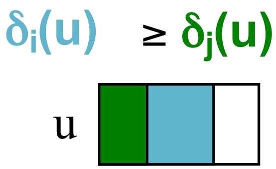
Background: Submodular functions



- **What do we know about optimizing submodular functions?**
  - A hill-climbing (i.e., greedy) is near optimal (that is, $(1 - \frac{1}{e}) \cdot OPT$)
- **But:**
  - **(2)** Hill-climbing algorithm is **slow!**
    - At each iteration we need to re-evaluate marginal gains of all nodes
    - Runtime $O(|V| \cdot K)$ for placing K sensors

Speeding up Hill-Climbing

- **In round $i+1$:** So far we picked $S_i = \{s_1, \dots, s_i\}$
  - Now pick $\mathbf{s}_{i+1} = \arg\max_{u} f(S_i \cup \{u\}) - f(S_i)$
    - This our old friend – greedy hill-climbing algorithm.
      It maximizes the "marginal benefit"
      $\delta_i(u) = f(S_i \cup \{u\}) - f(S_i)$
- **By submodularity property:**
  $f(S_i \cup \{u\}) - f(S_i) \geq f(S_j \cup \{u\}) - f(S_j)$ for $i < j$
- **Observation: By submodularity:**
  For every u
  $\delta_i(u) \geq \delta_j(u)$ for $i < j$ since $S_i \subset Sj$

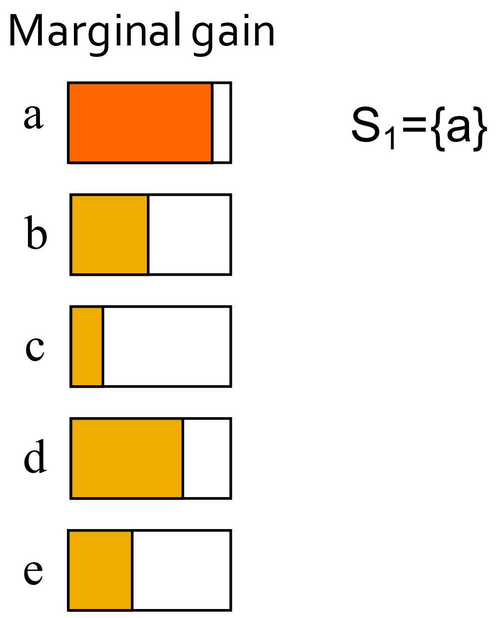


Marginal benefits δ_i(u) only shrink!
(as i grows)

Activating node *u* in step *i* helps more than activating it at step *j* (j>i)

Lazy Hill Climbing

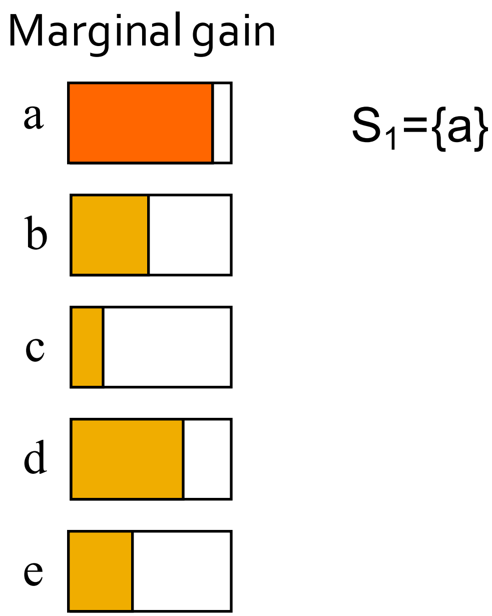
- **Idea:**
  - Use δ_i as upper-bound on δ_j ($j > i$)
- **Lazy hill-climbing:**
  - Keep an ordered list of marginal benefits δ_i from previous iteration
  - Re-evaluate δ_i only for top node
  - Re-sort and prune



$$f(S \cup \{u\}) - f(S) \geq f(T \cup \{u\}) - f(T) \qquad S \subseteq T$$

Lazy Hill Climbing

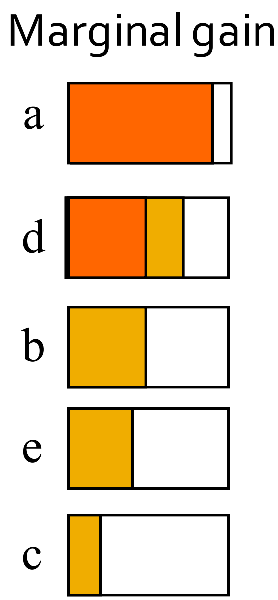
- **Idea:**
  - Use δ_i as upper-bound on δ_j ($j > i$)
- **Lazy hill-climbing:**
  - Keep an ordered list of marginal benefits δ_i from previous iteration
  - Re-evaluate δ_i only for top node
  - Re-sort and prune



$$f(S \cup \{u\}) - f(S) \geq f(T \cup \{u\}) - f(T) \qquad S \subseteq T$$

Lazy Hill Climbing

- **Idea:**
  - Use δ_i as upper-bound on δ_j ($j > i$)
- **Lazy hill-climbing:**
  - Keep an ordered list of marginal benefits δ_i from previous iteration
  - Re-evaluate δ_i only for top node
  - Re-sort and prune

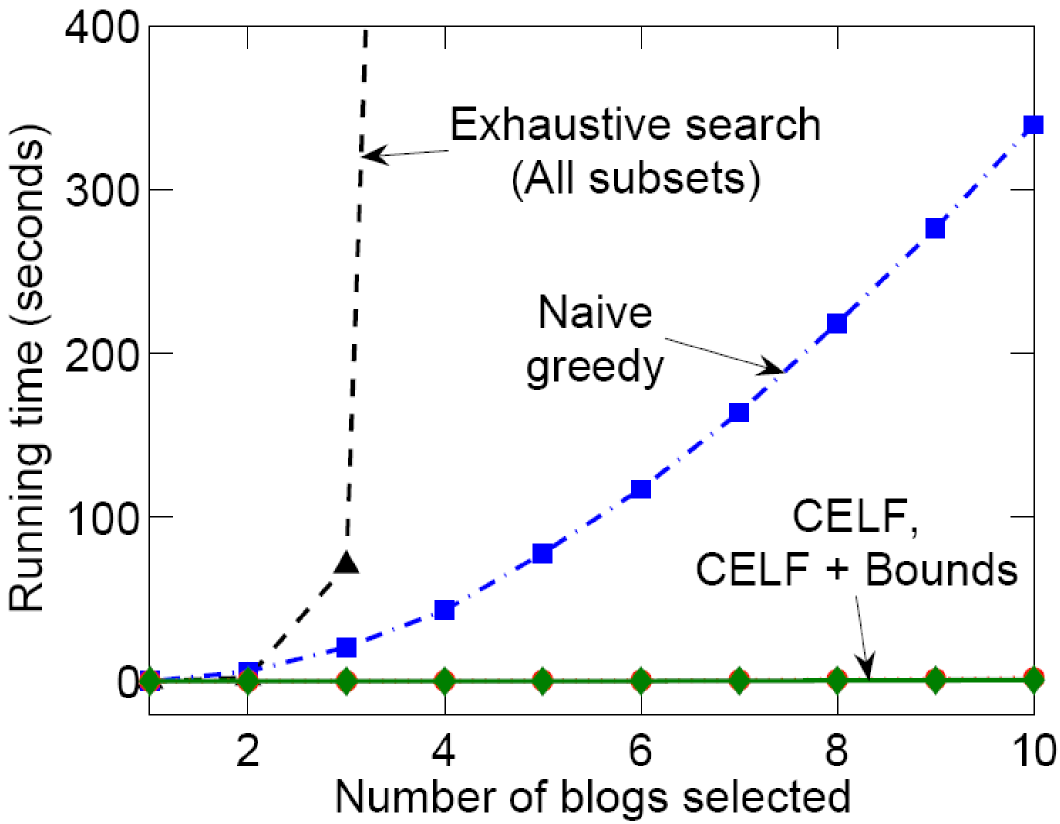


$S_1=\{a\}$

$S_2=\{a,b\}$

$$f(S \cup \{u\}) - f(S) \geq f(T \cup \{u\}) - f(T) \qquad S \subseteq T$$

CELF: Scalability



- CELF (using Lazy evaluation) runs **700** times faster than greedy hill-climbing algorithm

Data Dependent Bound on the Solution Quality

Solution Quality

- **Back to the solution quality!**
- **The (1-1/e) bound for submodular functions is the worst case bound (worst over all possible inputs)**
- **Data dependent bound:**
  - Value of the bound depends on the input data
    - On “easy” data, hill climbing may do better than 63%
  - **Can we say something about the solution quality when we know the input data?**

Data Dependent Bound

- Suppose S **is some solution** to $f(S)$ s.t. $|S| \leq k$
  - $f(S)$ is monotone & submodular
- Let $OPT = \{t_1, \dots, t_k\}$ be the OPT solution
- For each u let $\delta(u) = f(S \cup \{u\}) - f(S)$
- Order $\delta(u)$ so that $\delta(1) \geq \delta(2) \geq \cdots$
- **Then:** $f(OPT) \leq f(S) + \sum_{i=1}^{k} \delta(i)$
  - **Note:**
    - This is a data dependent bound ($\delta(i)$ depends on input data)
    - Bound holds for **any** algorithm
      - Makes no assumption about how S was computed
    - For some inputs it can be very "loose" (worse than 63%)

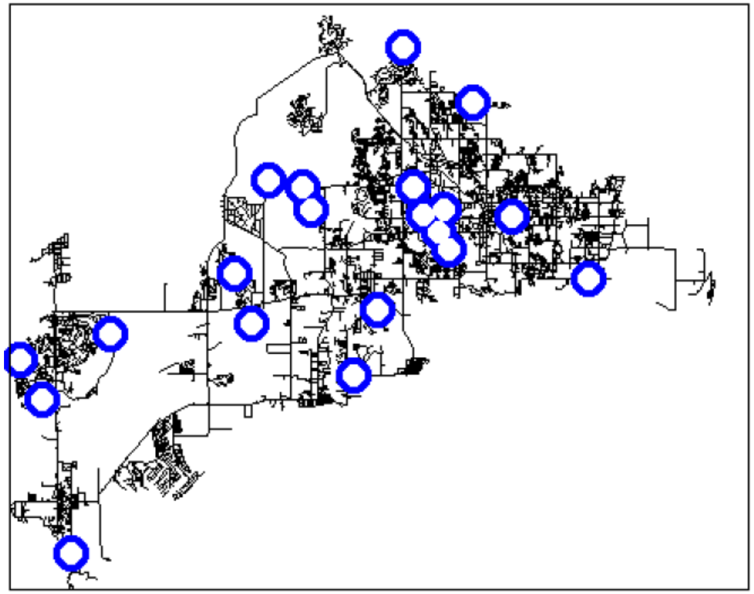
Data Dependent Bound

- **Claim:**
  - For each u let $\boldsymbol{\delta(u) = f(S \cup \{u\}) - f(S)}$
  - Order $\boldsymbol{\delta(u)}$ so that $\boldsymbol{\delta(1) \geq \delta(2) \geq \cdots}$
  - **Then:** $\boldsymbol{f(OPT) \leq f(S) + \sum_{i=1}^{k} \delta(i)}$
- **Proof:**
  - $f(OPT) \leq f(OPT \cup S)$
  - $= f(S) + \sum_{i=1}^{k} [f(S \cup \{t_1 \dots t_i\}) - f(S \cup \{t_1 \dots t_{i-1}\})]$
  - $\leq f(S) + \sum_{i=1}^{k} [f(S \cup \{t_i\}) - f(S)]$ (we proved this last time)
  - $= f(S) + \sum_{i=1}^{k} \delta(t_i)$ Instead of taking $t_i \in OPT$ (of benefit $\delta(t_i)$), we take the best possible element ($\delta(i)$)
  - $\leq f(S) + \sum_{i=1}^{k} \delta(i) \Rightarrow \boldsymbol{f(T) \leq f(S) + \sum_{i=1}^{k} \delta(i)}$

Case Study: Water distribution network & blogs

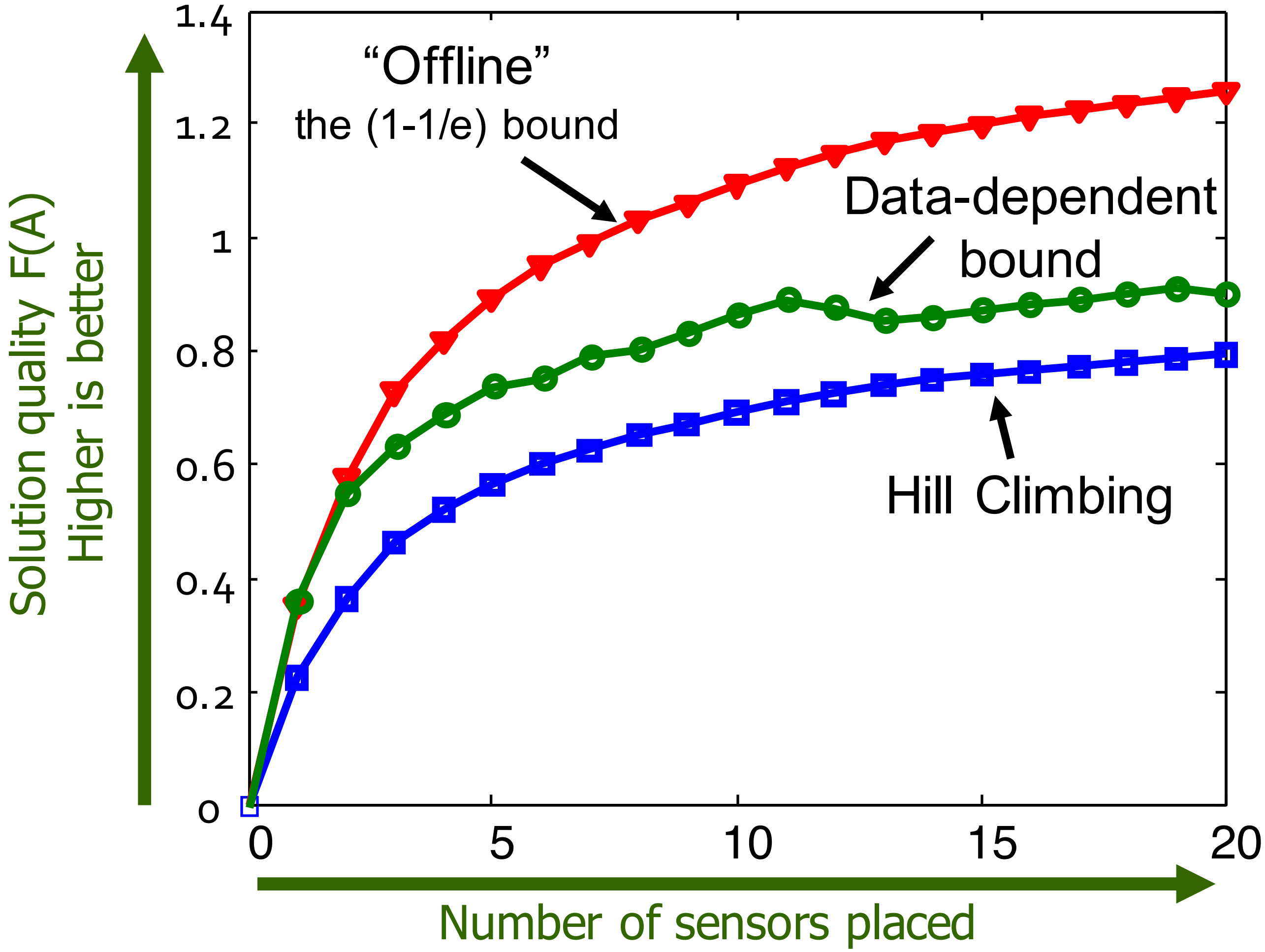
Case Study: Water Network

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



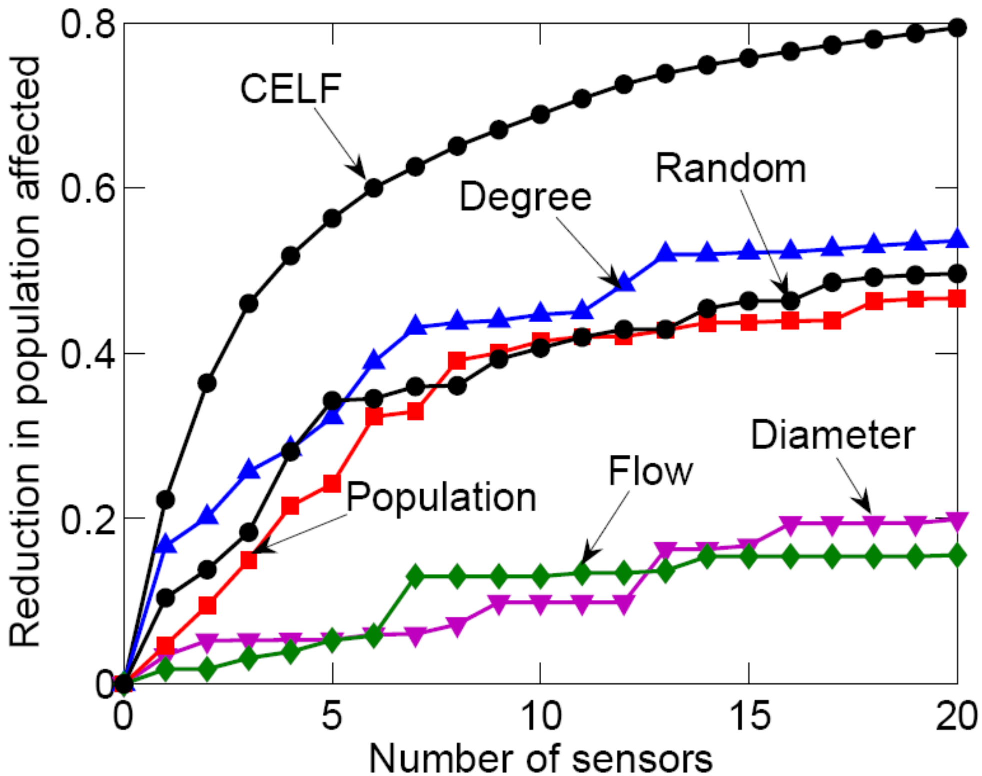
- Use a cluster of 50 machines for a month
- Simulate 3.6 million epidemic scenarios (random locations, random days, random time of the day)

Bounds on the Optimal Solution



Data-dependent bound is much tighter
(gives more accurate estimate of alg. performance)

Water: Heuristic Placement



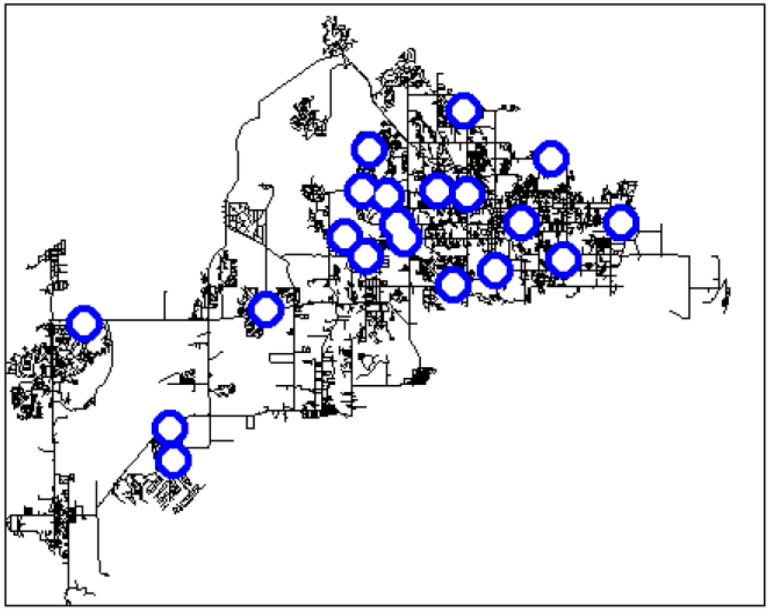
- **Placement heuristics perform much worse**

Author	Score
CELF	**26**
Sandia	21
U Exter	20
Bentley systems	19
Technion (1)	14
Bordeaux	12
U Cyprus	11
U Guelph	7
U Michigan	4
Michigan Tech U	3
Malcolm	2
Proteo	2
Technion (2)	1

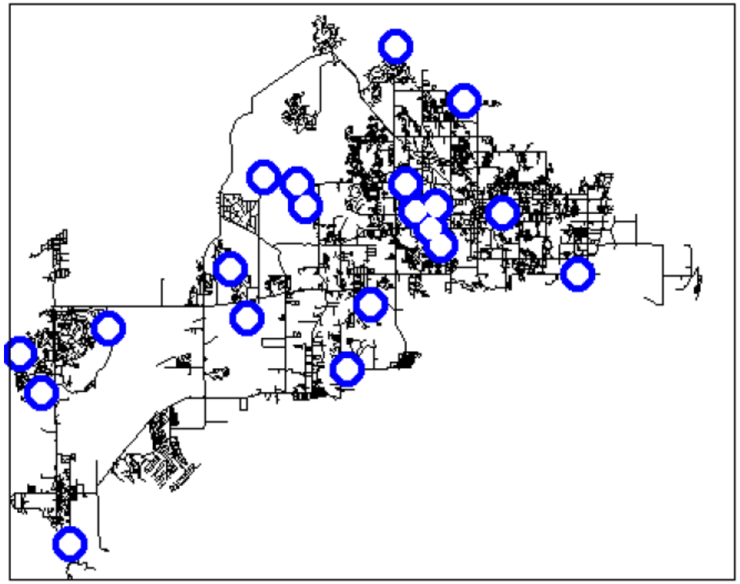
Battle of Water Sensor Networks competition

Water: Placement visualization

- Different objective functions give different sensor placements

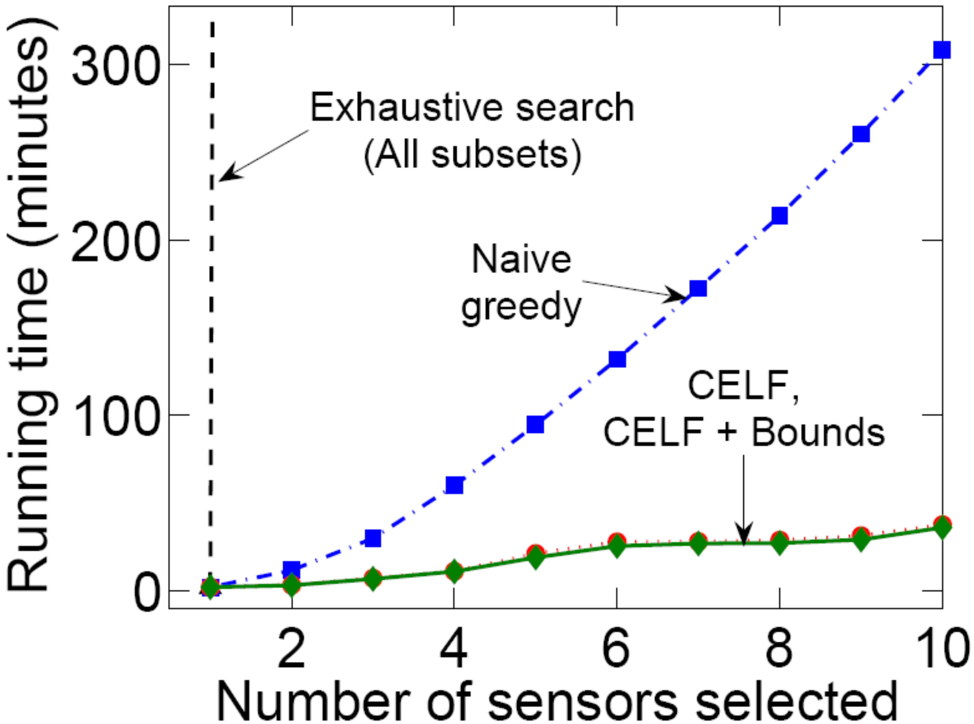


Population affected



Detection likelihood

Water: Scalability



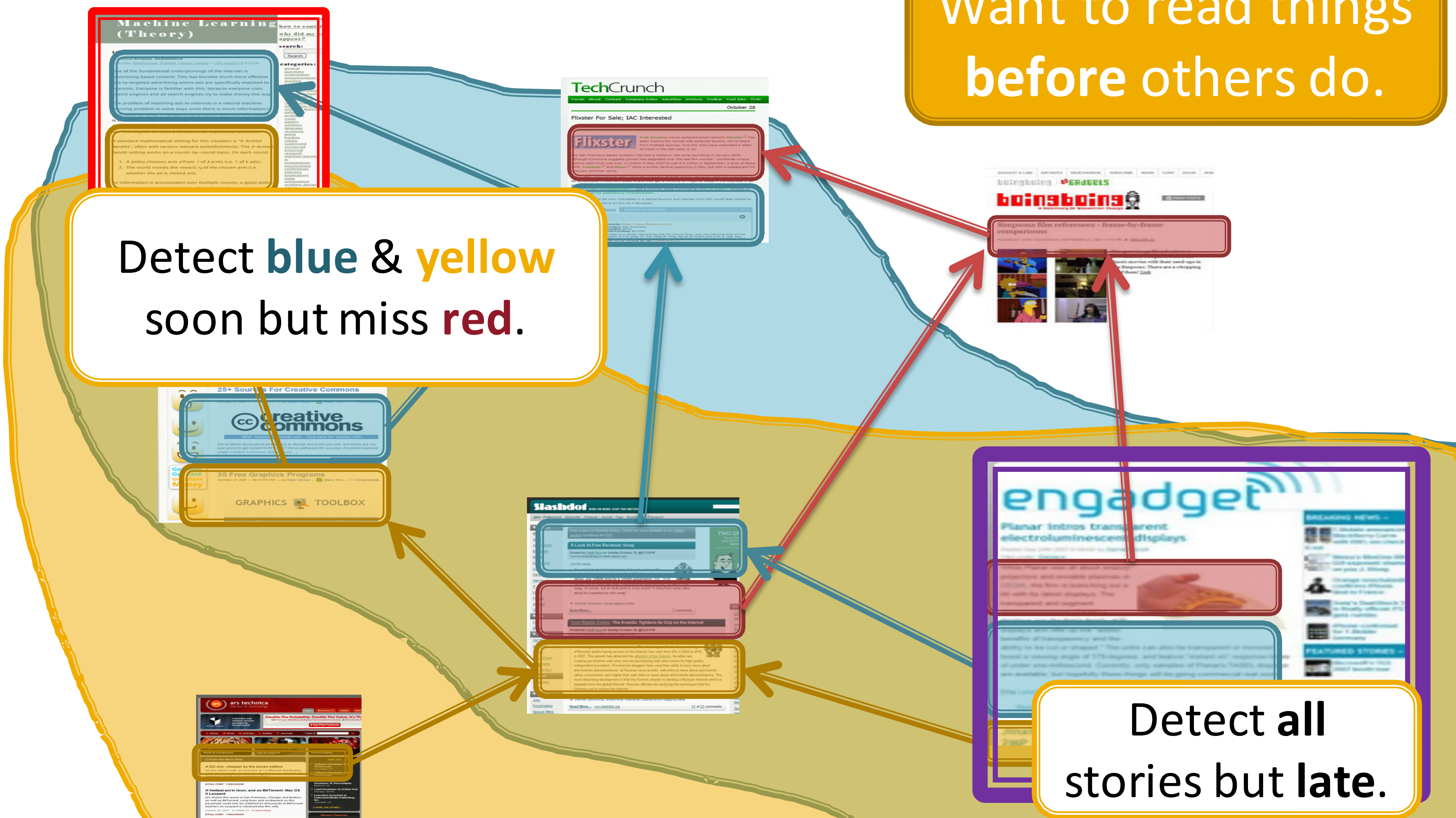
- CELF is **10** times faster than greedy hill-climbing!

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

Want to read things **before** others do.

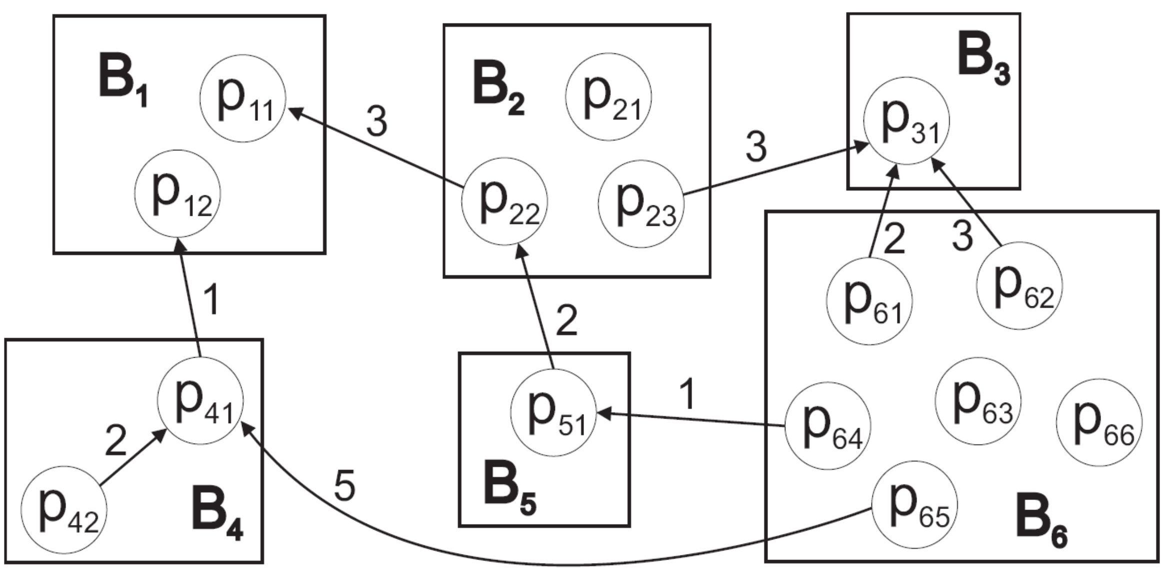


Detect **blue** & **yellow** soon but miss **red**.

Detect **all** stories but **late**.

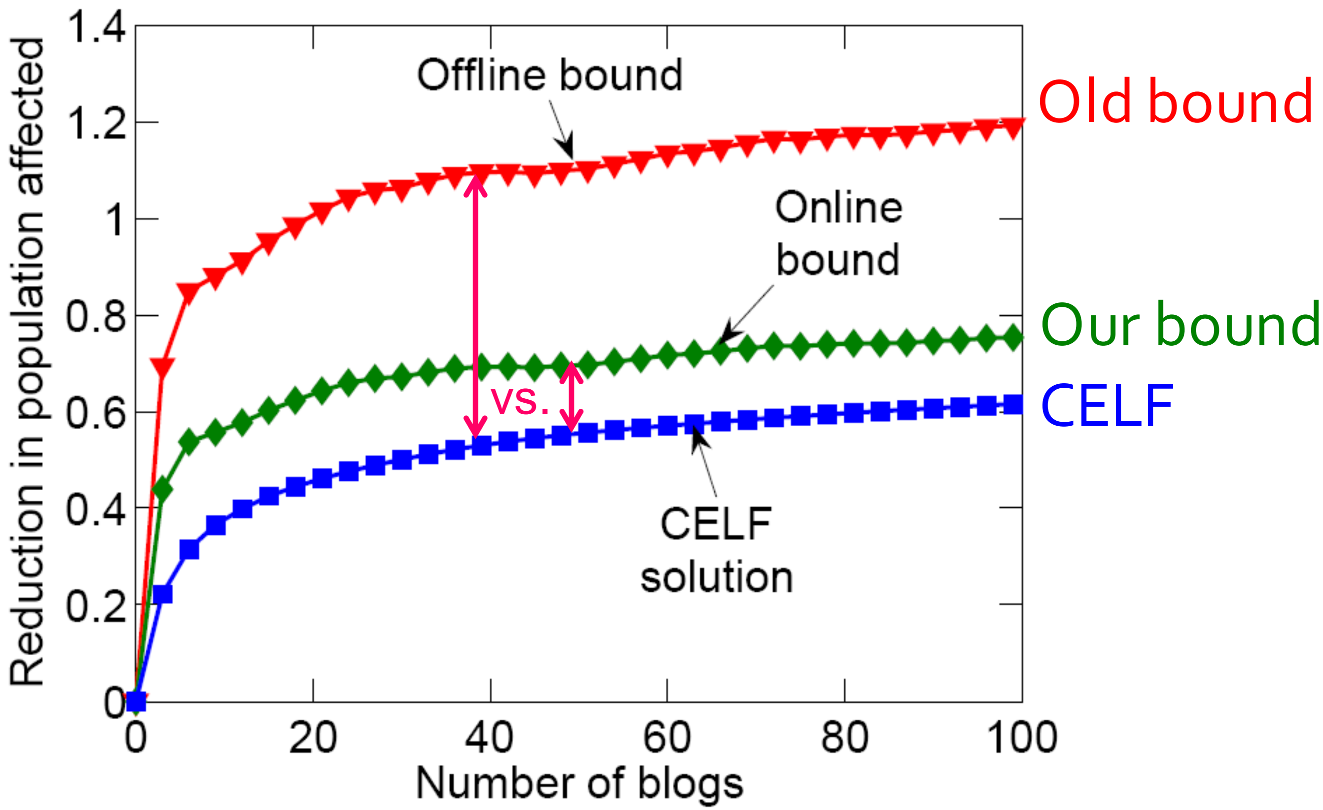
Case study 2: Cascades in blogs

- Crawled 45,000 blogs for 1 year
- Obtained 10 million posts
- And identified 350,000 cascades
- Cost of a blog is the number of posts it has

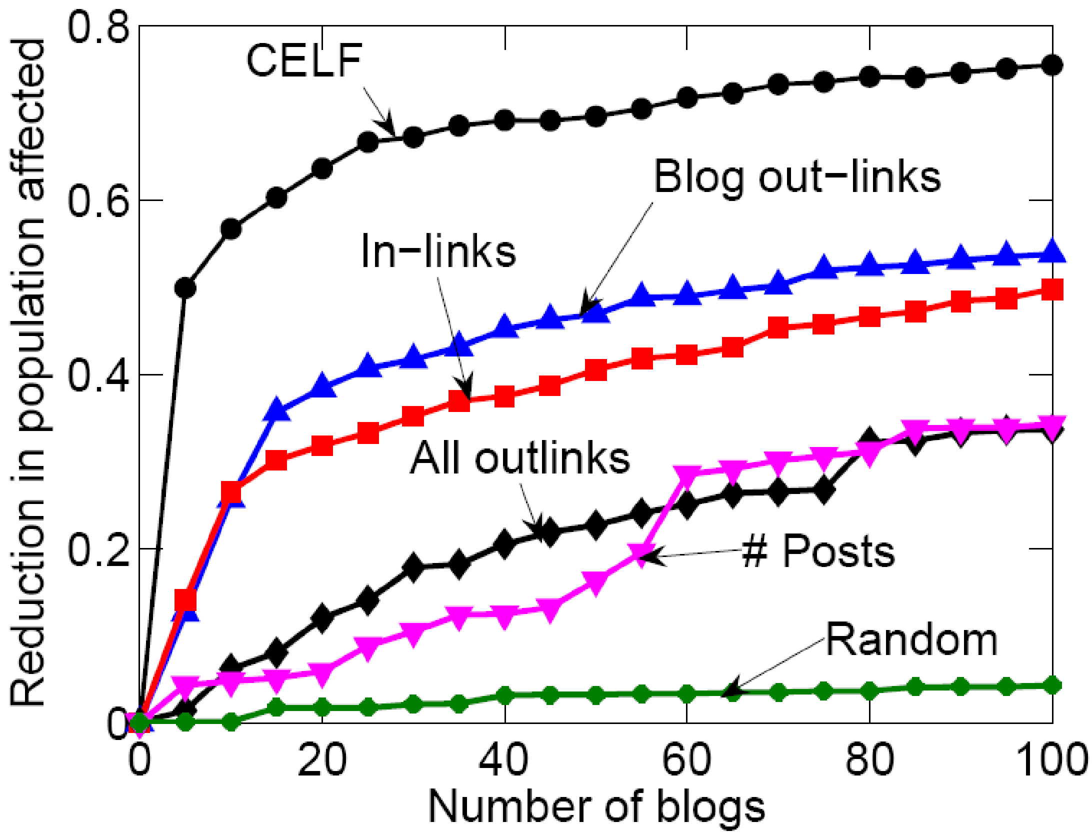


Blogs: Solution Quality

- **Online bound turns out to be much tighter!**
  - Based on the plot below: 87% instead of 32.5%



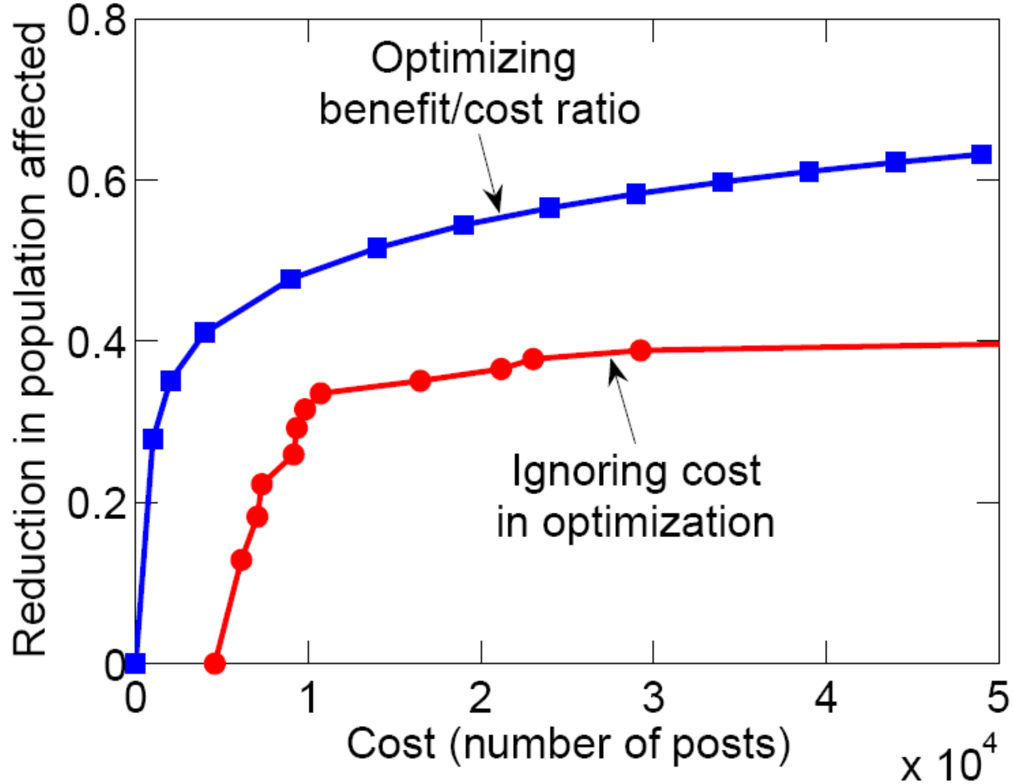
Blogs: Heuristic Selection



- **Heuristics perform much worse!**
- **One really needs to perform the optimization**

Blogs: Cost of a Blog

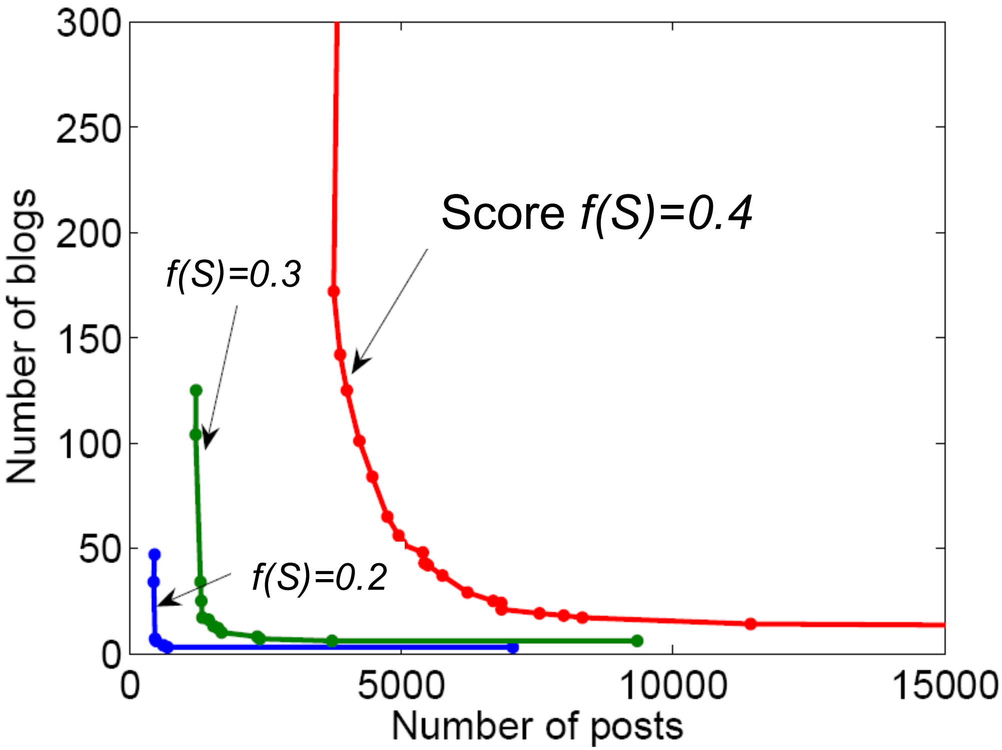
- **CELF has 2 sub-algorithms. Which wins?**
- **Unit cost:**
  - CELF picks large popular blogs
- **Cost-benefit:**
  - Cost proportional to the number of posts



- **We can do much better when considering costs**

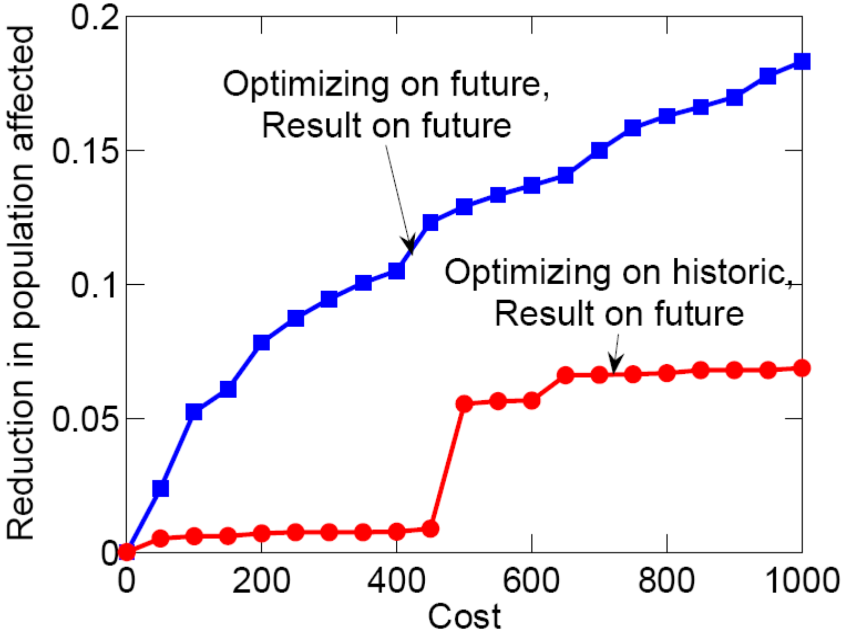
Blogs: Cost of a Blog

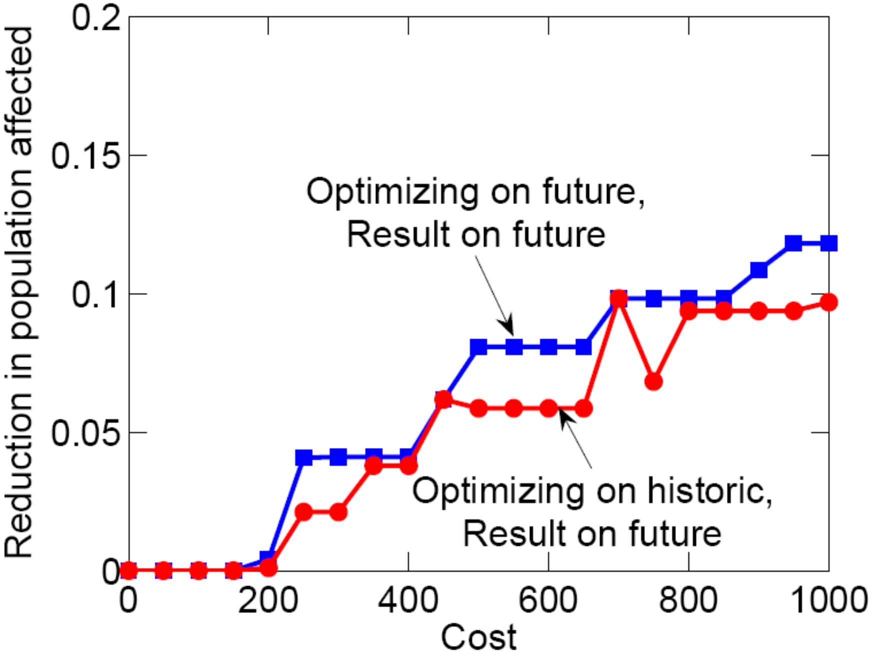
- **Problem:** Then CELF picks **lots of small blogs** that participate in few cascades
- We pick best solution that interpolates between the costs
- We can get good solutions with **few blogs and few posts**



Each curve represents a set of solutions *S* with the same final reward *f(S)*

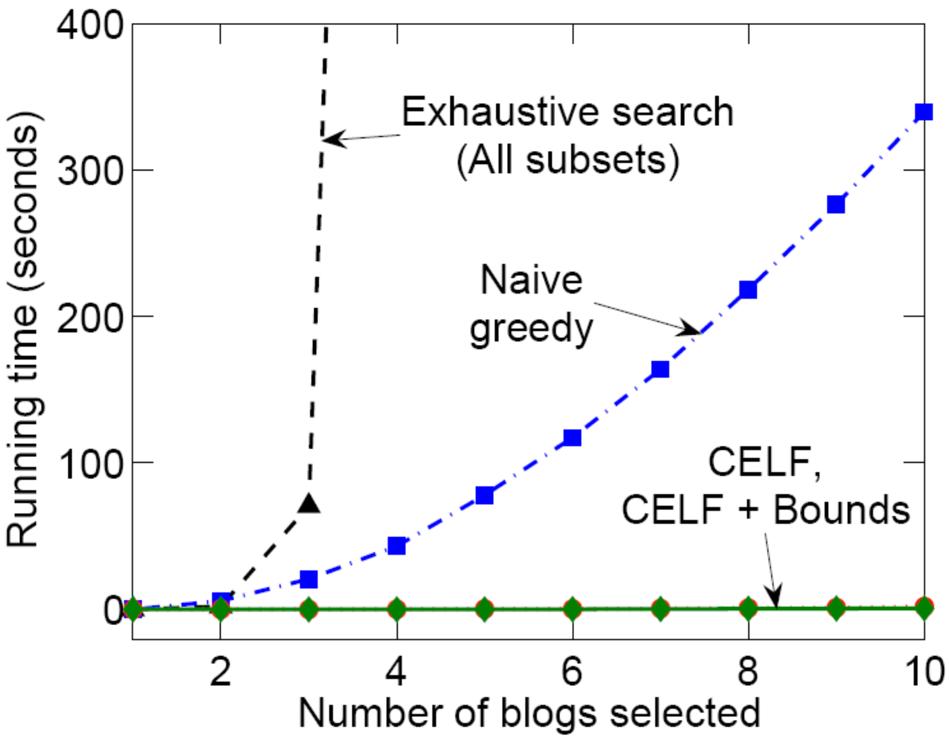
Blogs: Generalization to Future





- We want to generalize well to future (unknown) cascades
- Limiting selection to bigger blogs improves generalization!

Blogs: Scalability



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