Modelling and Analysis of Complex Networks

— Semester 3, Master of Data Science —

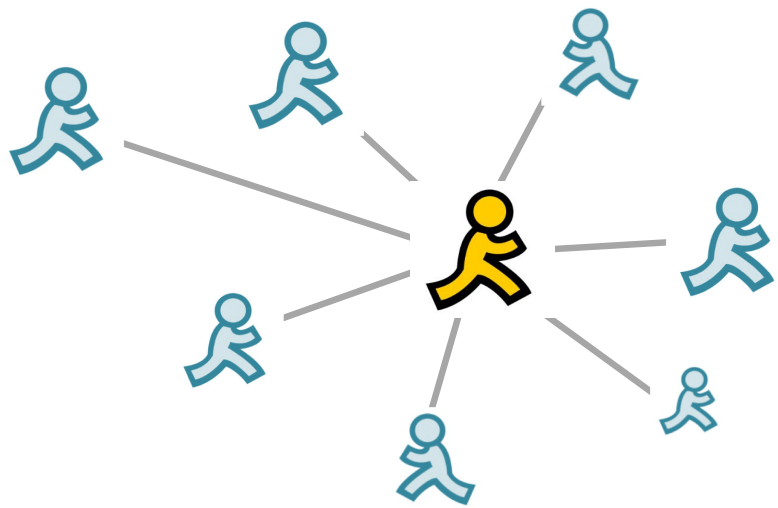
George Panagopoulos
University of Luxembourg

Exposure Curves

Exposures and Adoptions [Myers et al. 2012]

From exposure to adoption

- Exposure: node's neighbour exposes the node to the contagion
- Adoption: The node acts on the contagion

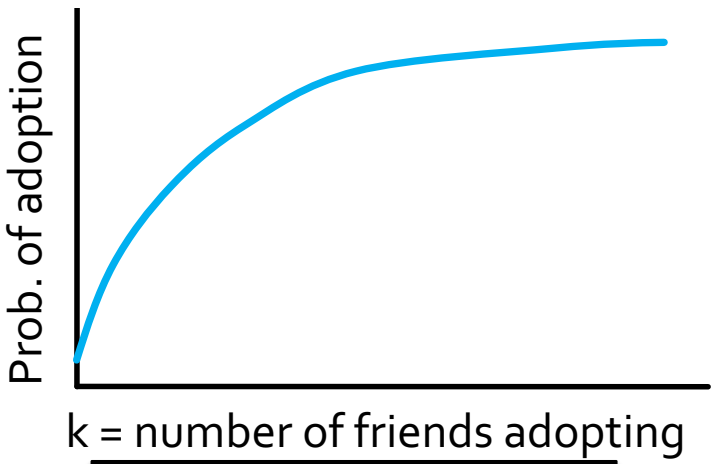


Exposures Curves

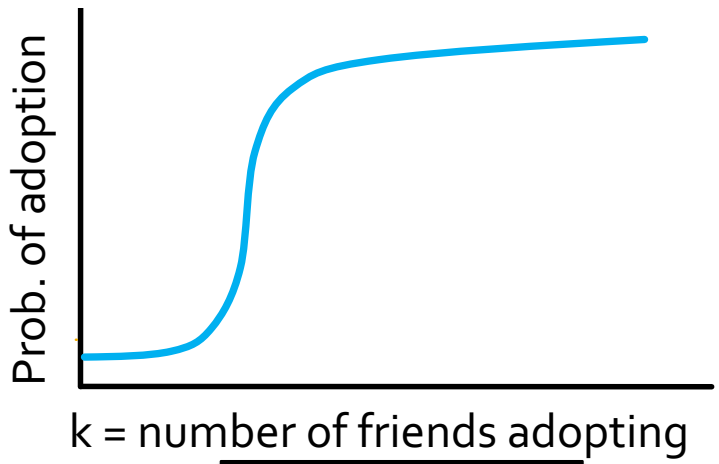
Exposure curve: probability of adopting new behaviour depends on the total number of neighbours who have already adopted it

Exposures Curves

Exposure curve: probability of adopting new behaviour depends on the total number of neighbours who have already adopted it



"Probabilistic" spreading:
Viruses, Information

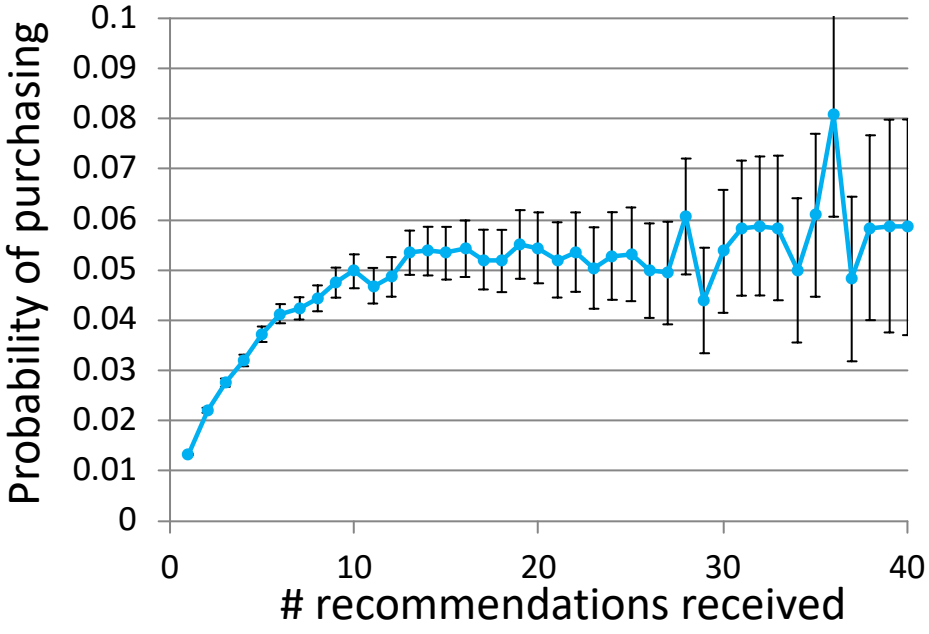


Critical mass:
Decision making

Exposures Curves

- Senders and followers of recommendations receive discounts
- Viral marketing program

  (16 million recommendations, 4 million people, 500k products)

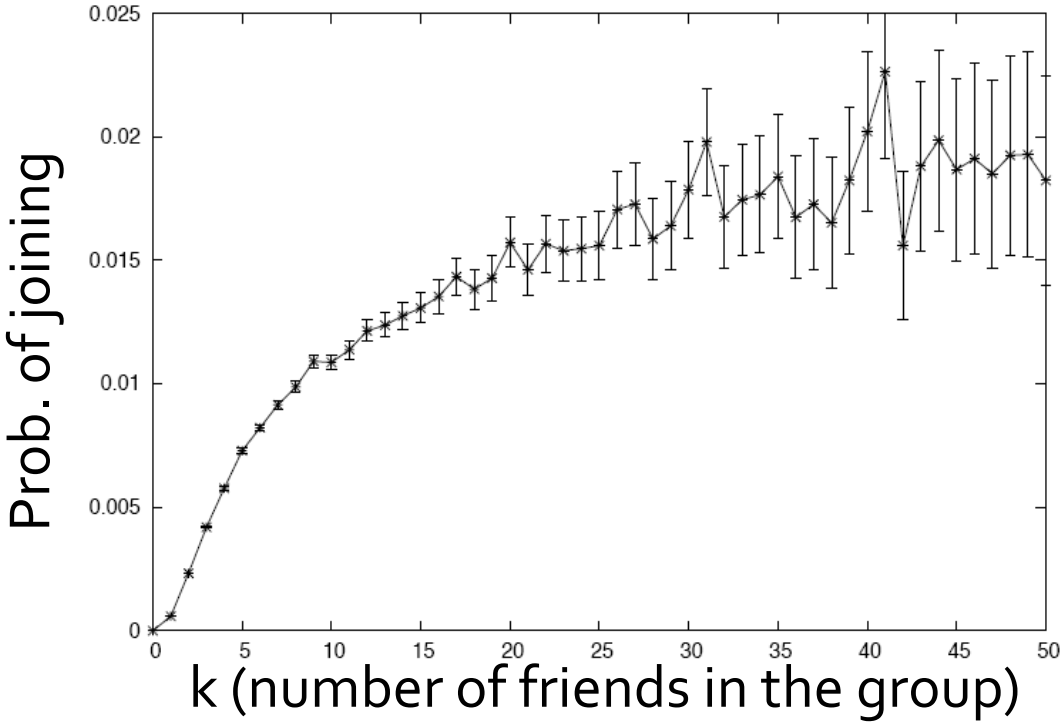


DVD recommendations
(8.2 million observations)

[Leskovec et al. 2007]

Exposures Curves

- Group memberships spread over the network
- How does probability of joining a group depend on the number of friends already in the group?



[Backstrom et al. 2006]

Viral Marketing

Social Media Influence

[Prakash and Ramakrishnan, KDD '16]

Viral Marketing

We are influenced more by our friends than strangers.

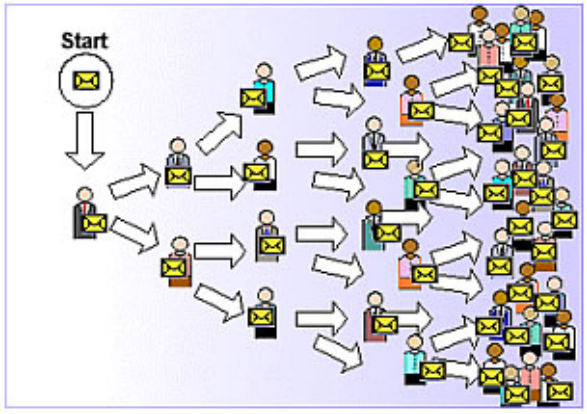
- 68% of consumers consult friends and family before purchasing home electronics
- 50% do research online before purchasing electronics

Viral Marketing

The Kate Middleton effect (also the Kate effect) refers to the trend effect that Kate Catherine has on others, from cosmetic surgery for brides, to sales of coral-coloured jeans.

Viral Marketing

Identify influential customers



These customers endorse the product among their friends

Convince them to adopt the product – Offer discount or free samples

Simple Contagion Models

Independent cascade model

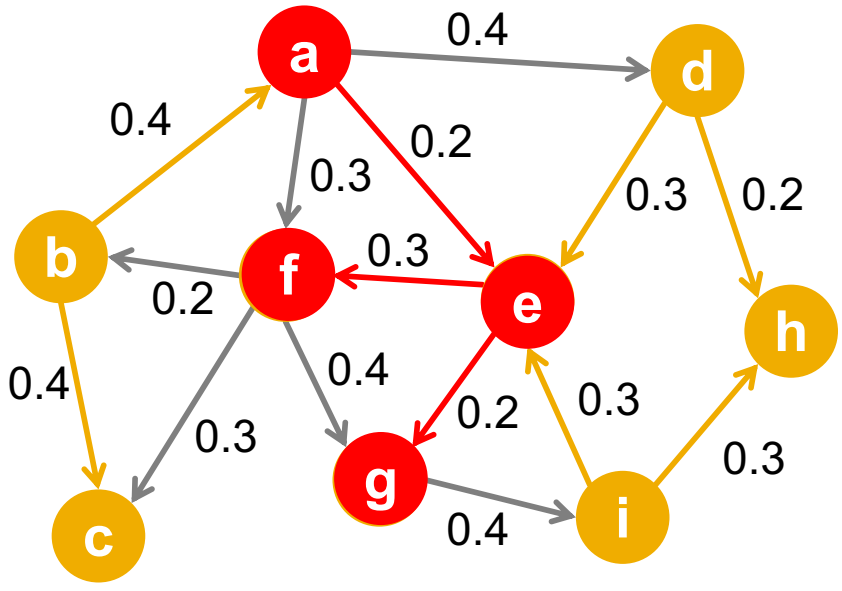
- Independent cascade model
  - Directed finite $G = (V, E)$
  - Set S starts out with new behaviour (nodes with this behaviour are "active")
  - Each edge (v, w) has a probability p_{vw}
  - If node v is active, it gets one chance to make w active, with probability p_{vw} (each edge fires at most once)
- Does scheduling matter? No.
  - If u, v are both active at the same time, it doesn't matter which tries to activate w first
  - Time moves in discrete steps

Independent cascade model

- Independent cascade model
  - Directed finite $G = (V, E)$
  - Set S starts out with new behaviour (nodes with this behaviour are "active")
  - Each edge (v, w) has a probability p_{vw}
  - If node v is active, it gets one chance to make w active, with probability p_{vw} (each edge fires at most once)
- Does scheduling matter? No.
  - If u, v are both active at the same time, it doesn't matter which tries to activate w first
  - Time moves in discrete steps

Independent Cascade Model

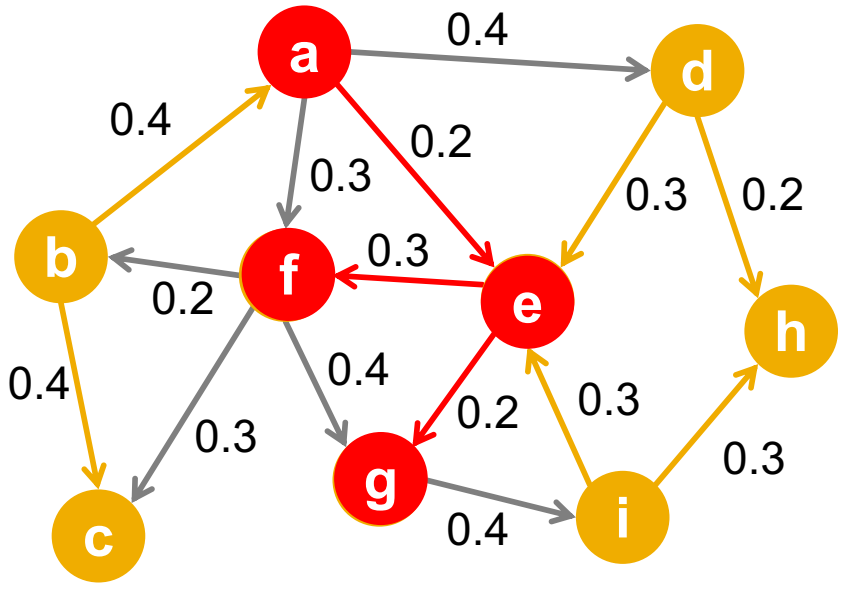
- Initially some nodes S are active
- Each edge (u, v) has a (random) probability p_{uv}



- When node u becomes active/infected, it activates each out-neighbour v with probability p_{uv}
- Activations spread through the networks!

Independent Cascade Model

- Initially some nodes S are active
- Each edge (u, v) has a (random) probability p_{uv}



- When node u becomes active/infected, it activates each out-neighbour v with probability p_{uv}
- Activations spread through the networks!

Linear Threshold Model

A node v has a random threshold $\theta_v \sim U[0,1]$

- v is influenced by each neighbor u according to a weight $w_{u,v}$.
- All weights for each v sum to 1:

$$\sum_{u \in N(v)} w_{u,v} \leq 1$$

Node v becomes active when the weighted combination of its active neighbors surpasses the threshold θ_v:

$$\sum_{u \in N(v) \cap A} w_{u,v} \geq \theta_v$$

The Live-edge Model

The aforementioned models are stochastic processes.

Hence to calculate the exact influence spread in a graph $G(V, E)$, one has to consider all possible $2^{|E|}$ combinations that represent different realizations of the graph.

The live-edge model: Flip a coin for each edge and keep the subgraph g formed by the succesfull ones.

- For every node v, the count of nodes reachable by the subgraph's paths is the influence spread under this realization: $\sigma_g(s)$.
- Compute $\sigma_g(s)$ for all possible graphs weighted by the respective probability, to get the estimated influence spread:
  $\sigma(v) = \sum_{g \subseteq G} P(g)\sigma_g(s)$

The Live-edge Model

The aforementioned models are stochastic processes.

Hence to calculate the exact influence spread in a graph $G(V, E)$, one has to consider all possible $2^{|E|}$ combinations that represent different realizations of the graph.

The live-edge model: Flip a coin for each edge and keep the subgraph g formed by the succesfull ones.

- For every node v, the count of nodes reachable by the subgraph's paths is the influence spread under this realization:$\sigma_g(s)$,
- Compute $\sigma_g(s)$ for all possible graphs weighted by the respective probability, to get the estimated influence spread:
  $\sigma(v) = \sum_{g \subseteq G} P(g)\sigma_g(s)$

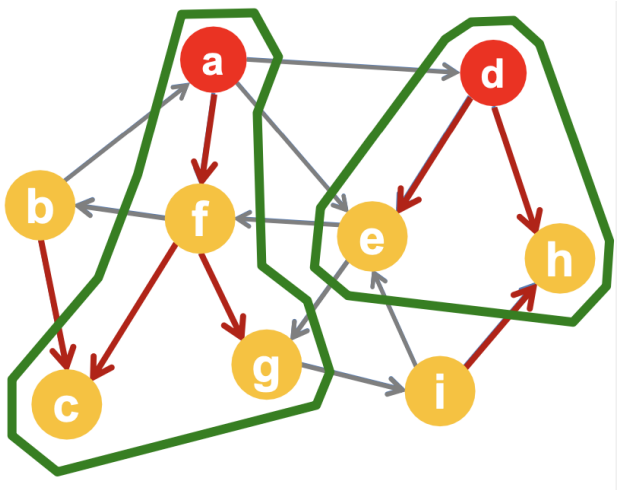
The Live-edge Model

The aforementioned models are stochastic processes.

Hence to calculate the exact influence spread in a graph $G(V, E)$, one has to consider all possible $2^{|E|}$ combinations that represent different realizations of the graph.

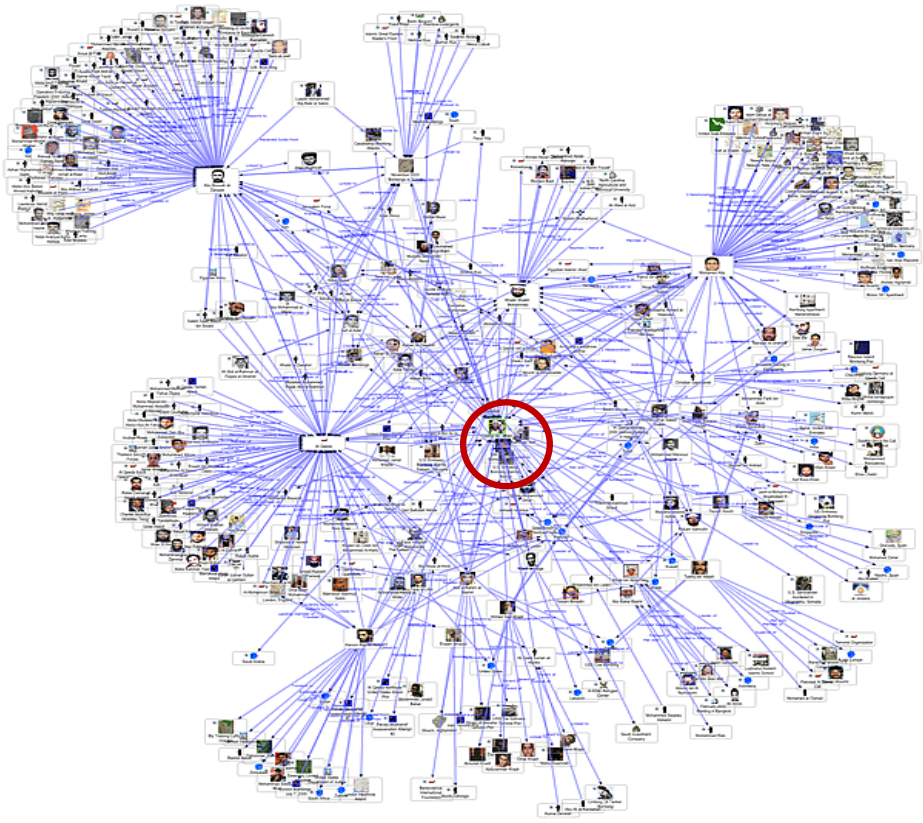
The live-edge model: Flip a coin for each edge and keep the subgraph g formed by the succesfull ones.

- For every node v, the count of nodes reachable by the subgraph's paths is the influence spread under this realization:$\sigma_g(s)$,
- Compute $\sigma_g(s)$ for all possible graphs weighted by the respective probability, to get the estimated influence spread:
  $\sigma(v) = \sum_{g \subseteq G} P(g)\sigma_g(s)$



Good Influencers and Where to Find Them

How to Find Influential Users



- Influential persons often have many friends
- Kate is one of the persons that have many friends in this social network
- How to find as many influential users as possible?

How to Find Influential Users

Typically, a two-step approach:

1. Consider a topological or centrality criterion of the nodes of the network.
2. Rank the nodes accordingly.
3. The top-ranked nodes are candidates for the most influential ones.
4. Simulate the spreading process over the network to examine the performance of the chosen nodes e.g., using the IC or the LT model.

Graph Degeneracy

Ranking based on degree and pagerank do not suffice to detect the most influential nodes.
Cascades require the dynamics of close-knit cliques to "convince" the users to adopt them.

- The k-core of a graph is the maximum subgraph such that each node has degree k or more in the subgraph.
- The core value of a node v is the maximum number $c(v)$ such that the node v belongs to the $c(v)$-core.
- The degeneracy of a graph is the maximum number c such that a c-core exists.

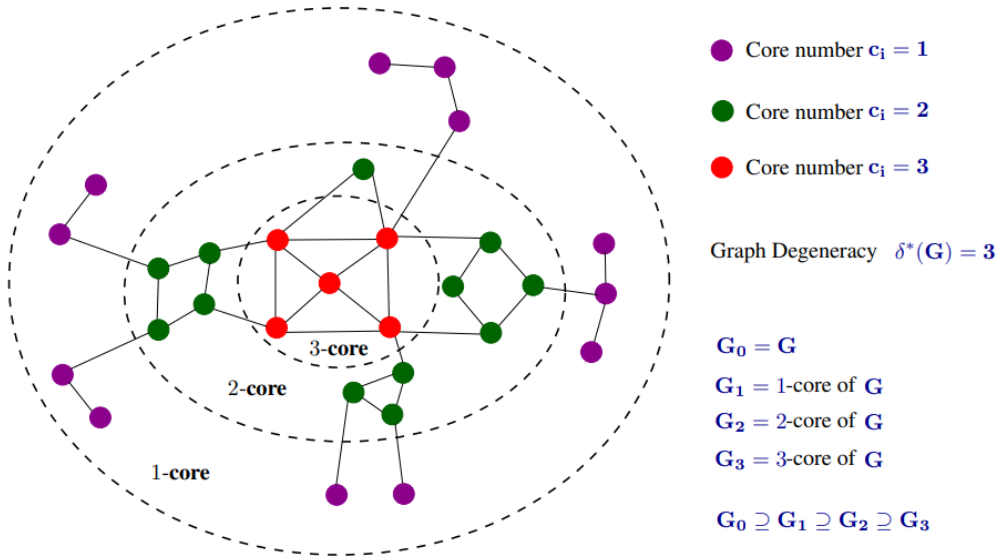
Graph Degeneracy

Ranking based on degree and pagerank do not suffice to detect the most influential nodes.
Cascades require the dynamics of close-knit cliques to "convince" the users to adopt them.

- The k-core of a graph is the maximum subgraph such that each node has degree k or more in the subgraph.
- The core value of a node v is the maximum number $c(v)$ such that the node v belongs to the $c(v)$-core.
- The degeneracy of a graph is the maximum number c such that a c-core exists.

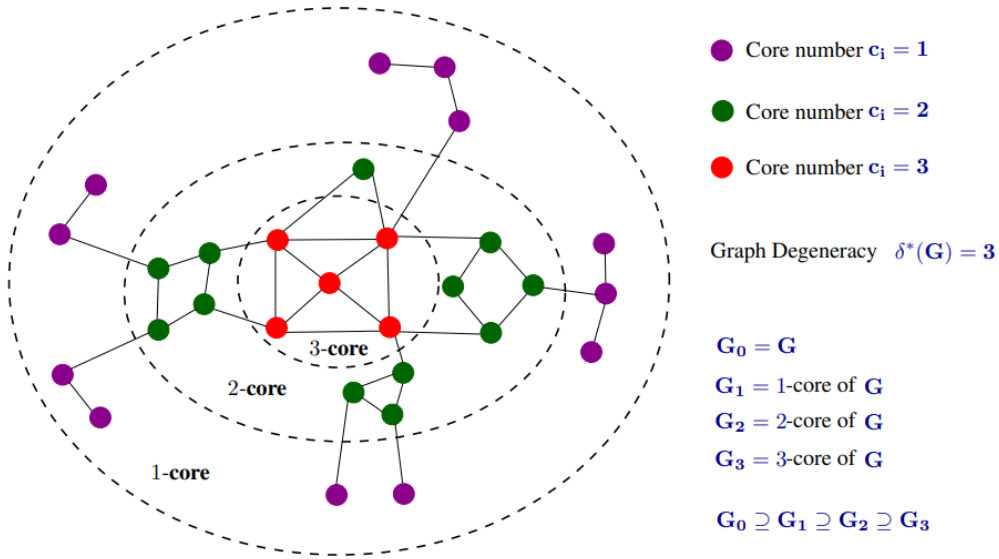
k-core Decomposition

- Start with $k = 1$ and iterate until all nodes are removed.
- For each k,remove recursively all nodes with degree less than a threshold k.
- Each retrieved subgraph indicates a progressively stronger clique that can support the spreading of a cascade.



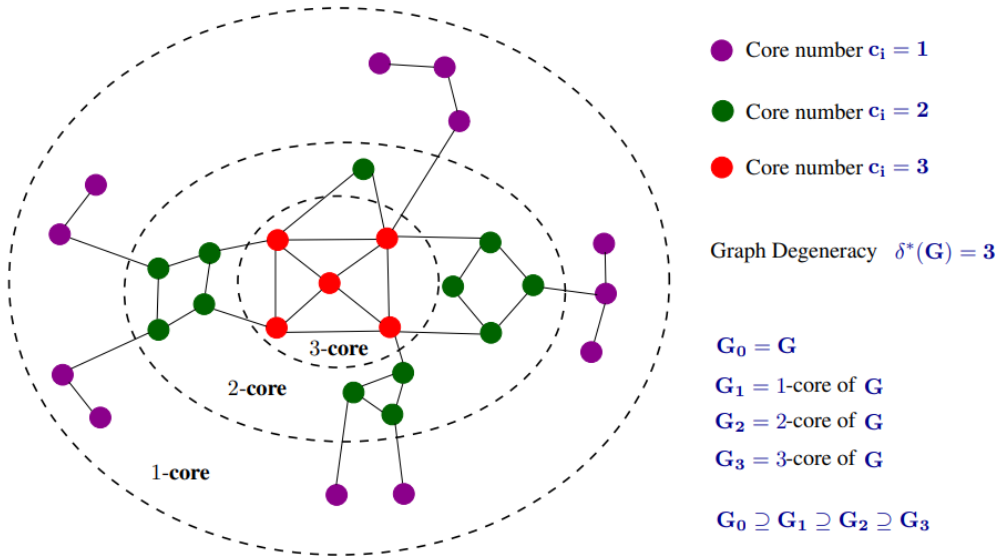
k-core Decomposition

- Start with $k = 1$ and iterate until all nodes are removed.
- For each k,remove recursively all nodes with degree less than a threshold k.
- Each retrieved subgraph indicates a progressively stronger clique that can support the spreading of a cascade.



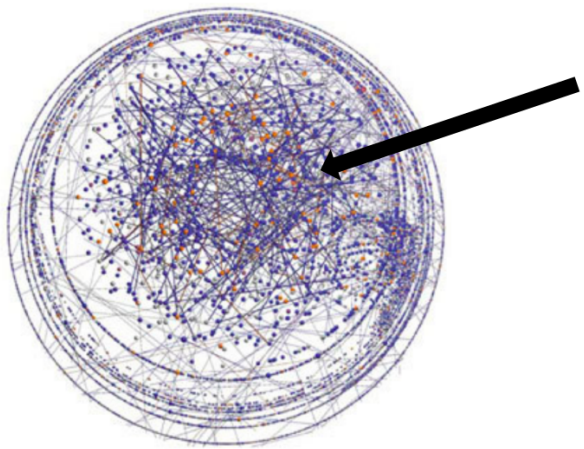
k-core Decomposition

- Start with $k = 1$ and iterate until all nodes are removed.
- For each k,remove recursively all nodes with degree less than a threshold k.
- Each retrieved subgraph indicates a progressively stronger clique that can support the spreading of a cascade.



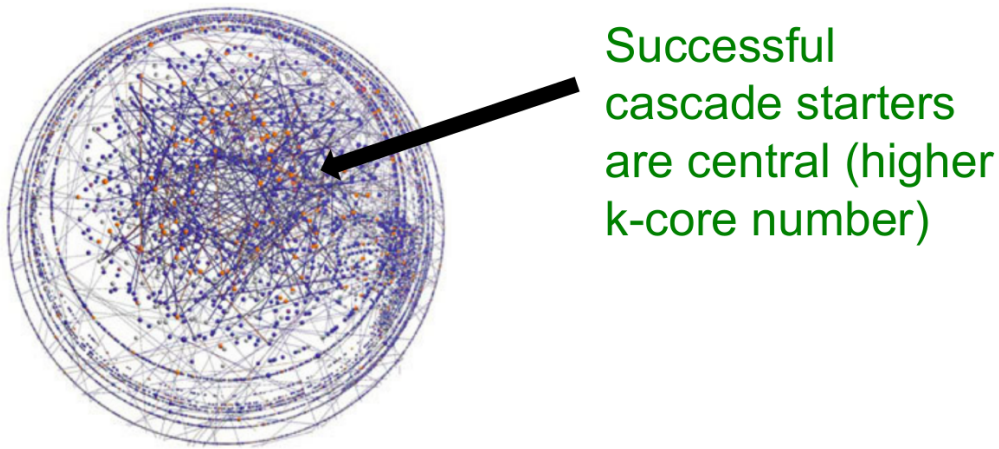
Who Starts the Successful Cascades?

- K-core decomposition of a following network.
- Indicated nodes have higher k-core values and indeed start the most successful cascades.
- A user needs to be central and connected to other central nodes to be an influencer.

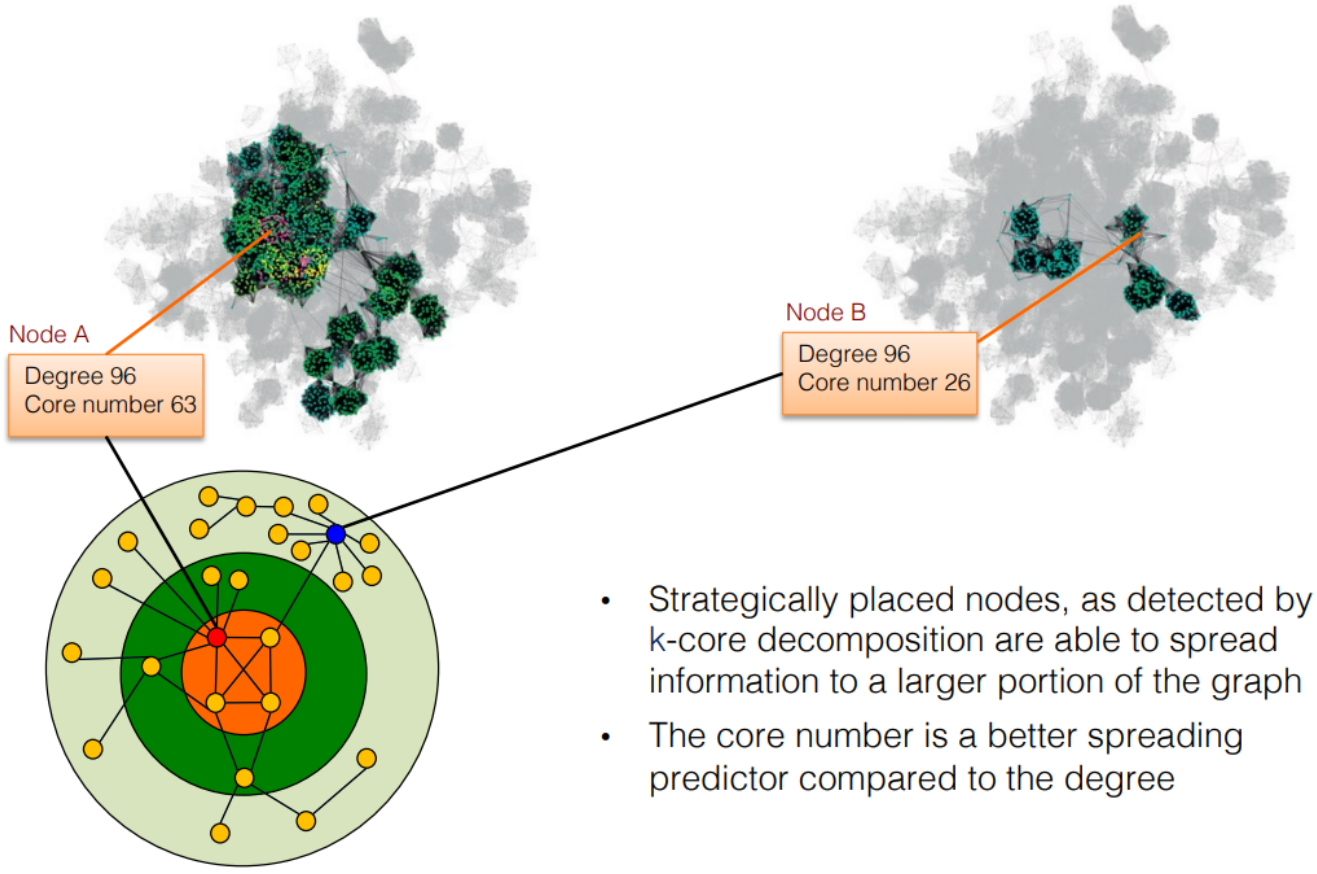


Who Starts the Successful Cascades?

- K-core decomposition of a following network.
- Indicated nodes have higher k-core values and indeed start the most successful cascades.
- A user needs to be central and connected to other central nodes to be an influencer.

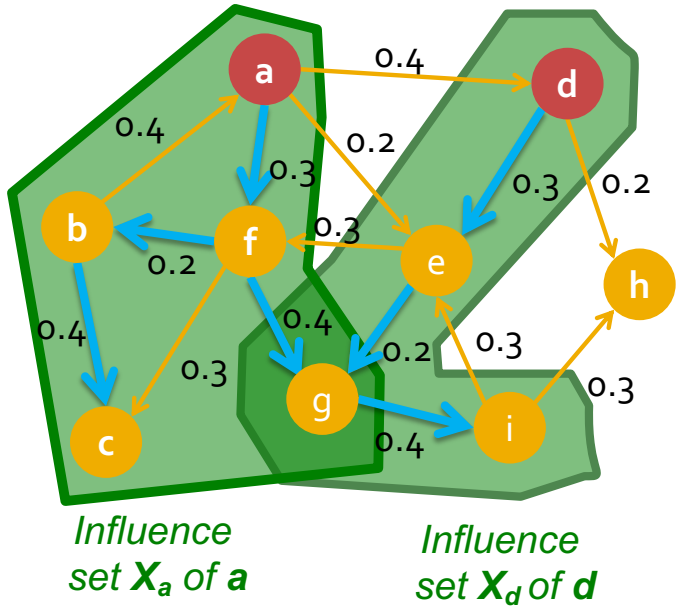


Who Starts the Successful Cascades? [Kitsak et al., Nature Physics '10]



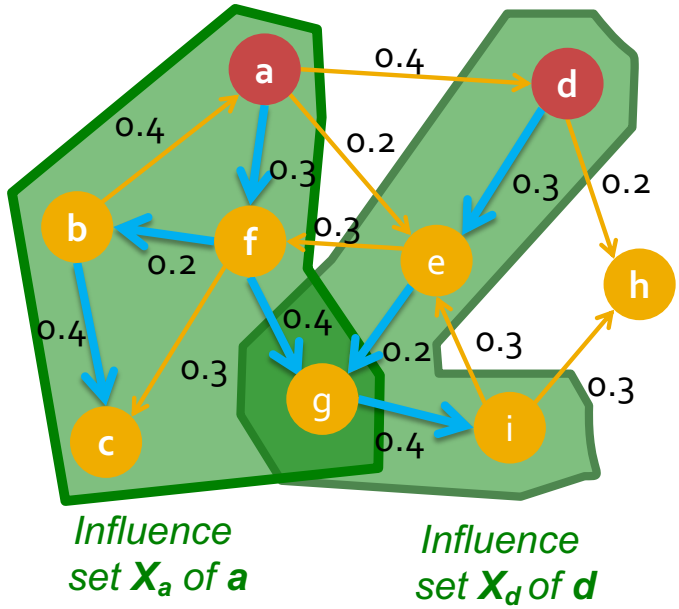
- Strategically placed nodes, as detected by the k-core decomposition are able to spread information to a larger portion of the graph
- The core number is a better spreading predictor compared to the degree

Most Influential Set



- Most influential set of size k under a diffusion model f: set S of k nodes producing largest expected cascade size $f(S)$ if activated
- Optimisation problem $\max_S f(S)$
- Why "expected cascade size"? as said above, $f(a)$ is a result of a random process. So in practice we compute $f(a)$ for many random realisations and then maximise the "average" value $f(S)$.

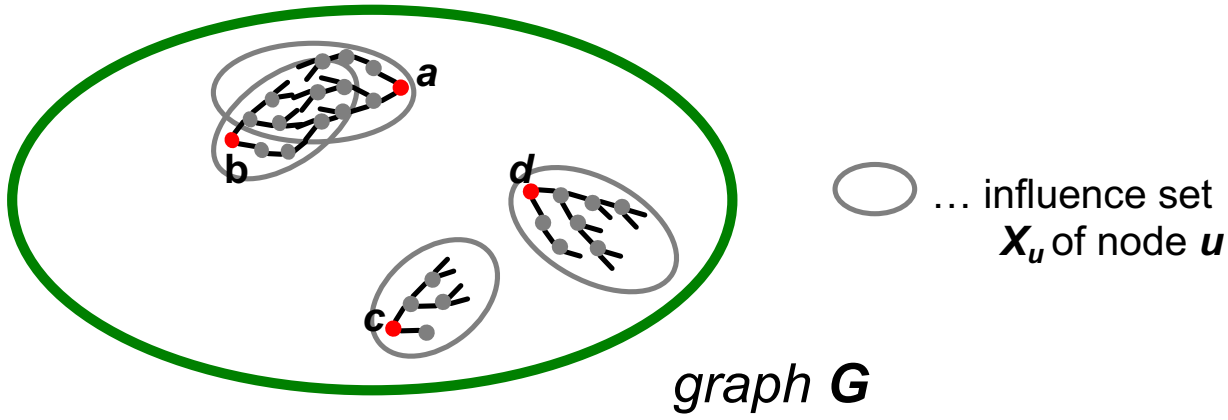
Most Influential Set



- Most influential set of size k under a diffusion model f: set S of k nodes producing largest expected cascade size $f(S)$ if activated
- Optimisation problem $\max_S f(S)$
- Why "expected cascade size"? as said above, $f(a)$ is a result of a random process. So in practice we compute $f(a)$ for many random realisations and then maximise the "average" value $f(S)$.

Most Influential Set

- S is an initial active set
- $f(S)$: the expected size of the final active set, i.e., $f(S)$ is the size of the union of X_u: $f(S) = |\cup_{u \in S} X_u|$



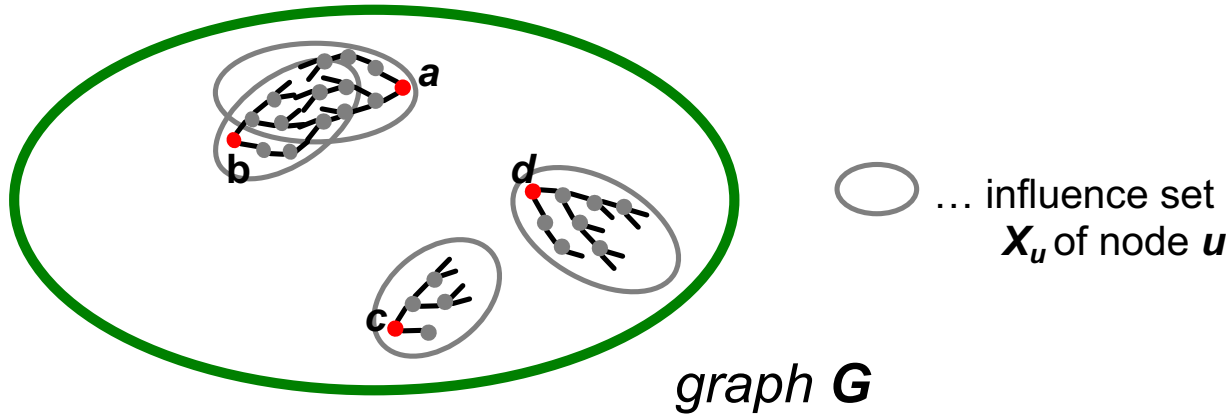
- Set S is more influential if $f(S)$ is larger.

$$f(\{a, b\}) < f(\{a, c\}) < f(\{a, d\})$$

- Hence heuristics are not enough!

Most Influential Set

- S is an initial active set
- $f(S)$: the expected size of the final active set, i.e., $f(S)$ is the size of the union of X_u: $f(S) = |\cup_{u \in S} X_u|$



- Set S is more influential if $f(S)$ is larger.

$$f(\{a, b\}) < f(\{a, c\}) < f(\{a, d\})$$

- Hence heuristics are not enough!

Most Influential Set

- Most influential set of size k: set S of k nodes producing largest expected cascade size $f(S)$ if activated
- Optimisation problem:

$$\max_S f(S)$$

- How hard is this problem? NP-Complete!
- Can be shown as a reduction from the set cover problem.
- Map each subset to the influence spread of a candidate seed set and the universe to the graph. Excellent proof sketch and notes can be found online[1].
- There exists an approximation algorithm!

[1] https://snap.stanford.edu/class/cs224w-2019/handouts/CS224W_Influence_Maximization_Handout.pdf

Most Influential Set

- Most influential set of size k: set S of k nodes producing largest expected cascade size $f(S)$ if activated
- Optimisation problem:

$$\max_S f(S)$$

- How hard is this problem? NP-Complete!
- Can be shown as a reduction from the set cover problem.
- Map each subset to the influence spread of a candidate seed set and the universe to the graph. Excellent proof sketch and notes can be found online[1].
- There exists an approximation algorithm!

[1] https://snap.stanford.edu/class/cs224w-2019/handouts/CS224W_Influence_Maximization_Handout.pdf

Most Influential Set

- Most influential set of size k: set S of k nodes producing largest expected cascade size $f(S)$ if activated
- Optimisation problem:

$$\max_S f(S)$$

- How hard is this problem? NP-Complete!
- Can be shown as a reduction from the set cover problem.
- Map each subset to the influence spread of a candidate seed set and the universe to the graph. Excellent proof sketch and notes can be found online[1].
- There exists an approximation algorithm!

[1] https://snap.stanford.edu/class/cs224w-2019/handouts/CS224W_Influence_Maximization_Handout.pdf

Most Influential Set

- Most influential set of size k: set S of k nodes producing largest expected cascade size $f(S)$ if activated
- Optimisation problem:

$$\max_S f(S)$$

- How hard is this problem? NP-Complete!
- Can be shown as a reduction from the set cover problem.
- Map each subset to the influence spread of a candidate seed set and the universe to the graph. Excellent proof sketch and notes can be found online[1].
- There exists an approximation algorithm!

[1] https://snap.stanford.edu/class/cs224w-2019/handouts/CS224W_Influence_Maximization_Handout.pdf

The Greedy

Consider a Greedy Hill Climbing algorithm to find S:

- Input: the influence spread $f(u) = \sigma(u)$ of each node u, $\sigma(u) = |\{v_1, v_2, \dots\}|$. If we activate u, nodes in $\{v_1, v_2, \dots\}$ will eventually get active.
- Algorithm: at each iteration i, activate the node u that gives the largest marginal gain

$$\max_u f(S_{i-1} \cup \{u\}) - f(S_{i-1})$$

The Greedy

Greedy Hill Climbing algorithm:

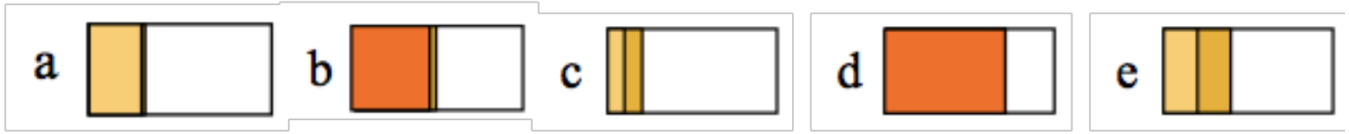
- Start with $S_0 = \{\}$
- For $i = 1, \ldots, k$
  - Activate the node u that $\max_u f(S_{i-1} \cup \{u\})$
  - $S_i = S_{i-1} \cup \{u\}$

- Example:
  1. Evaluate $f(\{a\}), \ldots f(\{e\})$, pick argmax of them
  2. Evaluate $f(\{d, a\}), \ldots f(\{d, e\})$, pick argmax of them
  3. Evaluate $f(\{d, b, a\}), \ldots f(\{d, b, e\})$, pick argmax of them

The Greedy

Greedy Hill Climbing algorithm:

- Start with $S_0 = \{\}$
- For $i = 1, \ldots, k$
  - Activate the node u that $\max_u f(S_{i-1} \cup \{u\})$
  - $S_i = S_{i-1} \cup \{u\}$

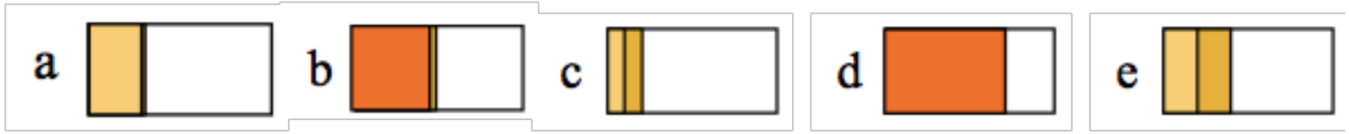


- Example:
  1. Evaluate $f(\{a\}), \ldots f(\{e\})$, pick argmax of them
  2. Evaluate $f(\{d, a\}), \ldots f(\{d, e\})$, pick argmax of them
  3. Evaluate $f(\{d, b, a\}), \ldots f(\{d, b, e\})$, pick argmax of them

The Greedy

Greedy Hill Climbing algorithm:

- Start with $S_0 = \{\}$
- For $i = 1, \ldots, k$
  - Activate the node u that $\max_u f(S_{i-1} \cup \{u\})$
  - $S_i = S_{i-1} \cup \{u\}$

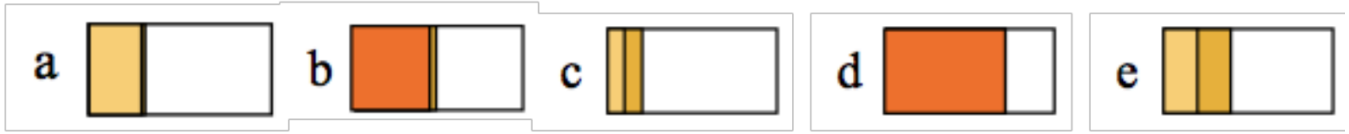


- Example:
  1. Evaluate $f(\{a\}), \ldots f(\{e\})$, pick argmax of them
  2. Evaluate $f(\{d, a\}), \ldots f(\{d, e\})$, pick argmax of them
  3. Evaluate $f(\{d, b, a\}), \ldots f(\{d, b, e\})$, pick argmax of them

The Greedy

Greedy Hill Climbing algorithm:

- Start with $S_0 = \{\}$
- For $i = 1, \ldots, k$
  - Activate the node u that $\max_u f(S_{i-1} \cup \{u\})$
  - $S_i = S_{i-1} \cup \{u\}$



- Example:
  1. Evaluate $f(\{a\}), \ldots f(\{e\})$, pick argmax of them
  2. Evaluate $f(\{d, a\}), \ldots f(\{d, e\})$, pick argmax of them
  3. Evaluate $f(\{d, b, a\}), \ldots f(\{d, b, e\})$, pick argmax of them

The Approximation Guarantees

- Hill climbing produces a solution S where $f(S) \geq (1 - \frac{1}{e}) \times f(OPT)$ (OPT is the optimal solution)
- This claim holds for f with the following two properties
- σ is monotone:
- This means that activating more nodes can not diminish the influence spread.

The Approximation Guarantees

- Hill climbing produces a solution S where $f(S) \geq (1 - \frac{1}{e}) \times f(OPT)$ (OPT is the optimal solution)
- This claim holds for f with the following two properties
- σ is monotone:

$$\sigma(S \cup \{v\}) \geq \sigma(S), \forall v \in V, \forall S \subseteq V$$

- This means that activating more nodes can not diminish the influence spread.

The Approximation Guarantees

- σ is submodular:

$$\underbrace{\sigma(S \cup \{v\}) - \sigma(S)}_{\text{Marginal gain of adding a node to a small set}} \geq \underbrace{\sigma(T \cup \{v\}) - \sigma(T)}_{\text{Marginal gain of adding a node to a large set}} \qquad S \subseteq T \subseteq V$$

- This means that adding an element to a set gives less improvement than adding it to one of its subsets.
- In other words, if we think of T as an S of previous iterations, there is a diminishing returns property:

The Approximation Guarantees

- σ is submodular:

$$\underbrace{\sigma(S \cup \{v\}) - \sigma(S)}_{\text{Marginal gain of adding a node to a small set}} \geq \underbrace{\sigma(T \cup \{v\}) - \sigma(T)}_{\text{Marginal gain of adding a node to a large set}} \qquad S \subseteq T \subseteq V$$

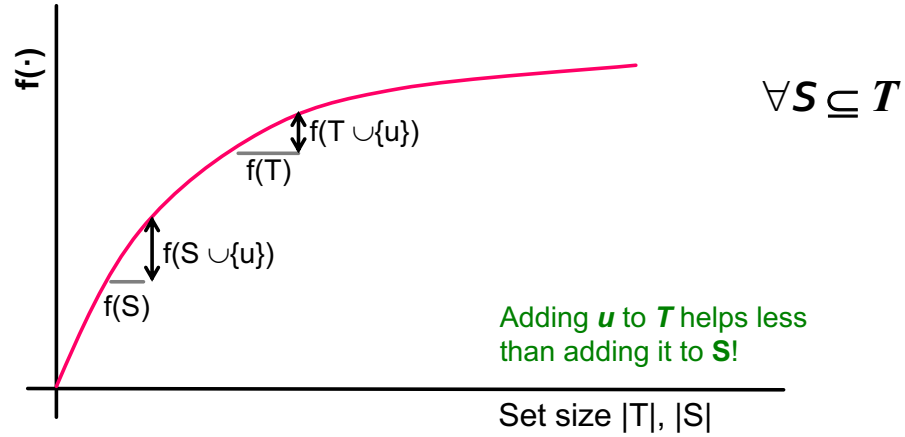
- This means that adding an element to a set gives less improvement than adding it to one of its subsets.
- In other words, if we think of T as an S of previous iterations, there is a diminishing returns property:

The Approximation Guarantees

- σ is submodular:

$$\underbrace{\sigma(S \cup \{v\}) - \sigma(S)}_{\text{Marginal gain of adding a node to a small set}} \geq \underbrace{\sigma(T \cup \{v\}) - \sigma(T)}_{\text{Marginal gain of adding a node to a large set}} \qquad S \subseteq T \subseteq V$$

- This means that adding an element to a set gives less improvement than adding it to one of its subsets.
- In other words, if we think of T as an S of previous iterations, there is a diminishing returns property:



$\forall S \subseteq T$

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

Accelerating Greedy

Greedy hill climbing is slow!

- At each iteration we need to re-evaluate marginal gains of all nodes.
- Number of influence spread computations is $O(|V||S|)$.
- Computing the influence spread using simulation of diffusion models is itself very costly $O(R|E|)$, where R is the number of simulations!

Cost Effective Lazy Forward

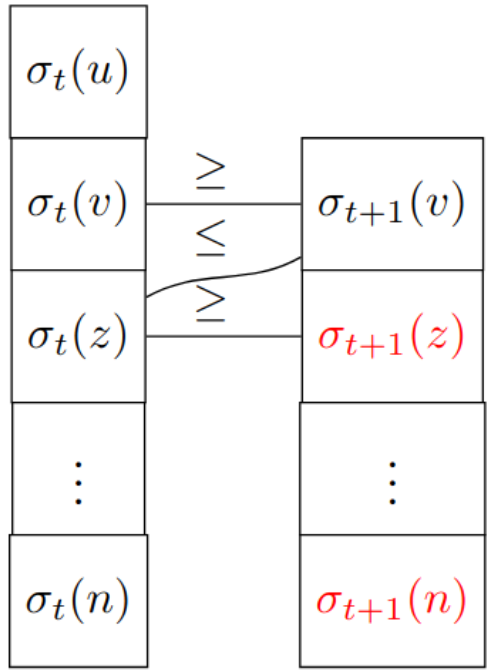
Assume we have the influence spread σ for all nodes in iteration t. Also assume $\sigma_t(v) = \sigma(S_t \cup \{v\}) - \sigma(S_t)$.

- By definition, $\sigma_{t+1}(v)$ of node v in $t+1$, will never be more then the previous iteration $\sigma_t(v)$: $\sigma_{t+1}(v) \leq \sigma_t(v)$ (1).
- Assuming there exist nodes z with $\sigma_t(z) \leq \sigma_{t+1}(v)$ (2).
- By (1) & (2) we have $\sigma_{t+1}(z) \leq \sigma_t(z) \leq \sigma_{t+1}(v)$.

Cost Effective Lazy Forward

Assume we have the influence spread σ for all nodes in iteration t. Also assume $\sigma_t(v) = \sigma(S_t \cup \{v\}) - \sigma(S_t)$.

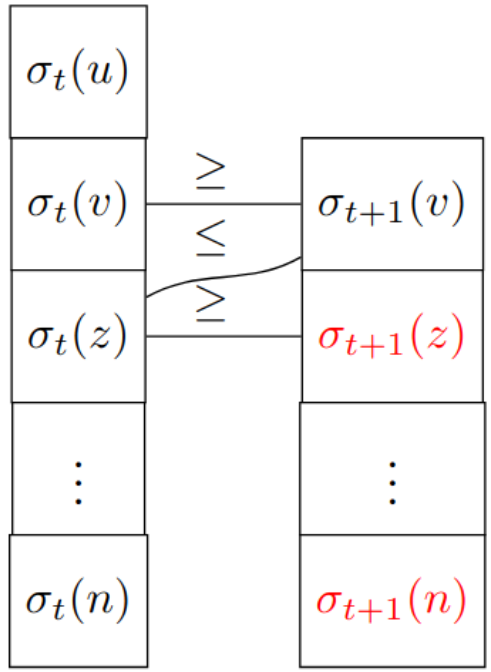
- By definition, $\sigma_{t+1}(v)$ of node v in $t+1$, will never be more then the previous iteration $\sigma_t(v)$: $\sigma_{t+1}(v) \leq \sigma_t(v)$ (1).
- Assuming there exist nodes z with $\sigma_t(z) \leq \sigma_{t+1}(v)$ (2).
- By (1) & (2) we have $\sigma_{t+1}(z) \leq \sigma_t(z) \leq \sigma_{t+1}(v)$.



Cost Effective Lazy Forward

Assume we have the influence spread σ for all nodes in iteration t. Also assume $\sigma_t(v) = \sigma(S_t \cup \{v\}) - \sigma(S_t)$.

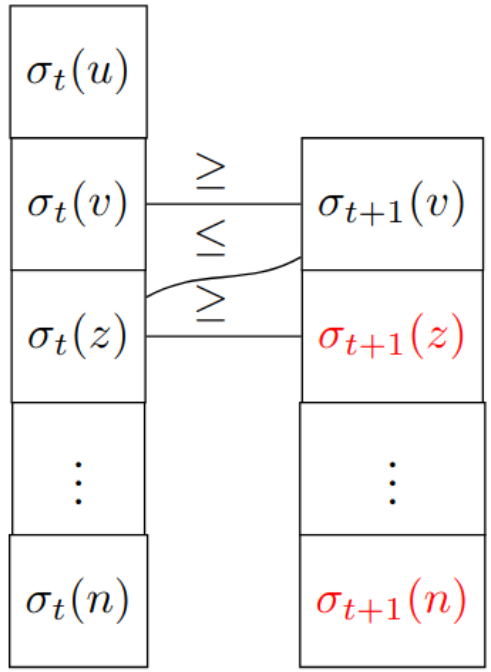
- By definition, $\sigma_{t+1}(v)$ of node v in $t+1$, will never be more then the previous iteration $\sigma_t(v)$: $\sigma_{t+1}(v) \leq \sigma_t(v)$ (1).
- Assuming there exist nodes z with $\sigma_t(z) \leq \sigma_{t+1}(v)$ (2).
- By (1) & (2) we have $\sigma_{t+1}(z) \leq \sigma_t(z) \leq \sigma_{t+1}(v)$.



Cost Effective Lazy Forward

Assume we have the influence spread σ for all nodes in iteration t. Also assume $\sigma_t(v) = \sigma(S_t \cup \{v\}) - \sigma(S_t)$.

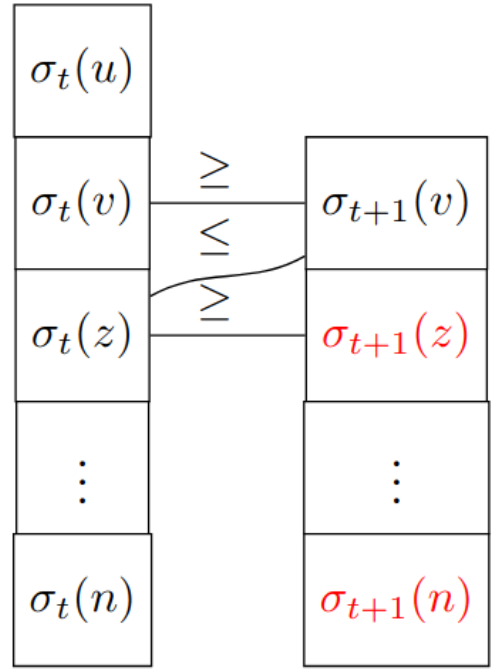
- By definition, $\sigma_{t+1}(v)$ of node v in $t+1$, will never be more then the previous iteration $\sigma_t(v)$: $\sigma_{t+1}(v) \leq \sigma_t(v)$ (1).
- Assuming there exist nodes z with $\sigma_t(z) \leq \sigma_{t+1}(v)$ (2).
- By (1) & (2) we have $\sigma_{t+1}(z) \leq \sigma_t(z) \leq \sigma_{t+1}(v)$.



Cost Effective Lazy Forward

Assume we have the influence spread σ for all nodes in iteration t. Also assume $\sigma_t(v) = \sigma(S_t \cup \{v\}) - \sigma(S_t)$.

- By definition, $\sigma_{t+1}(v)$ of node v in $t+1$, will never be more then the previous iteration $\sigma_t(v)$: $\sigma_{t+1}(v) \leq \sigma_t(v)$ (1).
- Assuming there exist nodes z with $\sigma_t(z) \leq \sigma_{t+1}(v)$ (2).
- By (1) & (2) we have $\sigma_{t+1}(z) \leq \sigma_t(z) \leq \sigma_{t+1}(v)$.



Hence we do not need to compute $\sigma_{t+1}(z)$ since it will always be less then $\sigma_{t+1}(v)$!

How can we utilize this observation to make an efficient algorithm?

Cost Effective Lazy Forward

Develop greedy with lazy evaluations!

- Initially compute the influence spread of every node, sort the list in descending order, and choose the first node as seed.
- At each iteration t, start computing the new influence spread from the top.
- If the current spread of the top node is bigger than the previous spread of the second node e.g. $\sigma_{t+1}(v) \geq \sigma_t(z)$, v is our new seed and we don't need any other computations!
- This happens because $\sigma_t(z) \geq \sigma_{t+1}(z)$ by definition and $\sigma_t(z) \geq \sigma_t(n) \forall n \in G$ because the list is sorted.
- Compute $\sigma_{t+1}(z)$ iff $\sigma_{t+1}(v) \leq \sigma_t(z)$, update the list, resort, and continue this process until the top spread is bigger than the second spread.

Cost Effective Lazy Forward

Develop greedy with lazy evaluations!

- Initially compute the influence spread of every node, sort the list in descending order, and choose the first node as seed.
- At each iteration t, start computing the new influence spread from the top.
- If the current spread of the top node is bigger than the previous spread of the second node e.g. $\sigma_{t+1}(v) \geq \sigma_t(z)$, v is our new seed and we don't need any other computations!
- This happens because $\sigma_t(z) \geq \sigma_{t+1}(z)$ by definition and $\sigma_t(z) \geq \sigma_t(n) \forall n \in G$ because the list is sorted.
- Compute $\sigma_{t+1}(z)$ iff $\sigma_{t+1}(v) \leq \sigma_t(z)$, update the list, resort, and continue this process until the top spread is bigger than the second spread.

Cost Effective Lazy Forward

Develop greedy with lazy evaluations!

- Initially compute the influence spread of every node, sort the list in descending order, and choose the first node as seed.
- At each iteration t, start computing the new influence spread from the top.
- If the current spread of the top node is bigger than the previous spread of the second node e.g. $\sigma_{t+1}(v) \geq \sigma_t(z)$, v is our new seed and we don't need any other computations!
- This happens because $\sigma_t(z) \geq \sigma_{t+1}(z)$ by definition and $\sigma_t(z) \geq \sigma_t(n) \forall n \in G$ because the list is sorted.
- Compute $\sigma_{t+1}(z)$ iff $\sigma_{t+1}(v) \leq \sigma_t(z)$, update the list, resort, and continue this process until the top spread is bigger than the second spread.

Cost Effective Lazy Forward

Develop greedy with lazy evaluations!

- Initially compute the influence spread of every node, sort the list in descending order, and choose the first node as seed.
- At each iteration t, start computing the new influence spread from the top.
- If the current spread of the top node is bigger than the previous spread of the second node e.g. $\sigma_{t+1}(v) \geq \sigma_t(z)$, v is our new seed and we don't need any other computations!
- This happens because $\sigma_t(z) \geq \sigma_{t+1}(z)$ by definition and $\sigma_t(z) \geq \sigma_t(n) \forall n \in G$ because the list is sorted.
- Compute $\sigma_{t+1}(z)$ iff $\sigma_{t+1}(v) \leq \sigma_t(z)$, update the list, resort, and continue this process until the top spread is bigger than the second spread.

Cost Effective Lazy Forward

Develop greedy with lazy evaluations!

- Initially compute the influence spread of every node, sort the list in descending order, and choose the first node as seed.
- At each iteration t, start computing the new influence spread from the top.
- If the current spread of the top node is bigger than the previous spread of the second node e.g. $\sigma_{t+1}(v) \geq \sigma_t(z)$, v is our new seed and we don't need any other computations!
- This happens because $\sigma_t(z) \geq \sigma_{t+1}(z)$ by definition and $\sigma_t(z) \geq \sigma_t(n) \forall n \in G$ because the list is sorted.
- Compute $\sigma_{t+1}(z)$ iff $\sigma_{t+1}(v) \leq \sigma_t(z)$, update the list, resort, and continue this process until the top spread is bigger than the second spread.

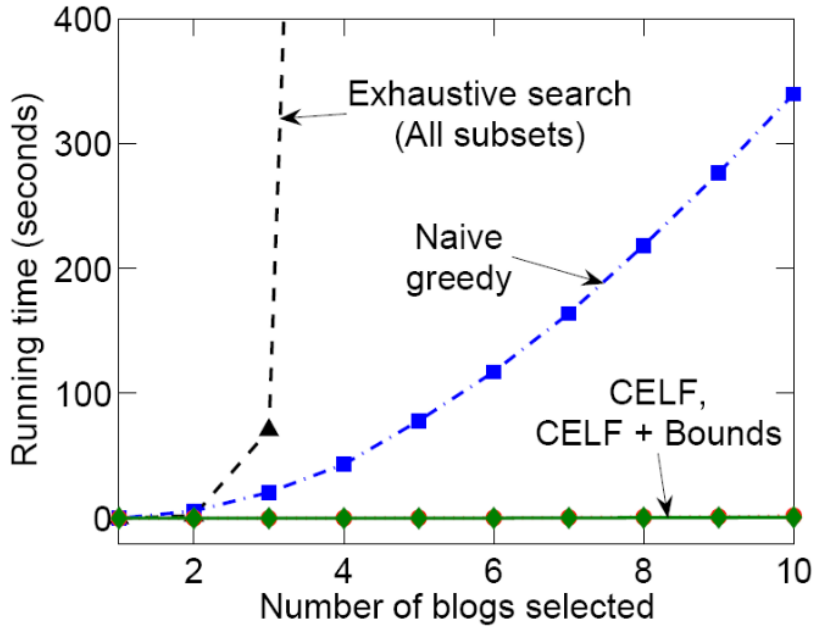
Cost Effective Lazy Forward

Develop greedy with lazy evaluations!

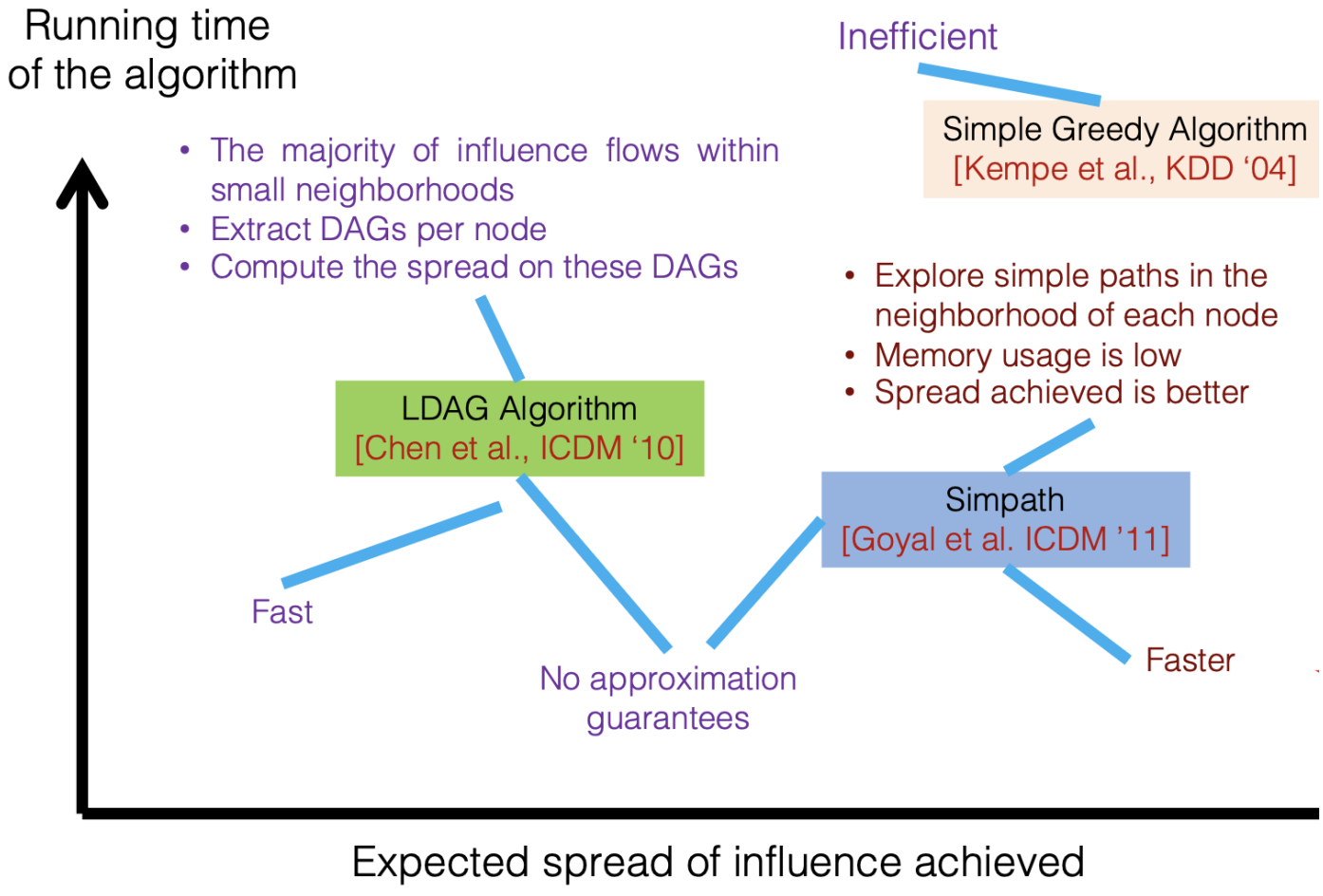
- Initially compute the influence spread of every node, sort the list in descending order, and choose the first node as seed.
- At each iteration t, start computing the new influence spread from the top.
- If the current spread of the top node is bigger than the previous spread of the second node e.g. $\sigma_{t+1}(v) \geq \sigma_t(z)$, v is our new seed and we don't need any other computations!
- This happens because $\sigma_t(z) \geq \sigma_{t+1}(z)$ by definition and $\sigma_t(z) \geq \sigma_t(n) \forall n \in G$ because the list is sorted.
- Compute $\sigma_{t+1}(z)$ iff $\sigma_{t+1}(v) \leq \sigma_t(z)$, update the list, resort, and continue this process until the top spread is bigger than the second spread.

CELF vs Greedy

CELF is up to 700 times faster then greedy in practice, having the same worst case complexity.



Heuristic Optimization Methods



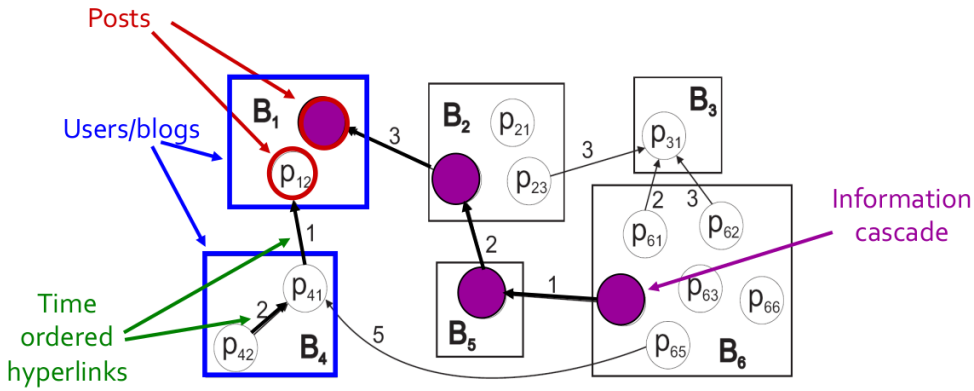
[Goyal et al. ICDM '11]

Influence Maximisation

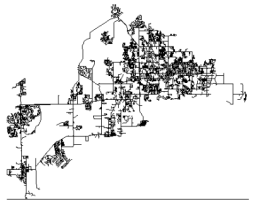
Open problems:

- Finding more general influence models.
- Deal with negative influence.
- Obtain more data (better models) about how activations occur in real networks.

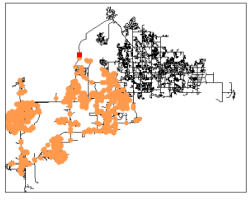
Other Use Cases



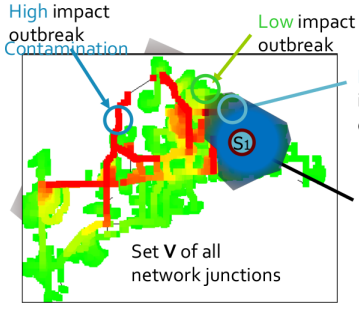
Which users/news sites should one follow to **detect cascades** as **effectively** as possible?



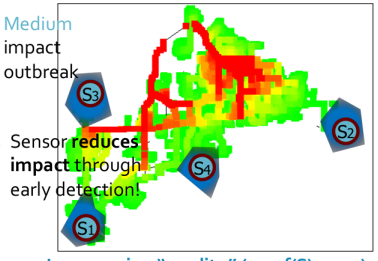
Water distribution network
(physical pipes and junctions)



Simulator of water consumption & flow
(built by Mech. Eng. people)



High sensing "quality" (e.g., f(S) = 0.9)



Low sensing "quality" (e.g. f(S)=0.01)

Leskovec et. al. 2007