Learning Social Networks, Actively and Passively

SFI Talk

Lev Reyzin
Yahoo! Research
(work done while at Yale University)

talk based on 2 papers, both with
Dana Angluin and James Aspnes

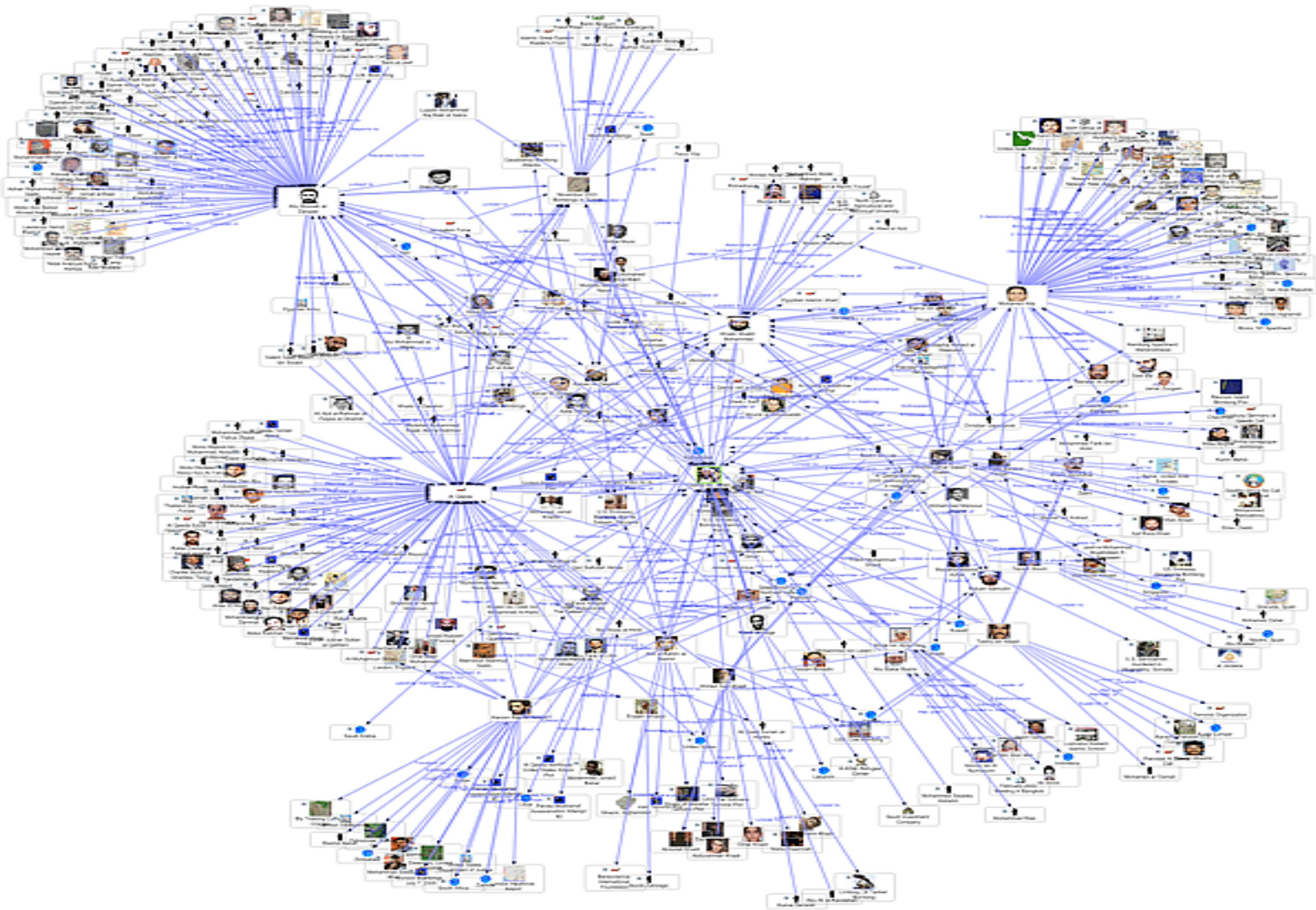
Learning Interaction Networks

- Reconstructing Evolutionary Trees via Distance Experiments
- Learning and Verifying Graphs (ie Genome Sequences with PCR)
- Learning Large-Alphabet Circuits (ie Gene Regulatory Networks)
- Learning Bayesian Networks
- Actively Learning Social Networks
- Passively Learning Social Networks

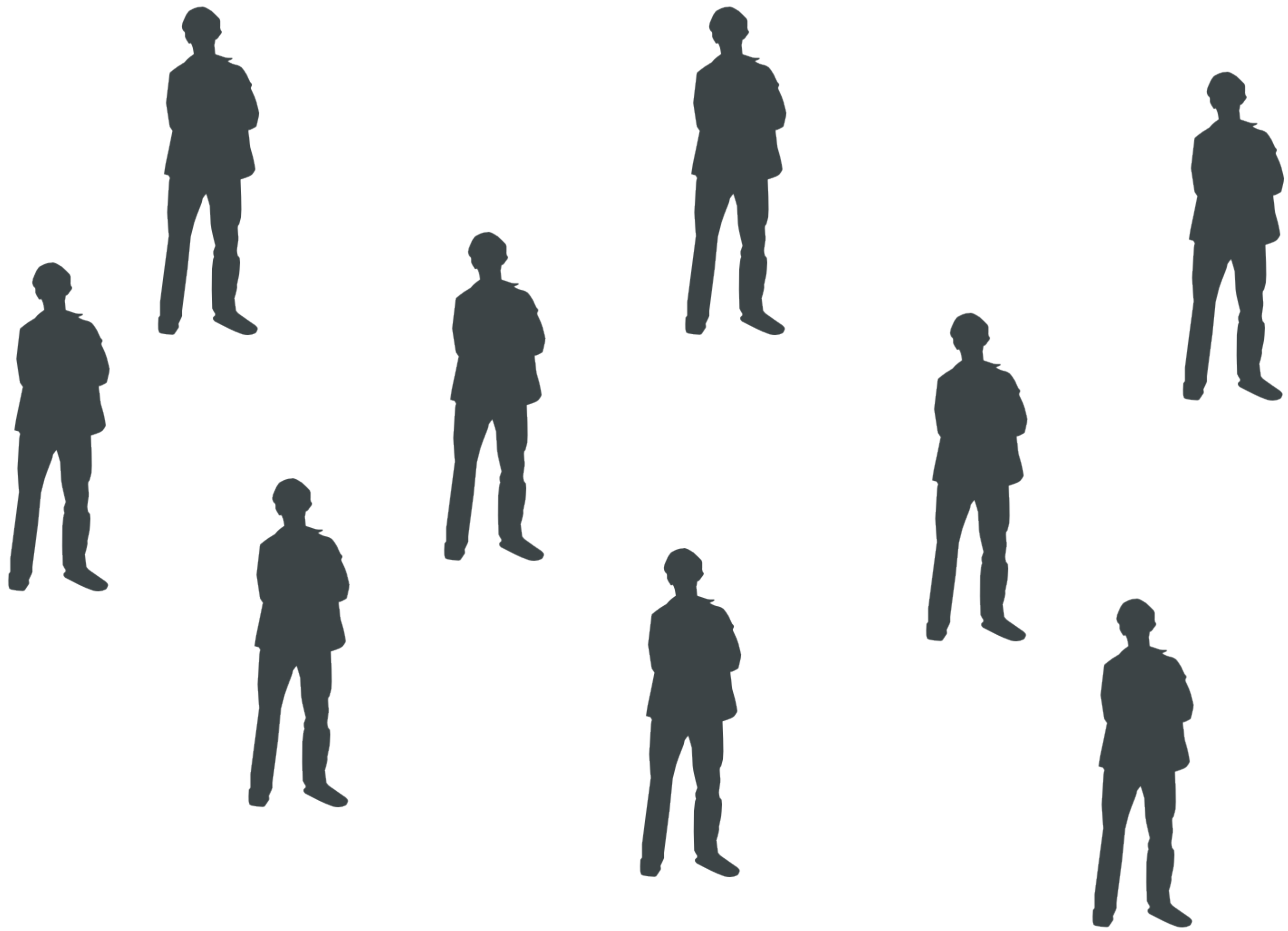
Learning Interaction Networks

- Reconstructing Evolutionary Trees via Distance Experiments
- Learning and Verifying Graphs (ie Genome Sequences with PCR)
- Learning Large-Alphabet Circuits (ie Gene Regulatory Networks)
- Learning Bayesian Networks
- Actively Learning Social Networks
- Passively Learning Social Networks

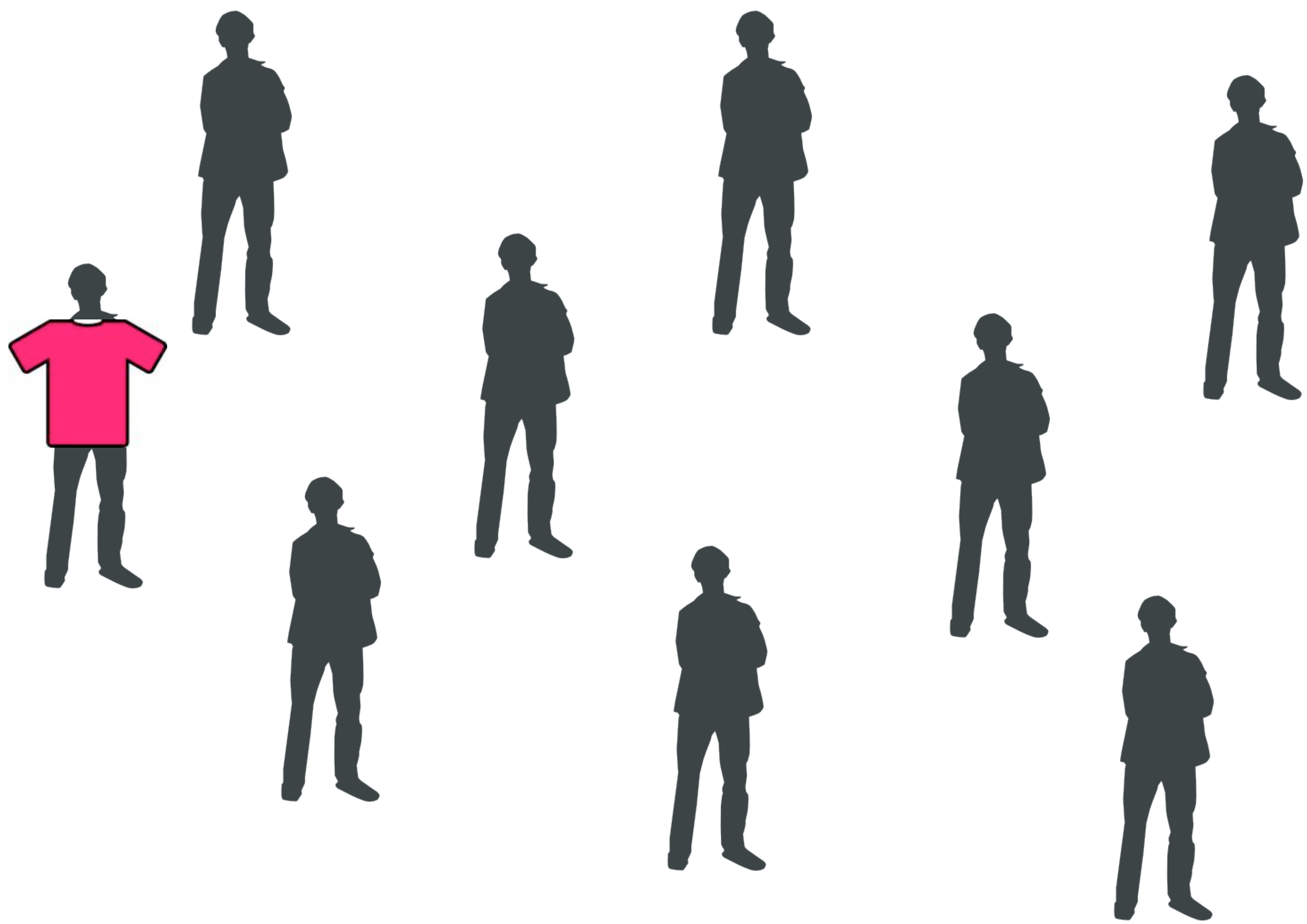
How Do We Learn Social Networks?



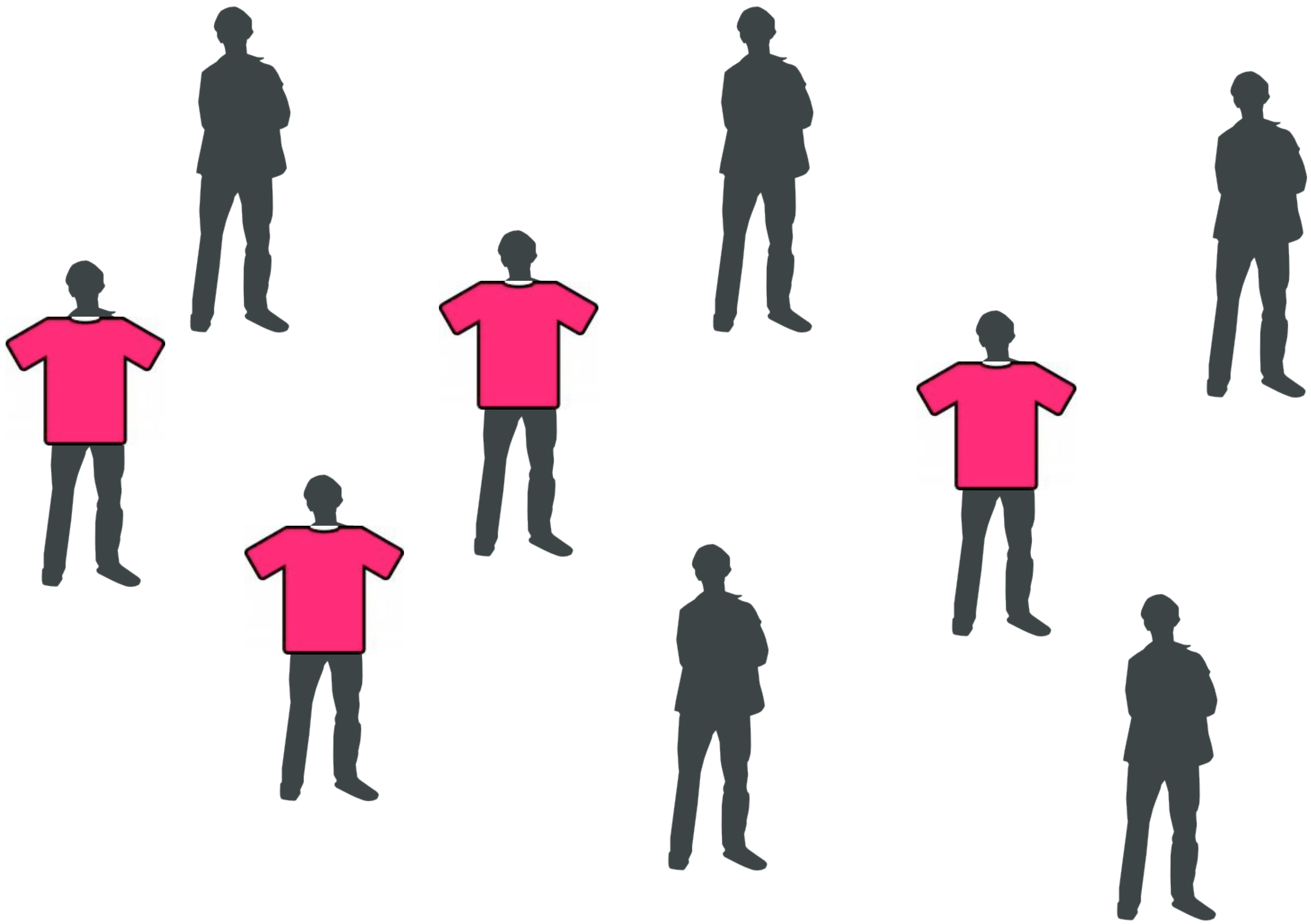
Trends Spreading through a Social Network



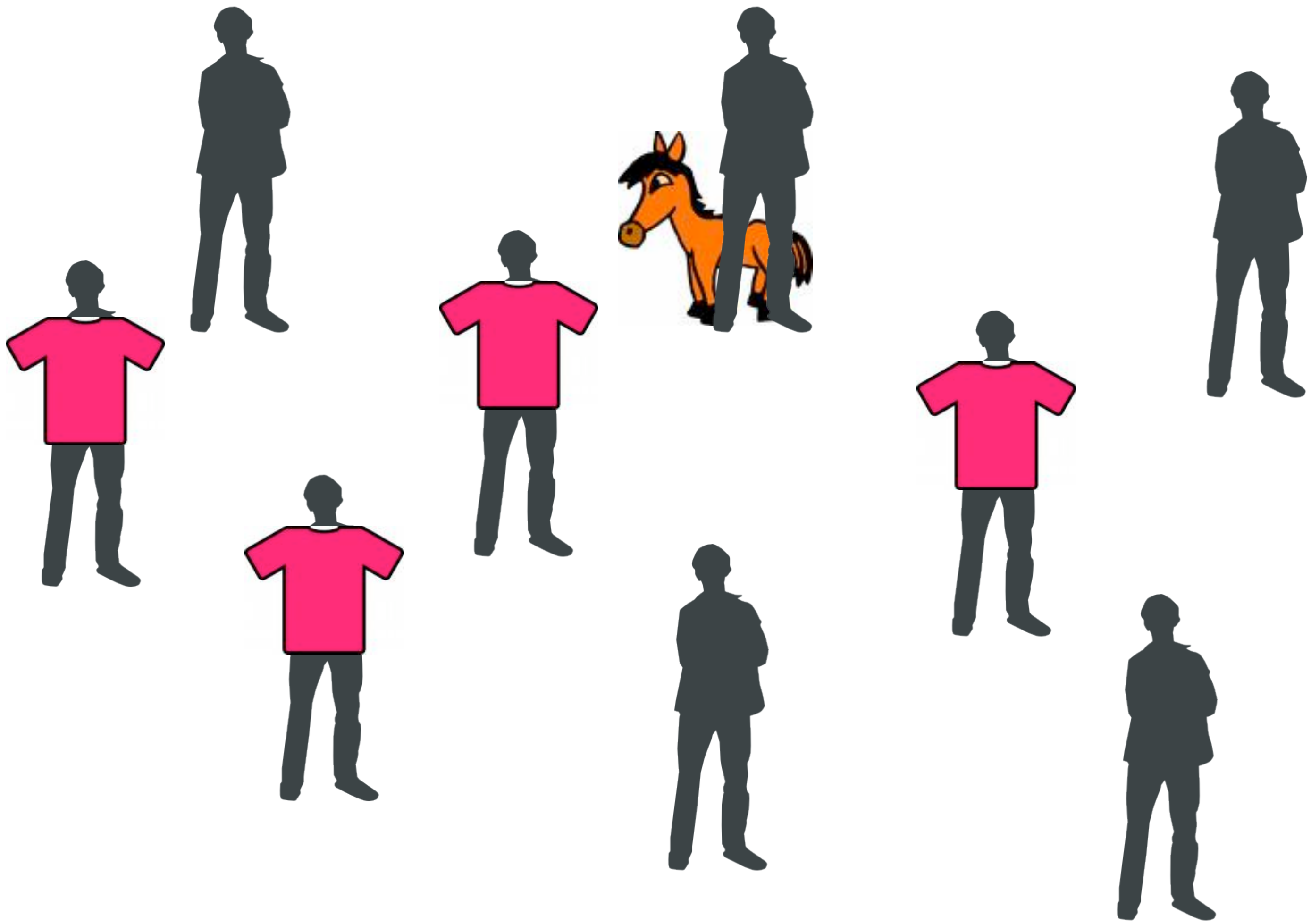
Trends Spreading through a Social Network



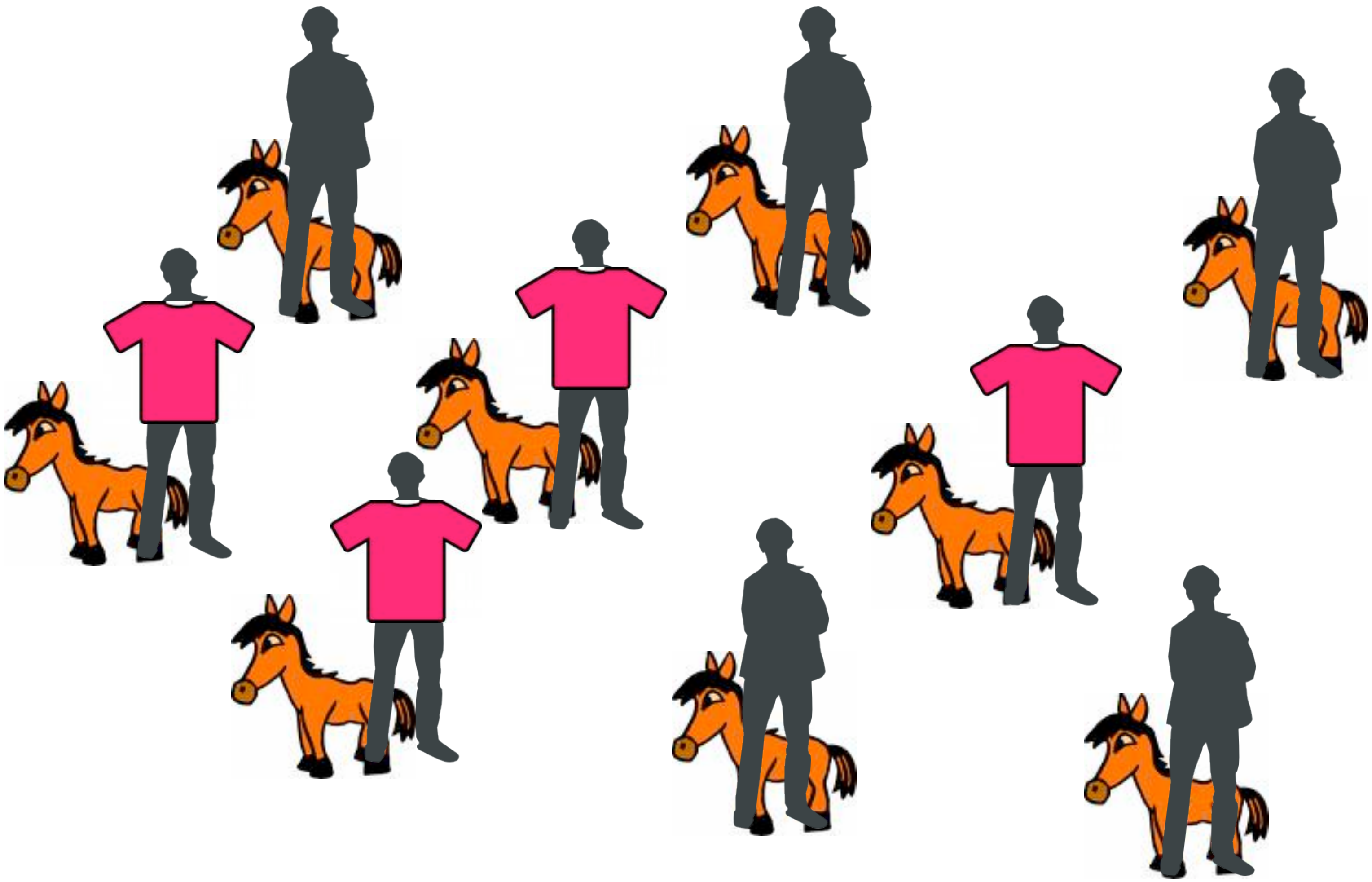
Trends Spreading through a Social Network



Trends Spreading through a Social Network



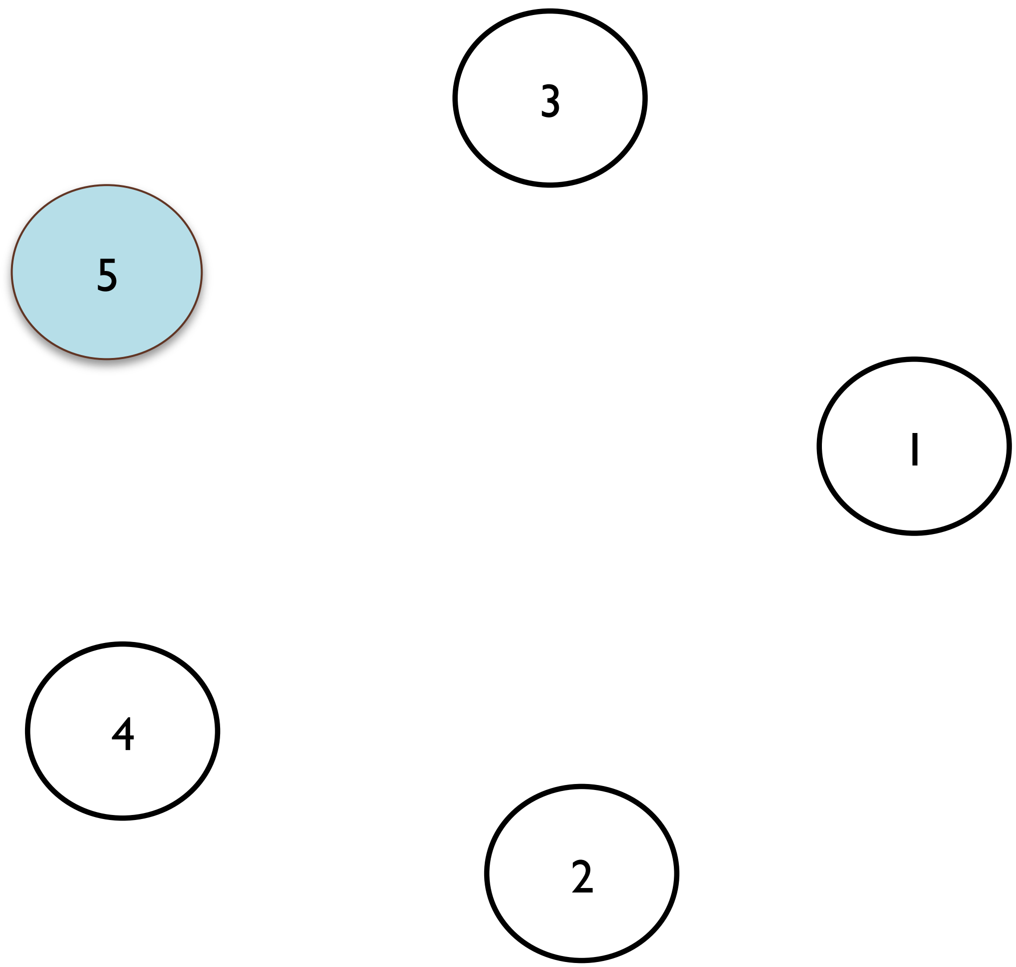
Trends Spreading through a Social Network



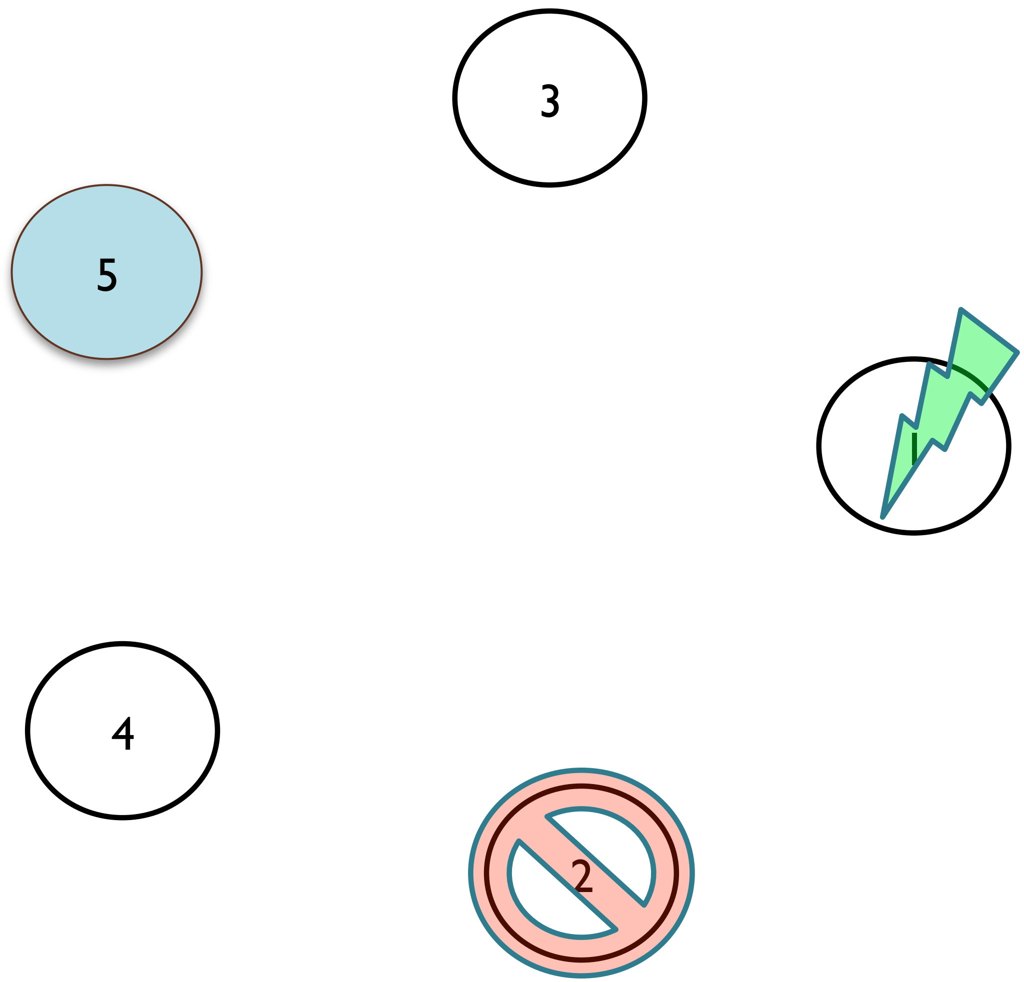
Value Injection Queries (VIQs) an Overview

- Model to study learning hidden circuits [AACW '06] – inspired by learning of gene regulatory networks.
- Allows for perturbing the circuit anywhere, but observing only its one output (ie phenotype, vote, $).
- In between input/output models and fully observable models.
- We apply VIQs to learning independent cascade social networks.

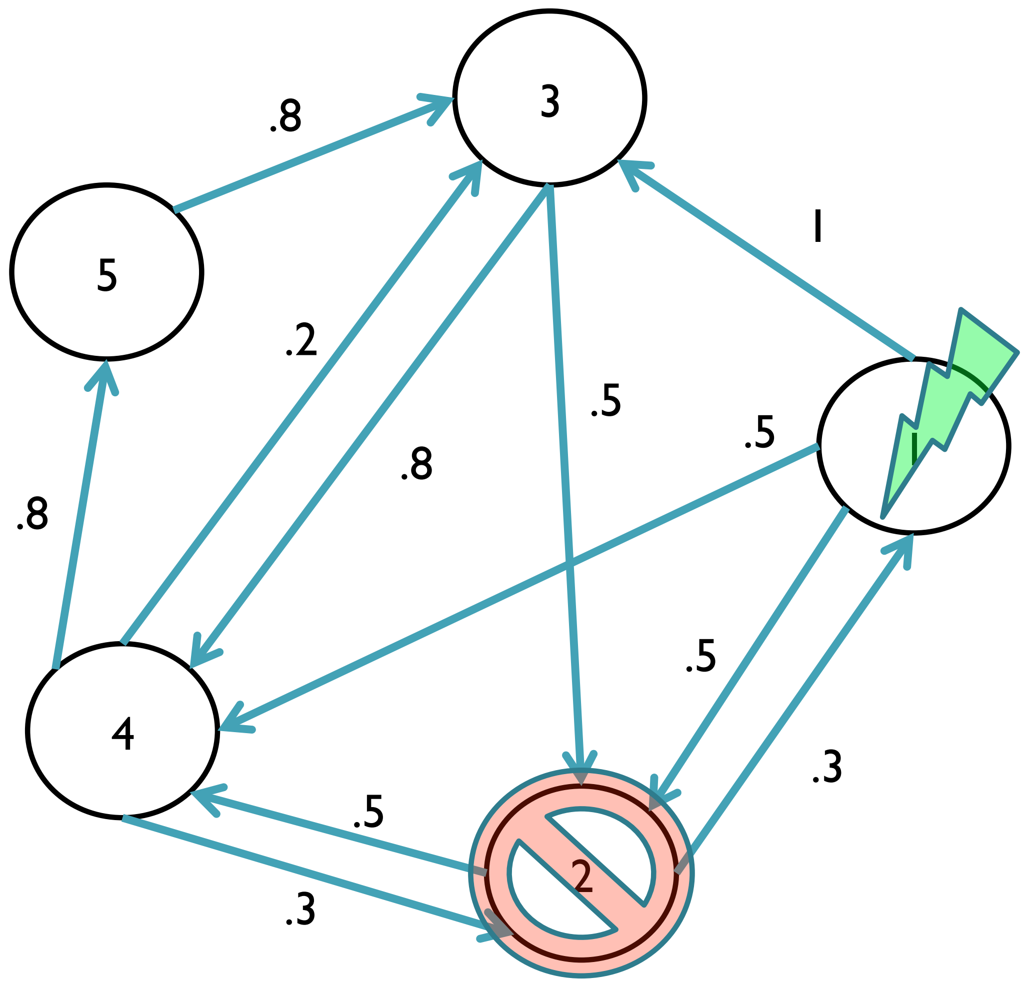
What the Learner Sees



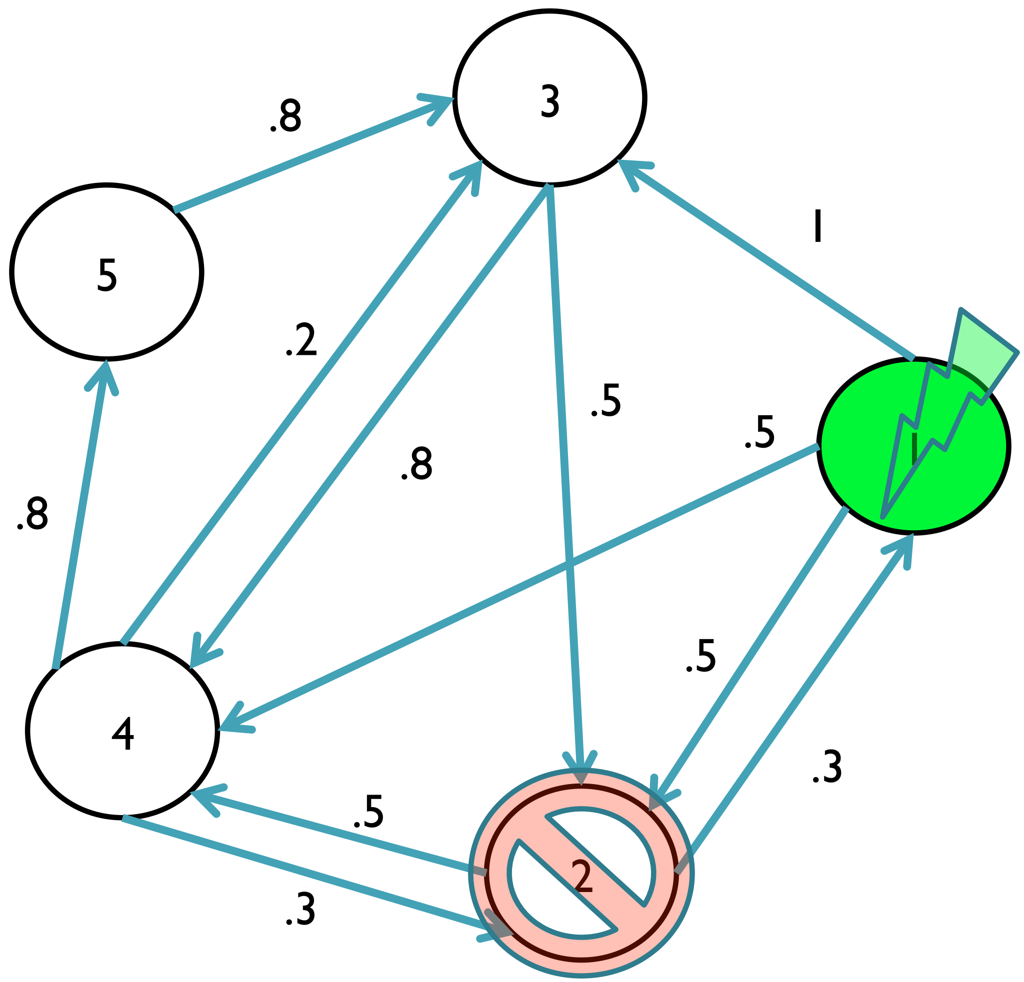
Activations and Suppressions



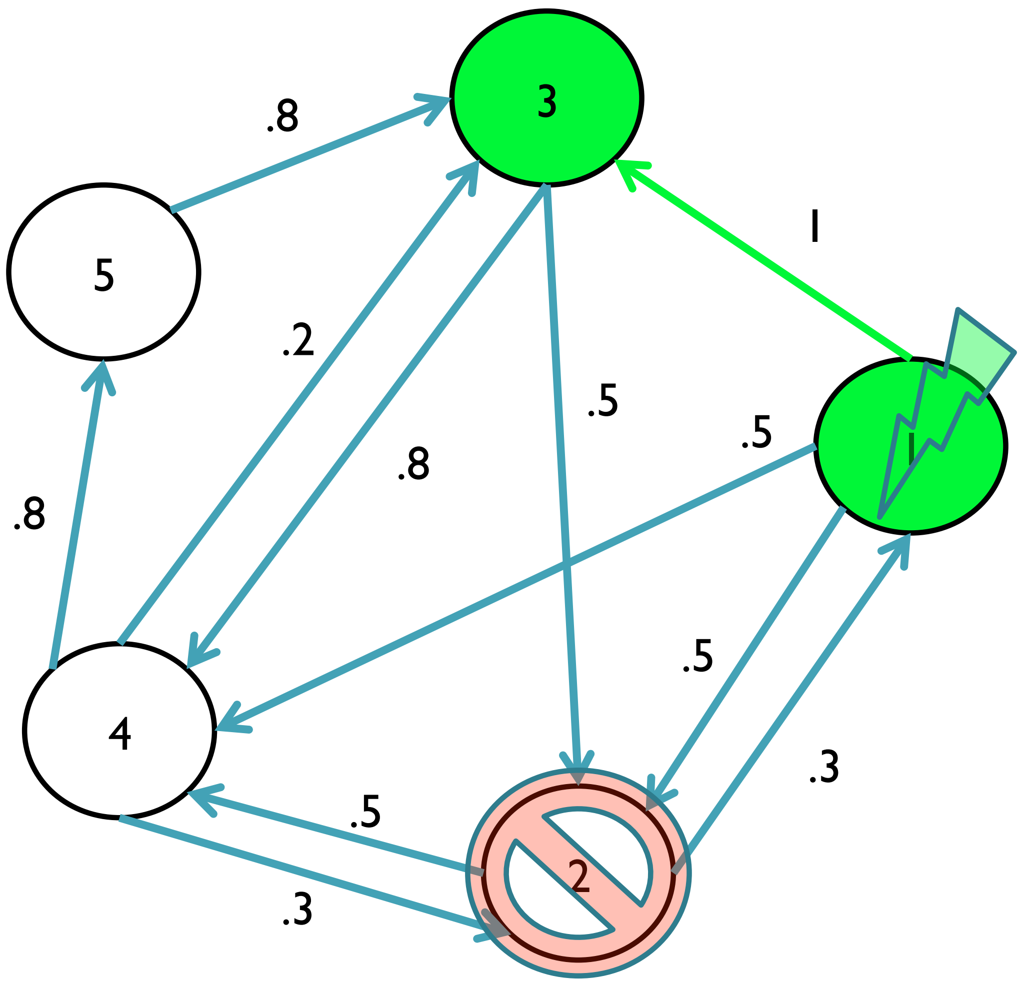
Activations and Suppressions



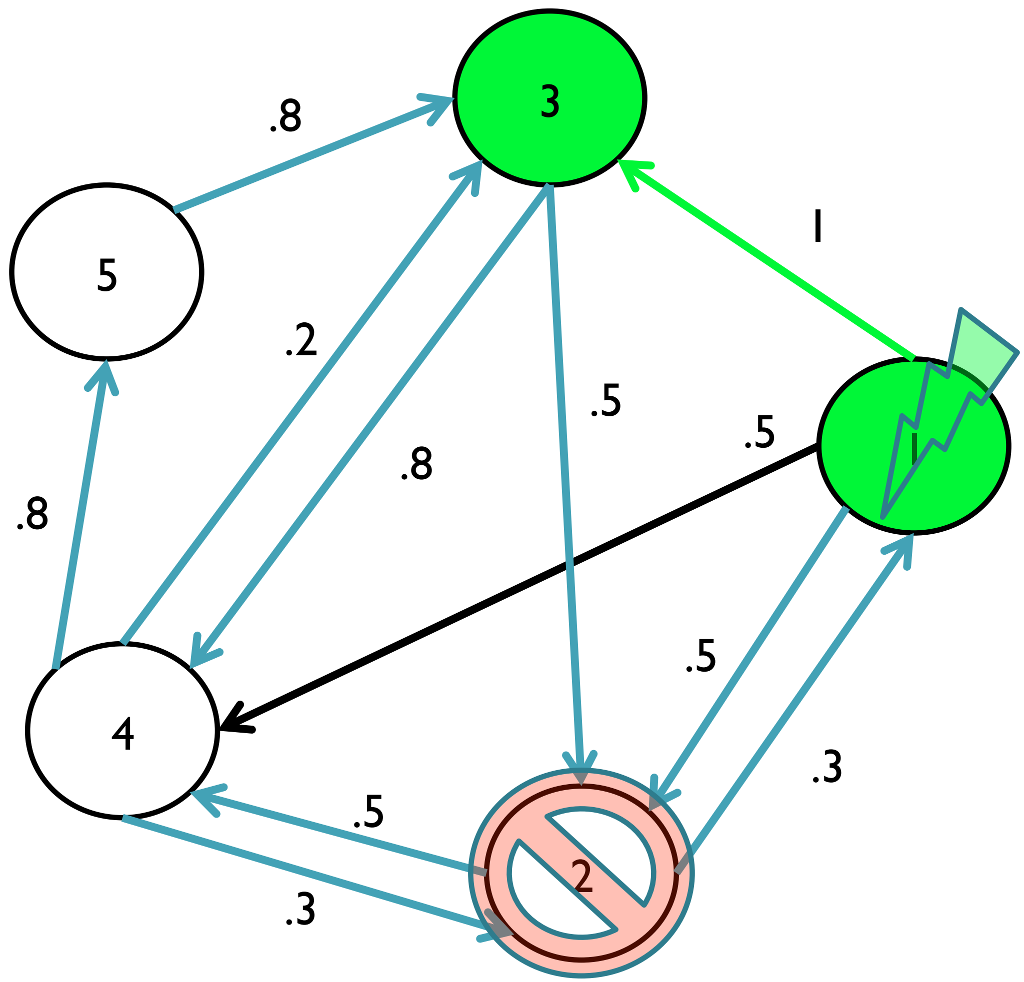
Activations and Suppressions



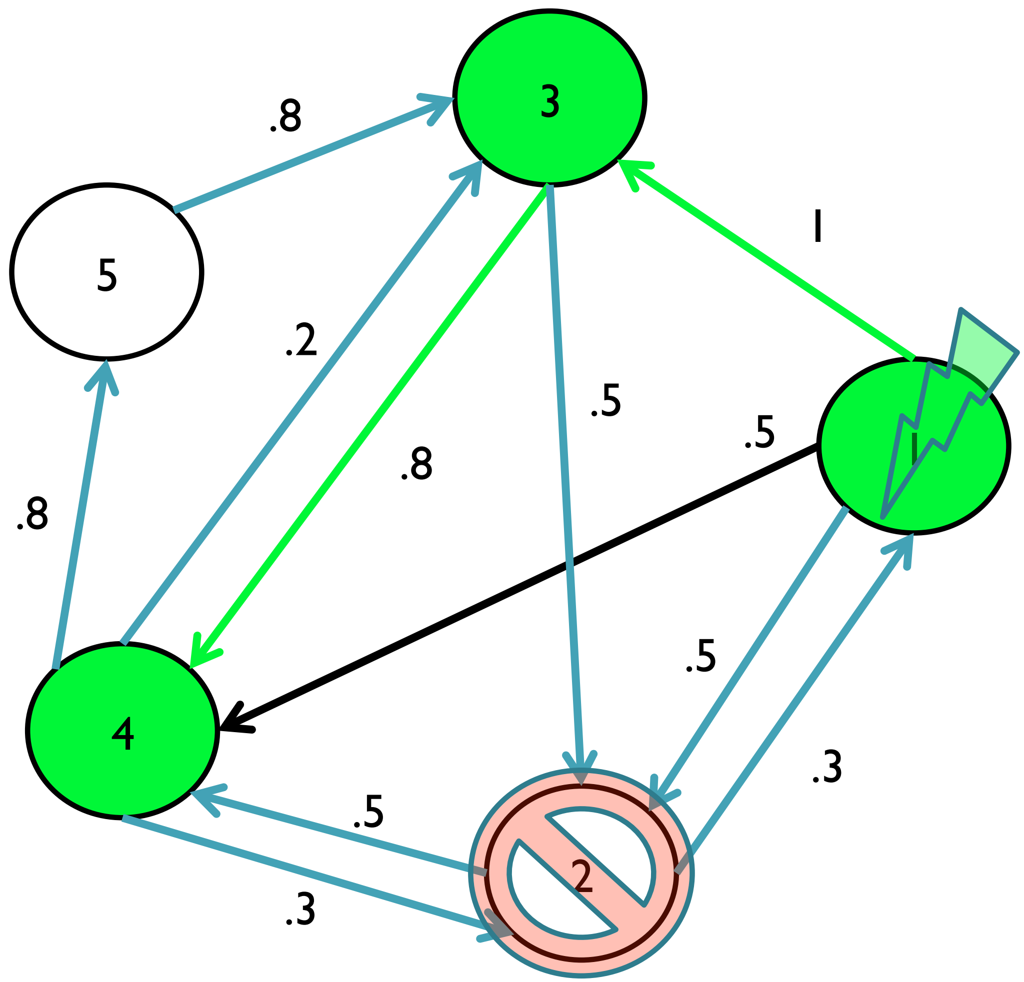
Activations and Suppressions



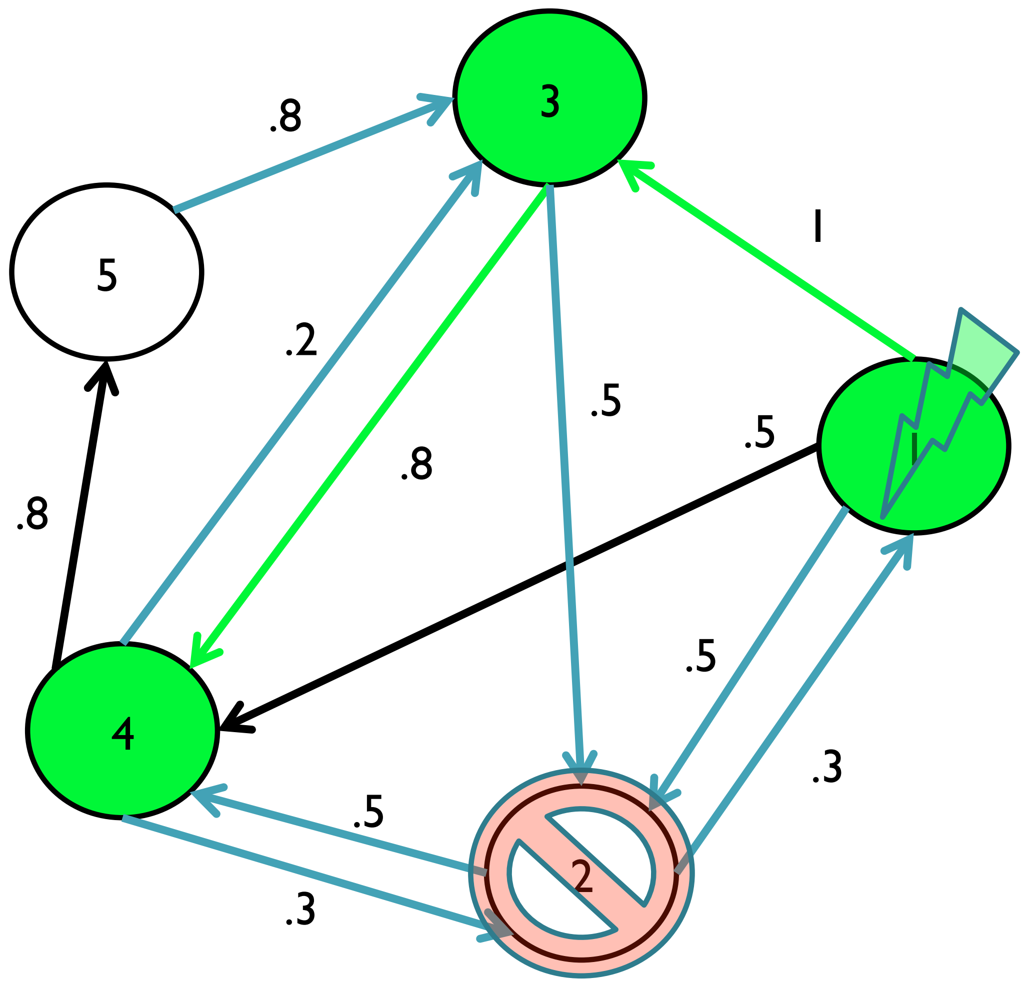
Activations and Suppressions



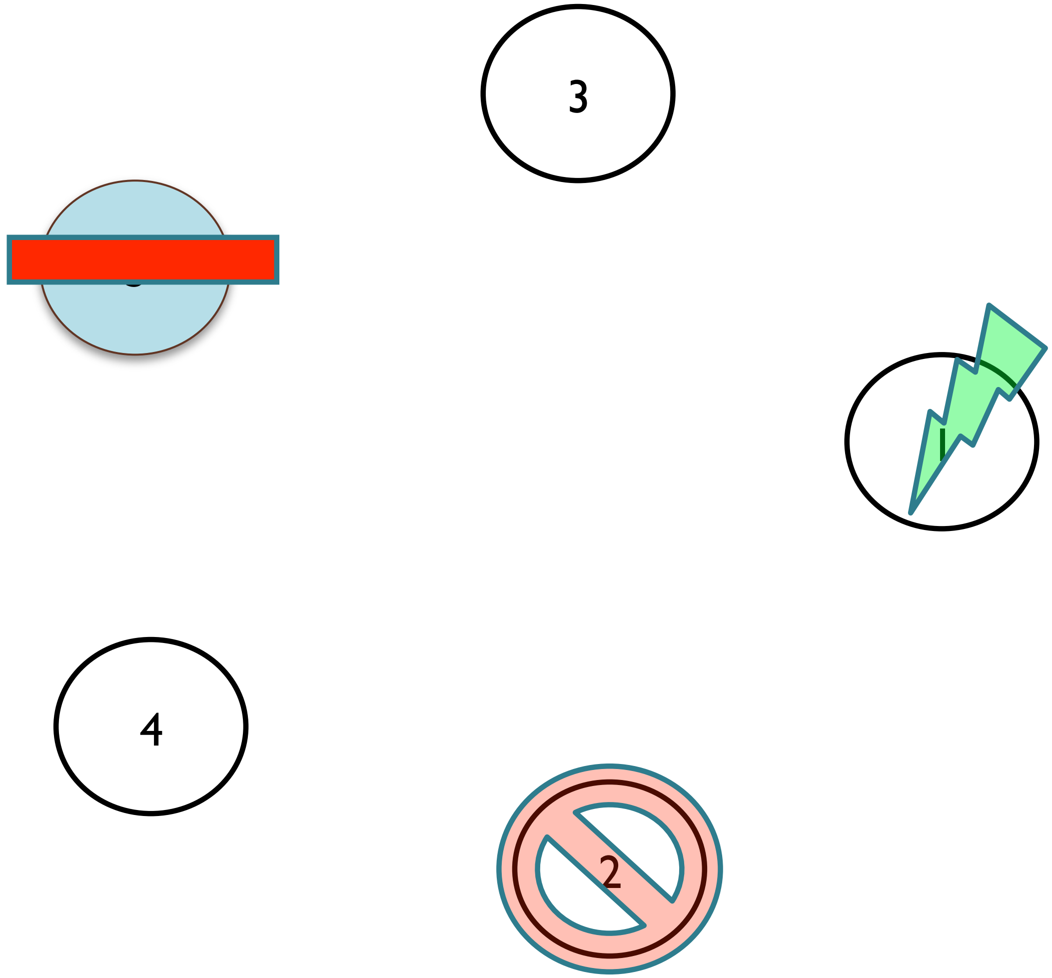
Activations and Suppressions



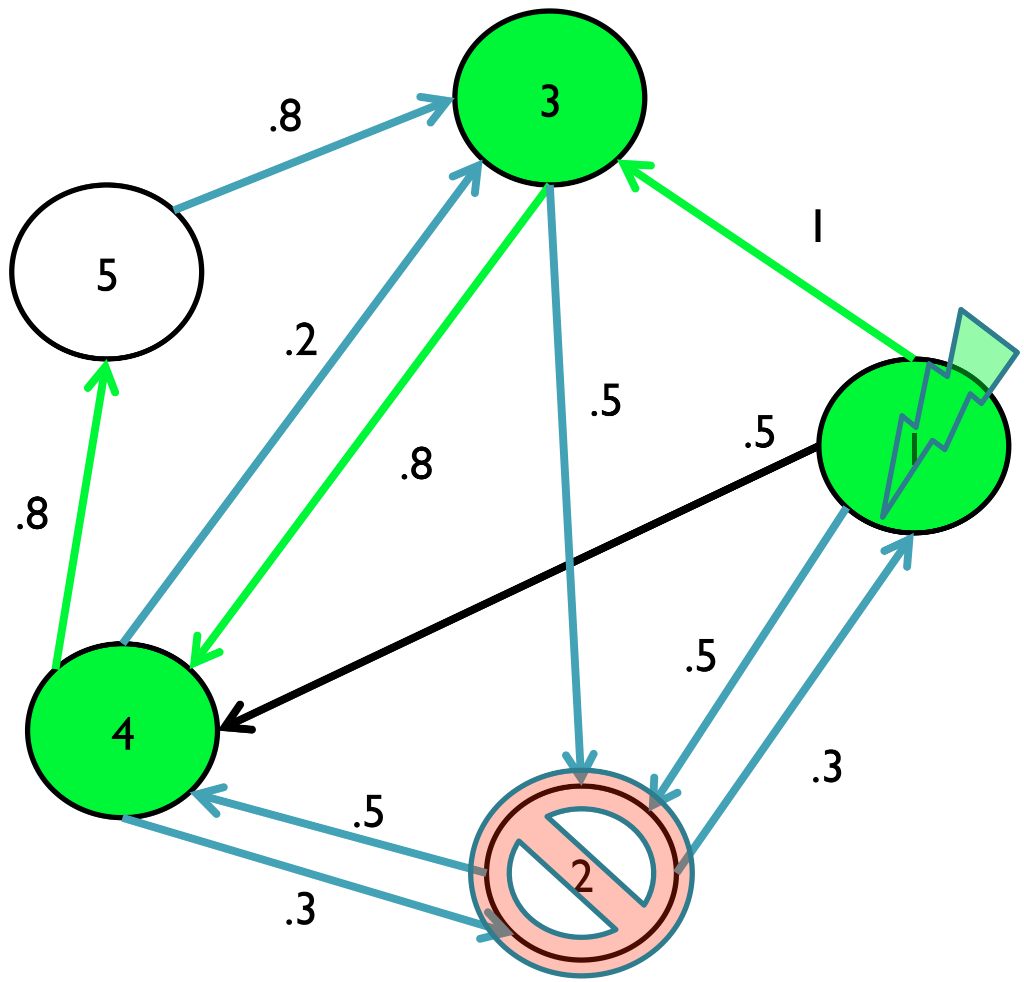
Activations and Suppressions



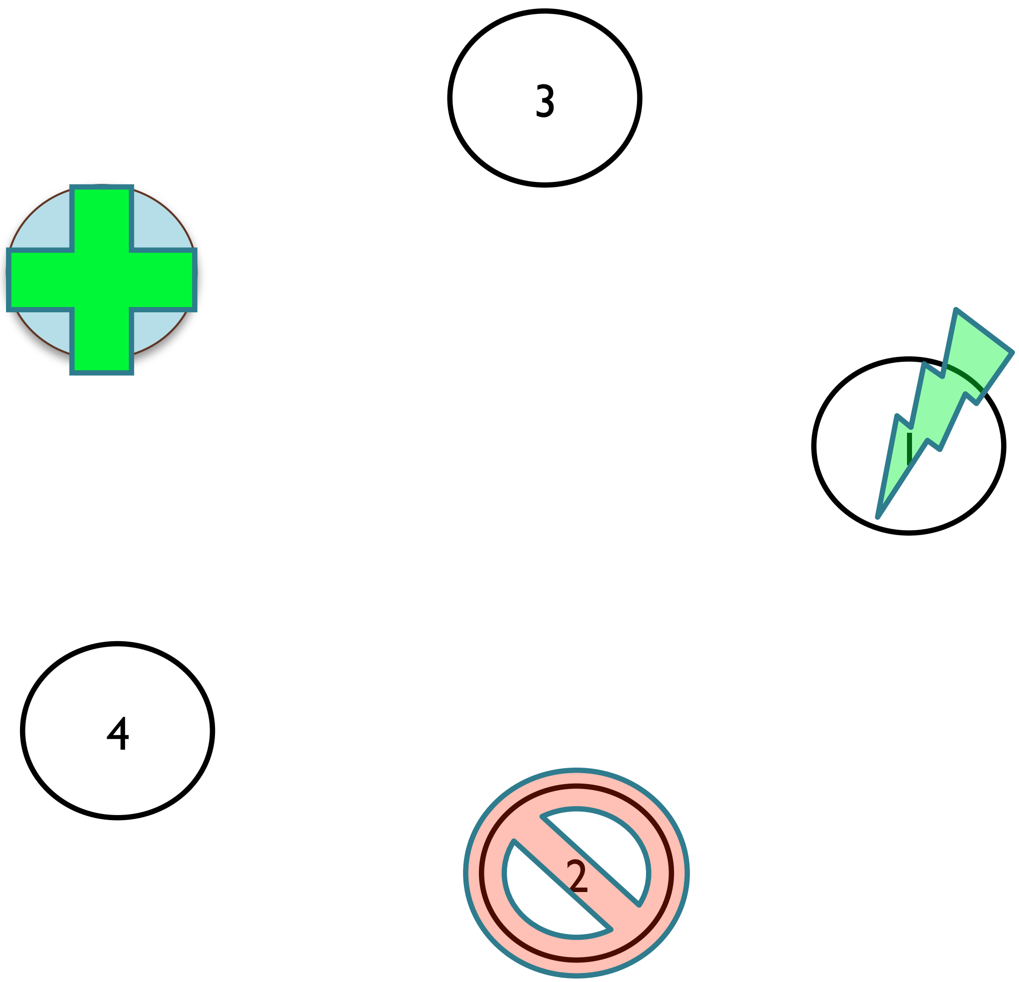
Activations and Suppressions



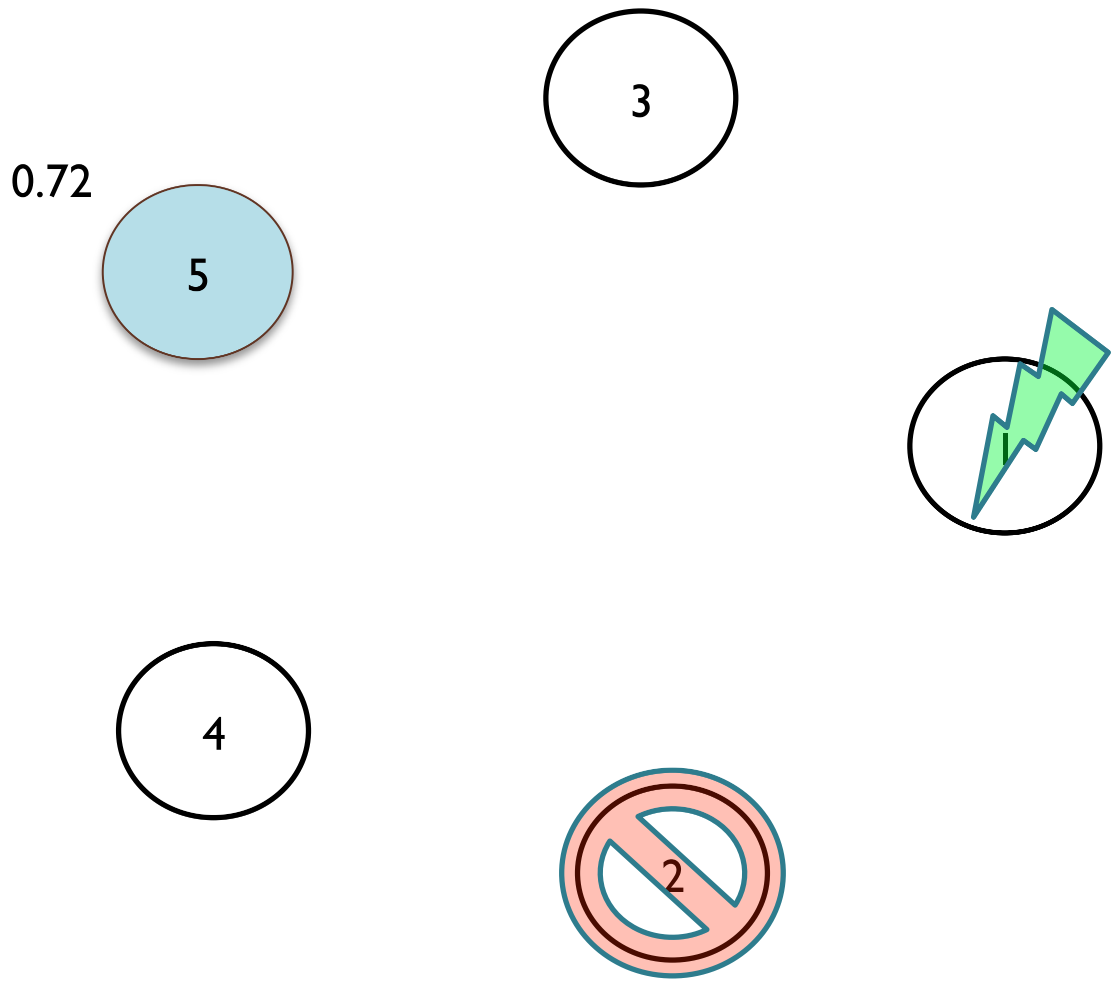
Activations and Suppressions



Activations and Suppressions



Exact Value Injection Queries



The Learning Task

- Two social networks S and S' are behaviorally equivalent if for any experiment e, S(e) = S'(e)

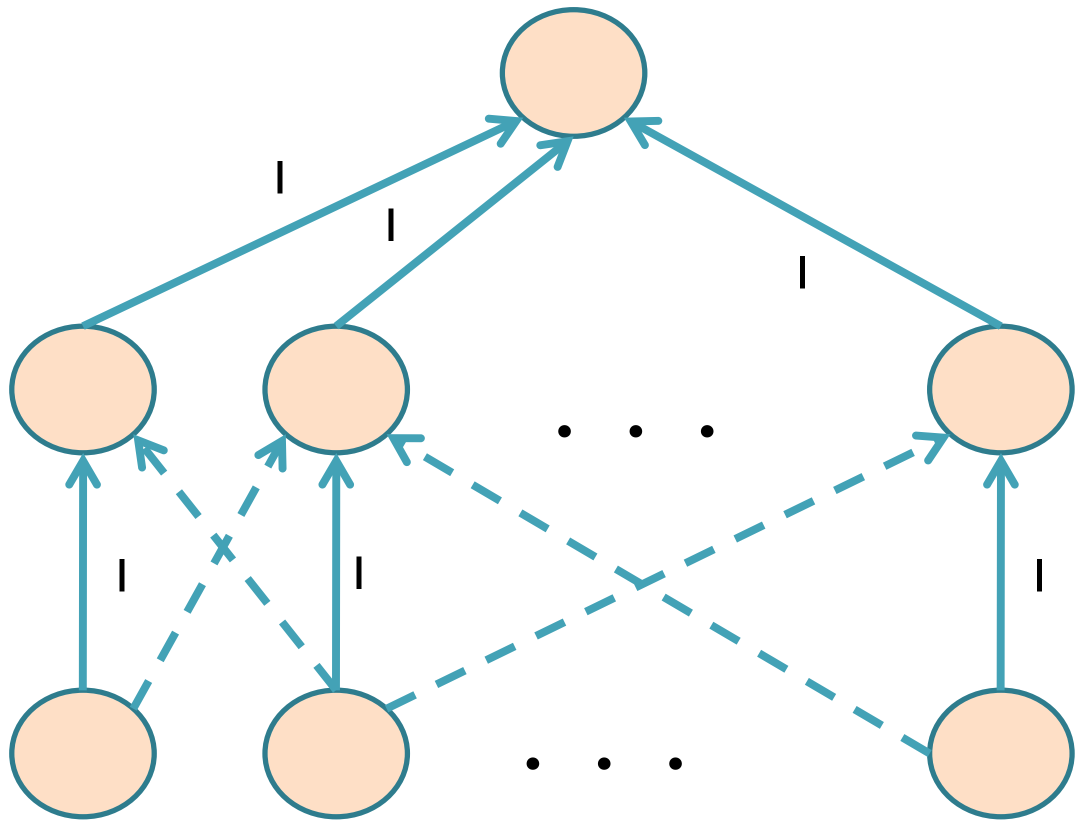
- Given access to a hidden social network S*, the learning problem is to find a social network S behaviorally equivalent to S* using value injection queries.

The Percolation Model

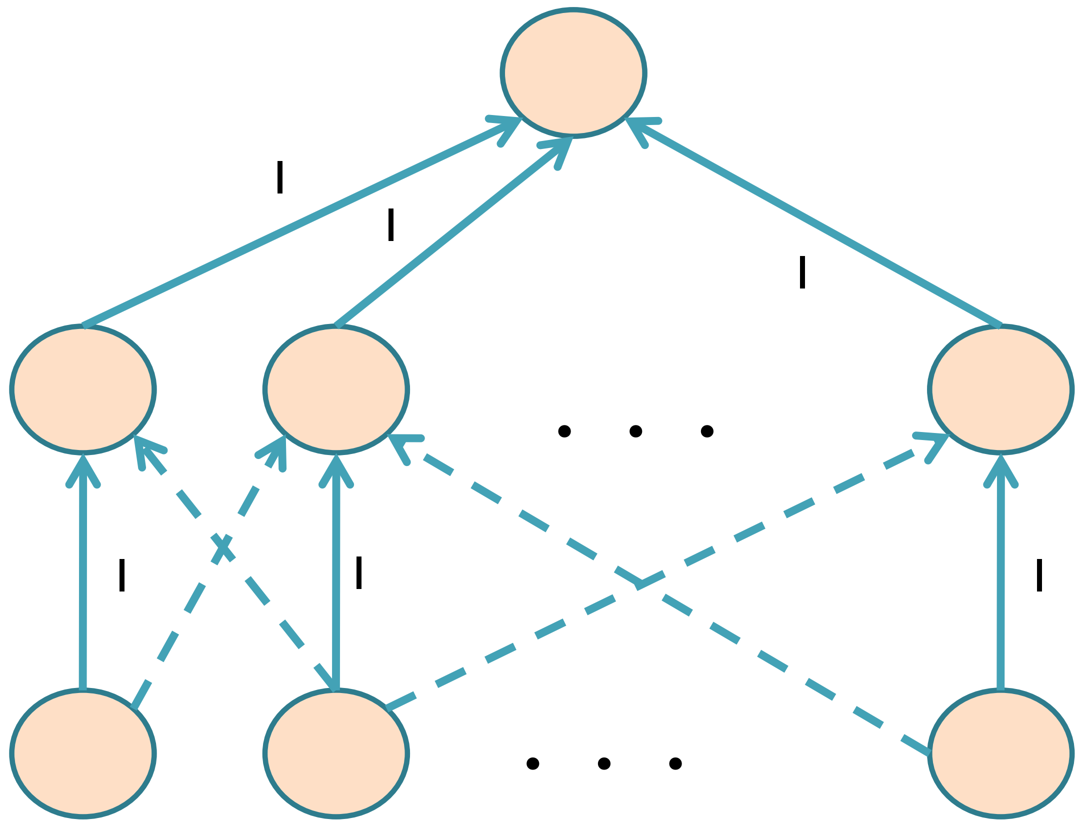
Given a network S and a VIQ

- All edges entering or leaving a suppressed node are automatically "closed."
- Each remaining edge (u,v) is "open" with probability $p_{(u,v)}$ and "closed" with probability $(1 - p_{(u,v)})$
- The result of a VIQ is the probability there is a path from a activated node to the output via open edges in S

A Lower Bound

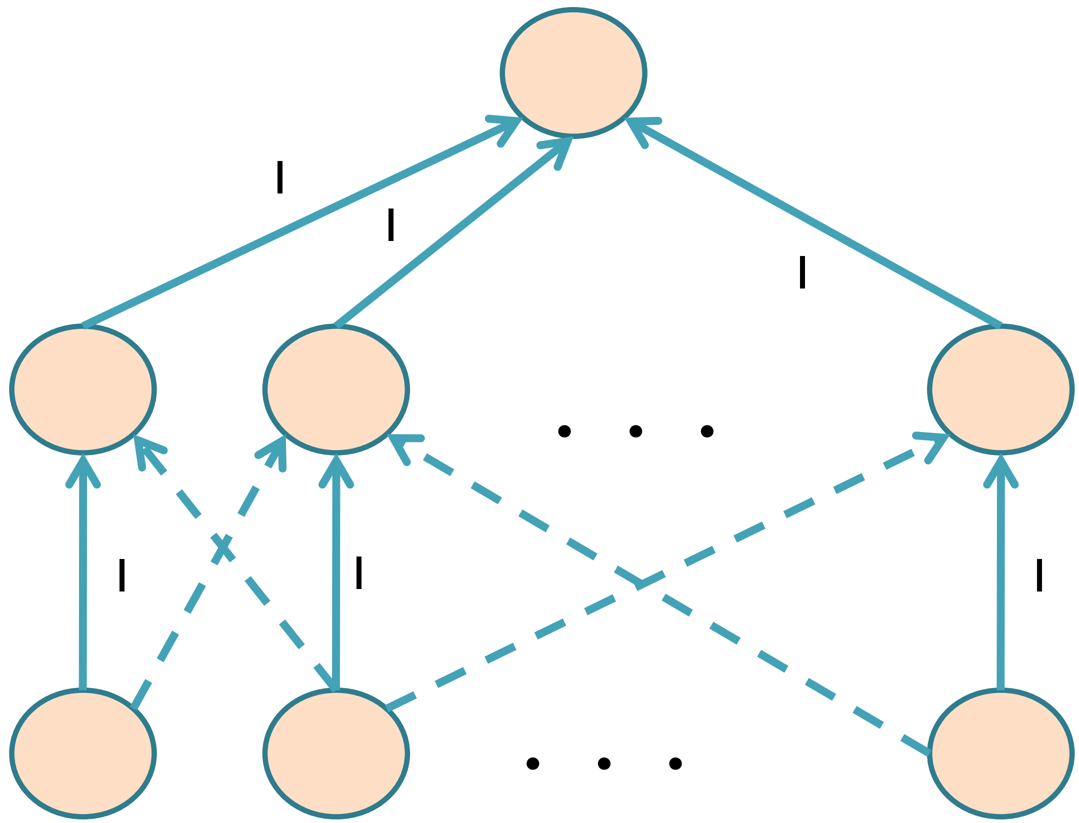


A Lower Bound



All queries give 1-bit answers

A Lower Bound



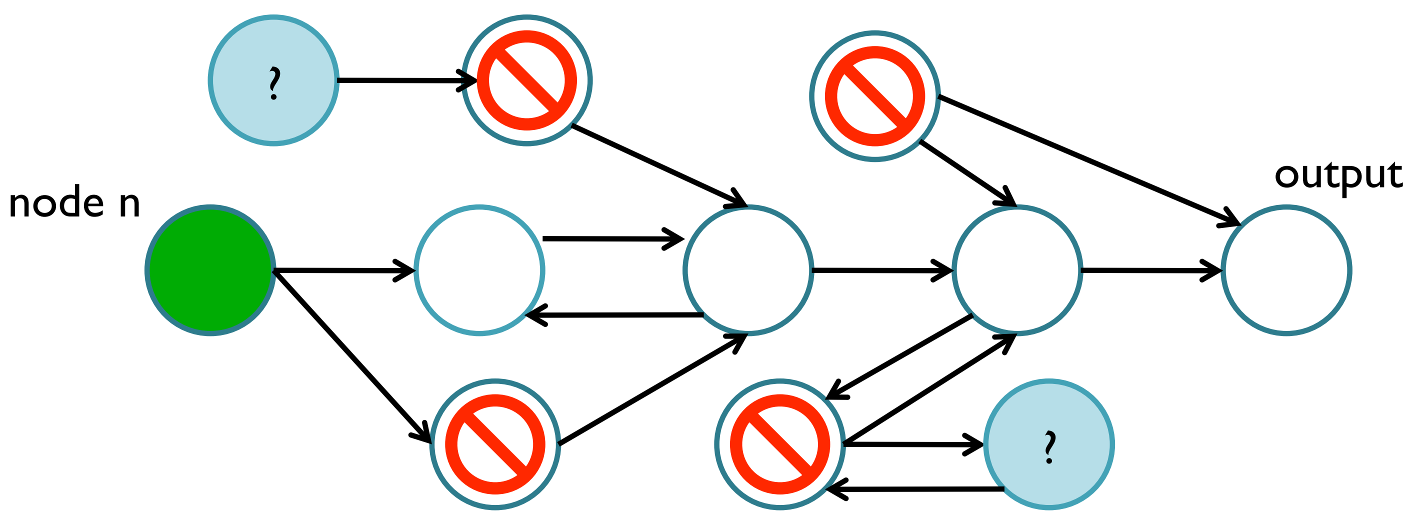
$2^{\Omega(n^2)}$ such graphs, $\Omega(n^2)$ l.b.

An Algorithm: First Some Definitions

- The depth of a node is its distance to the root
- An Up edge is an edge from a node of larger depth to a node of smaller depth
- A Level edge is an edge between two nodes of same depth
- A Down edge is an edge from a node at smaller depth to a node at higher depth
- A leveled graph of a social network is the graph of its Up edges

Excitation Paths

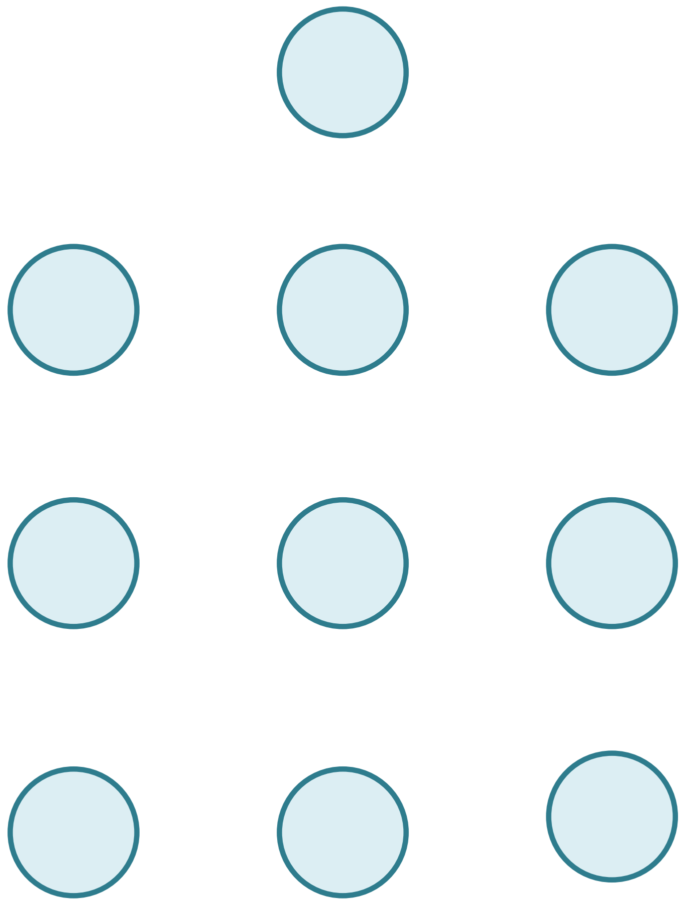
- An excitation path for a node n is a VIQ in which a subset of the free agents form a simple directed path from n to the output. All agents not on the path with inputs into the path are suppressed.
- We also have a shortest excitation path



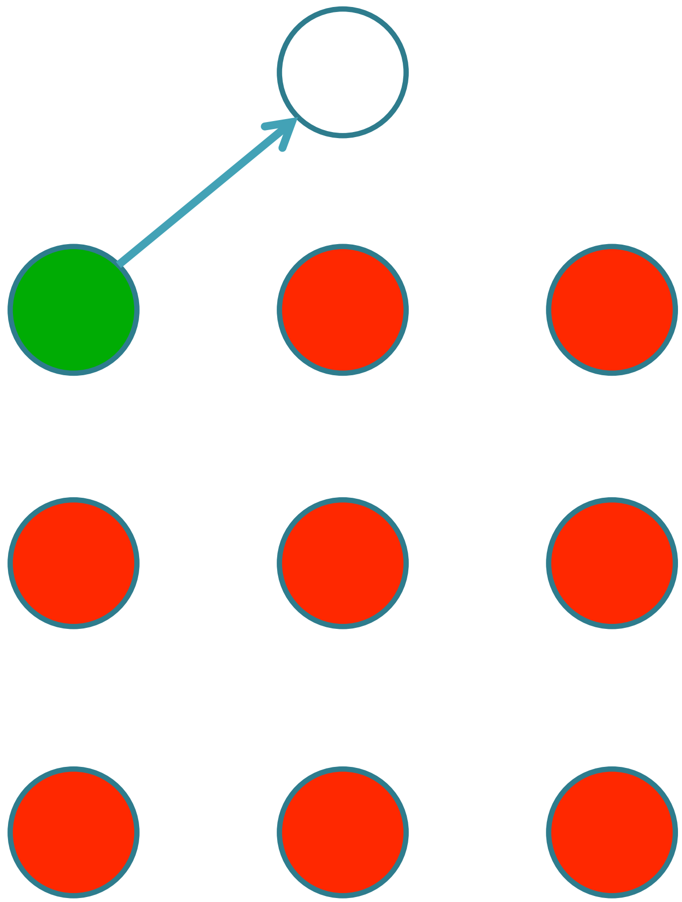
The Learning Algorithm For Networks w/o Probability1 Edges

- First Find-Up-Edges to learn the leveled graph of S
- For each level, Find-Level-Edges
- For each level, starting from the bottom, Find-Down-Edges

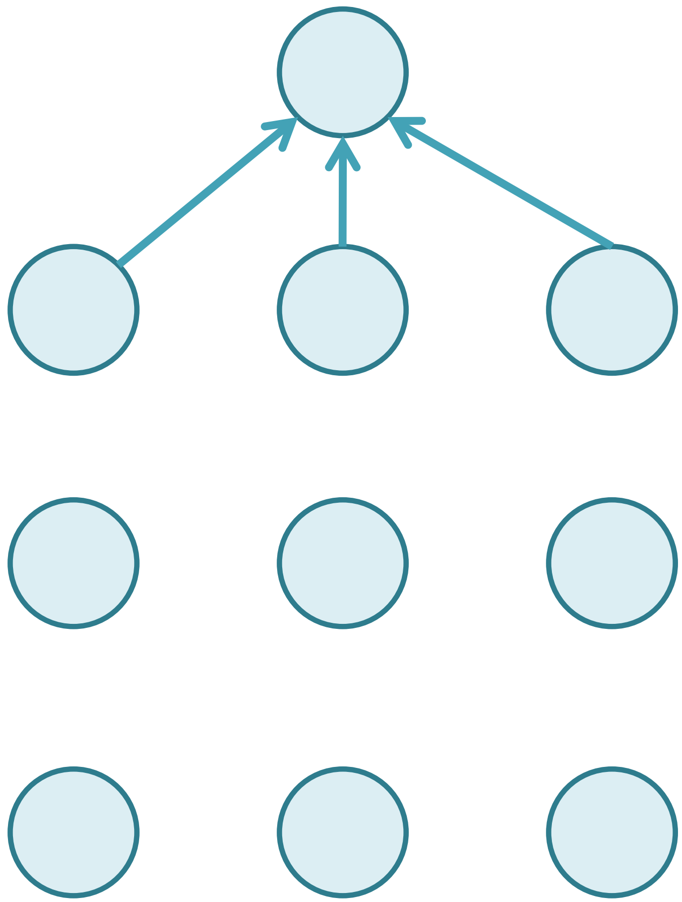
Find-Up-Edges



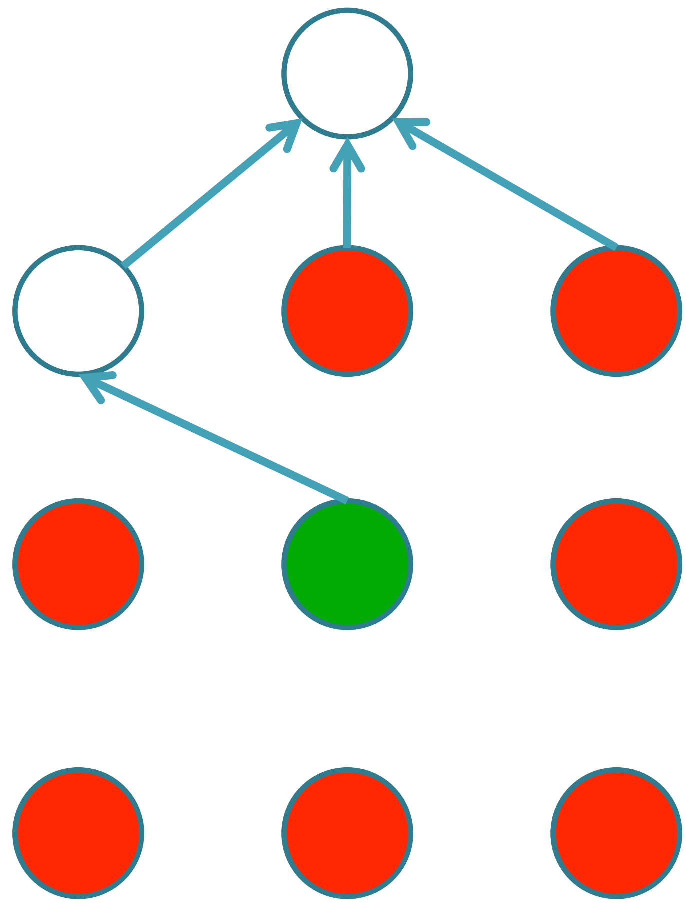
Find-Up-Edges



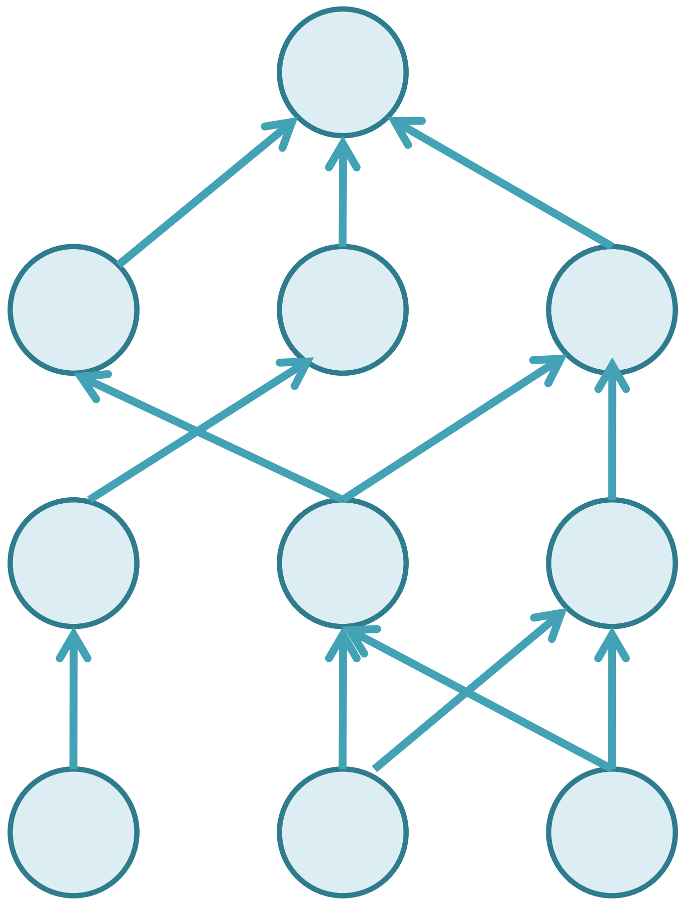
Find-Up-Edges



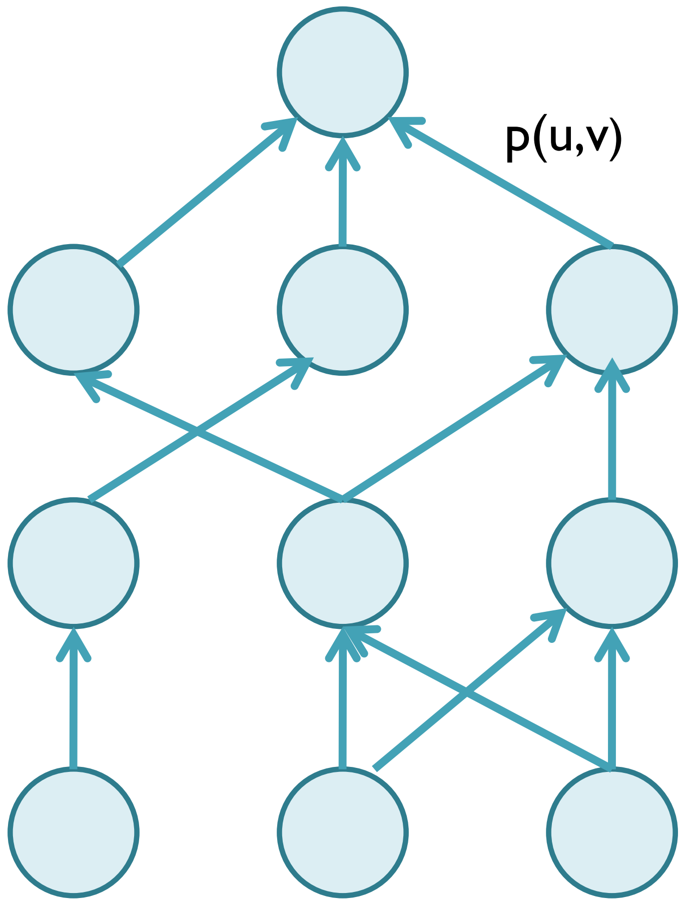
Find-Up-Edges



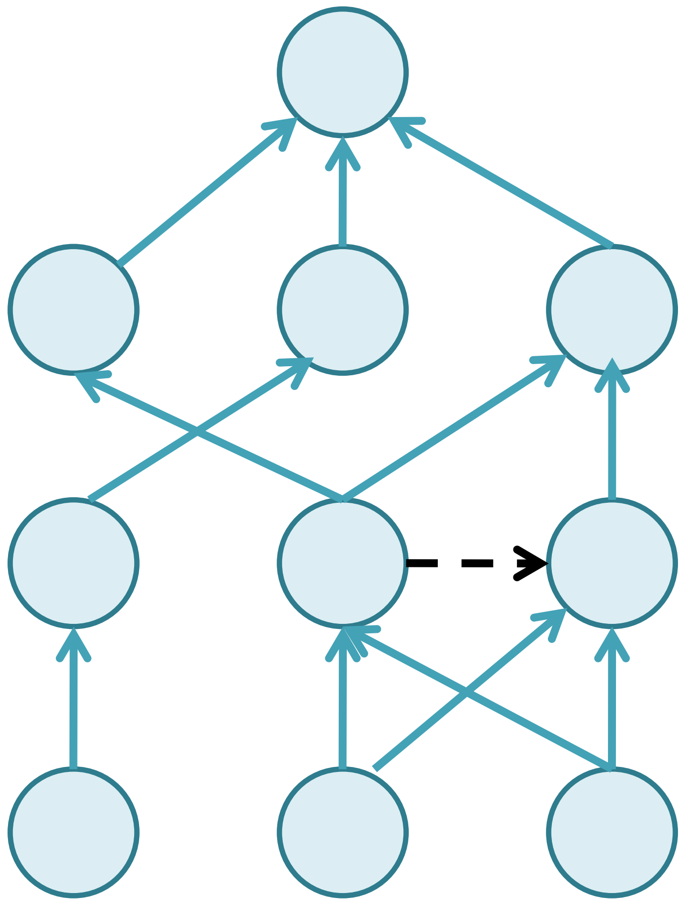
Find-Up-Edges



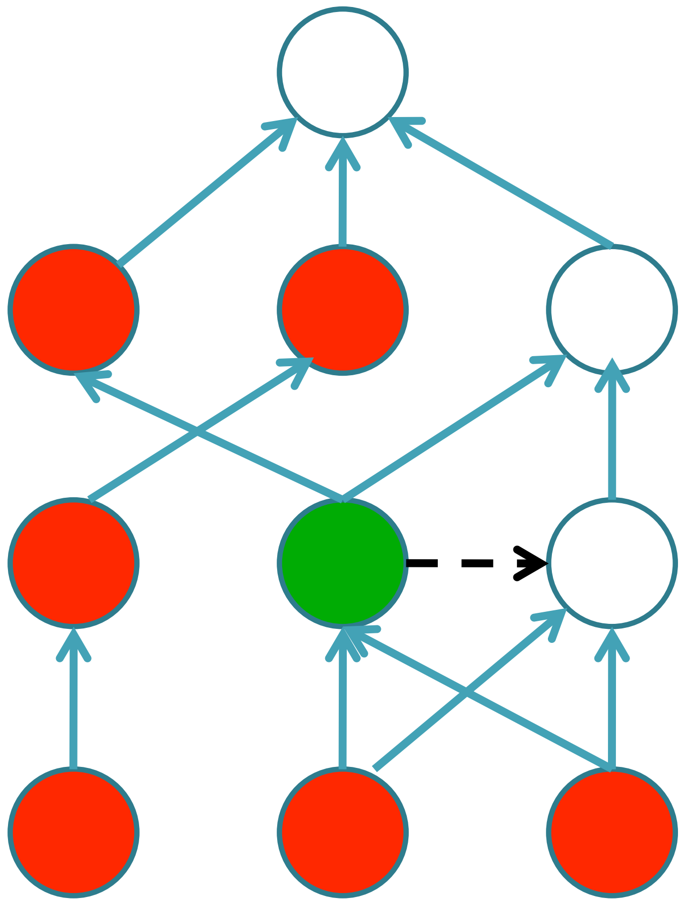
Find-Up-Edges



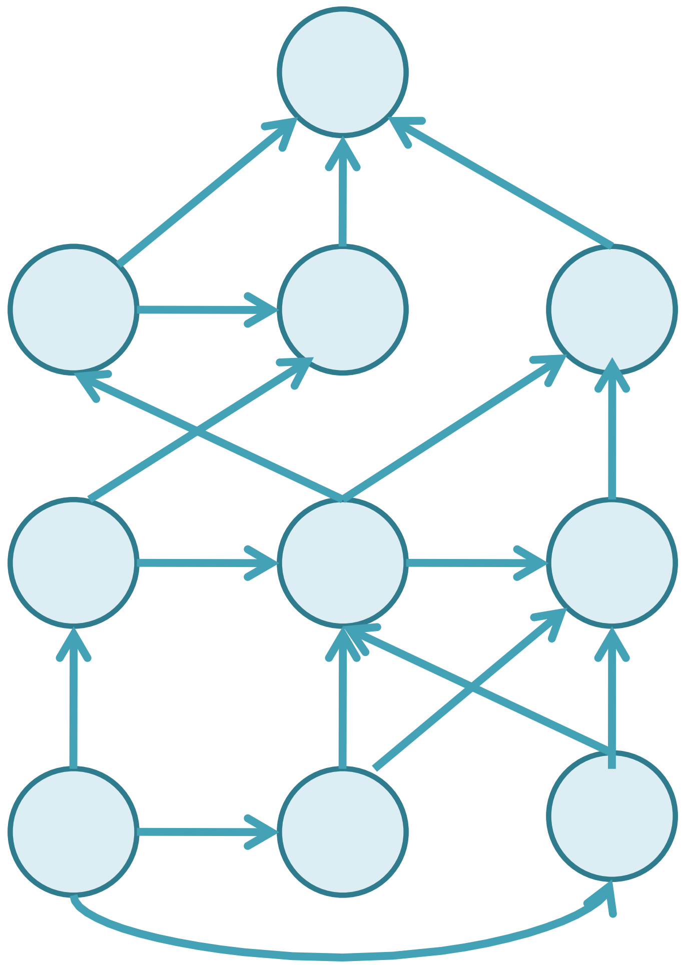
Find-Level-Edges



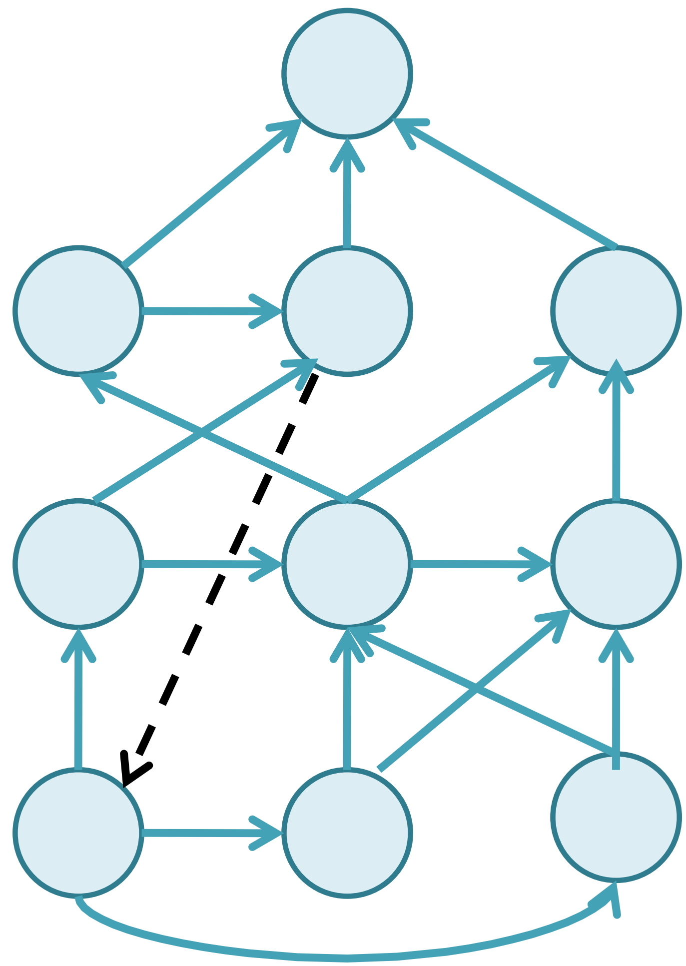
Find-Level-Edges



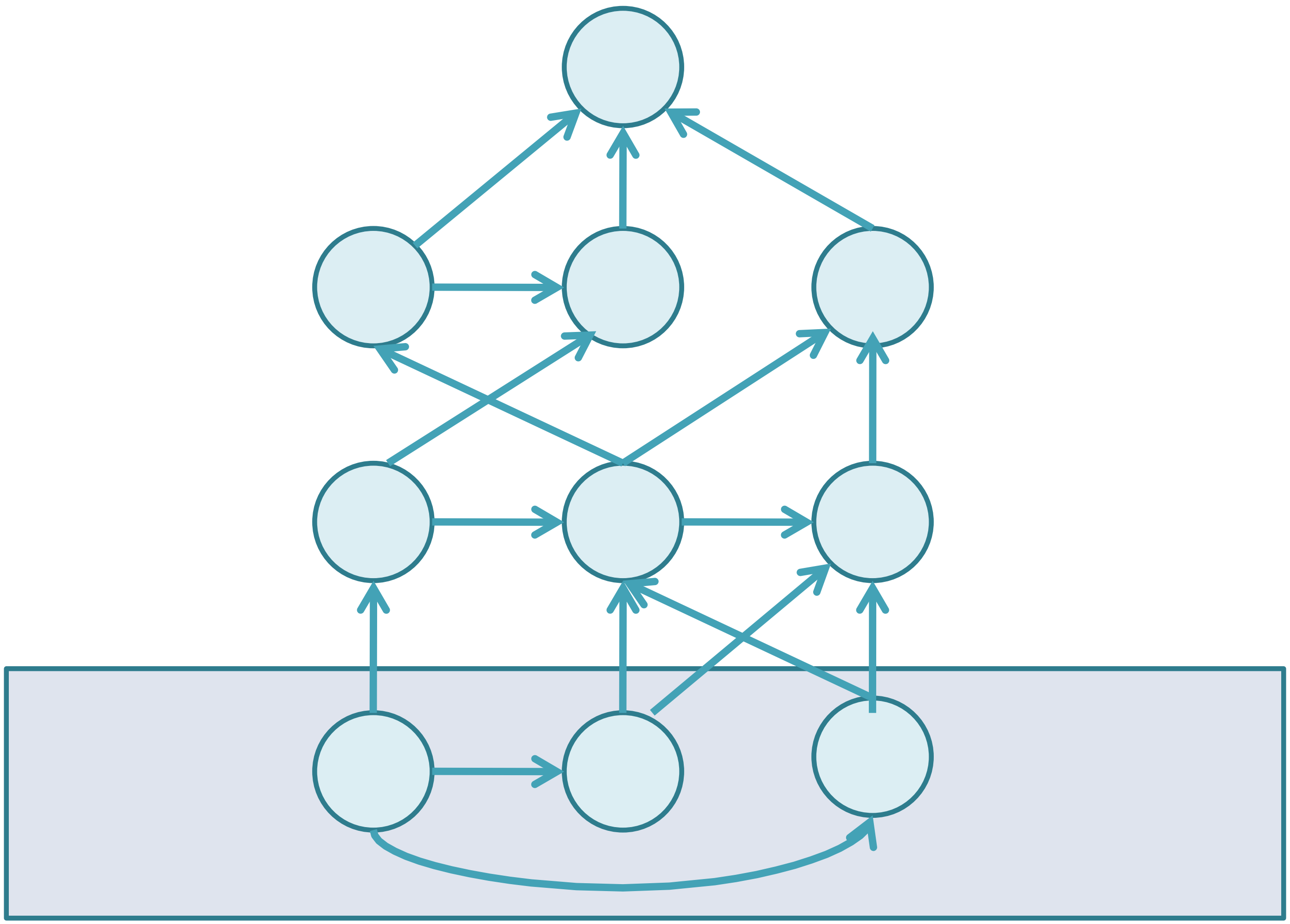
Find-Level-Edges



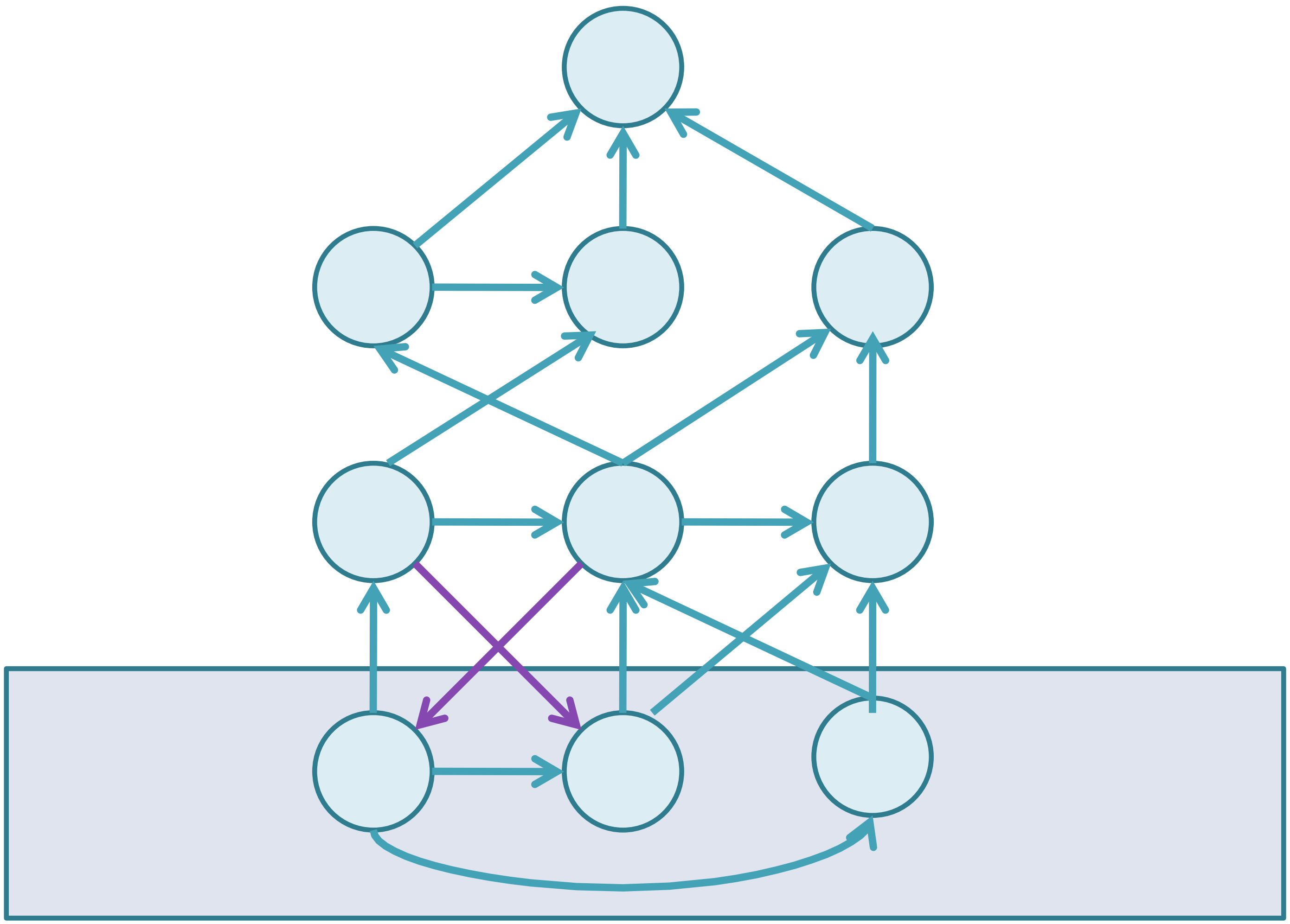
Find-Down-Edges



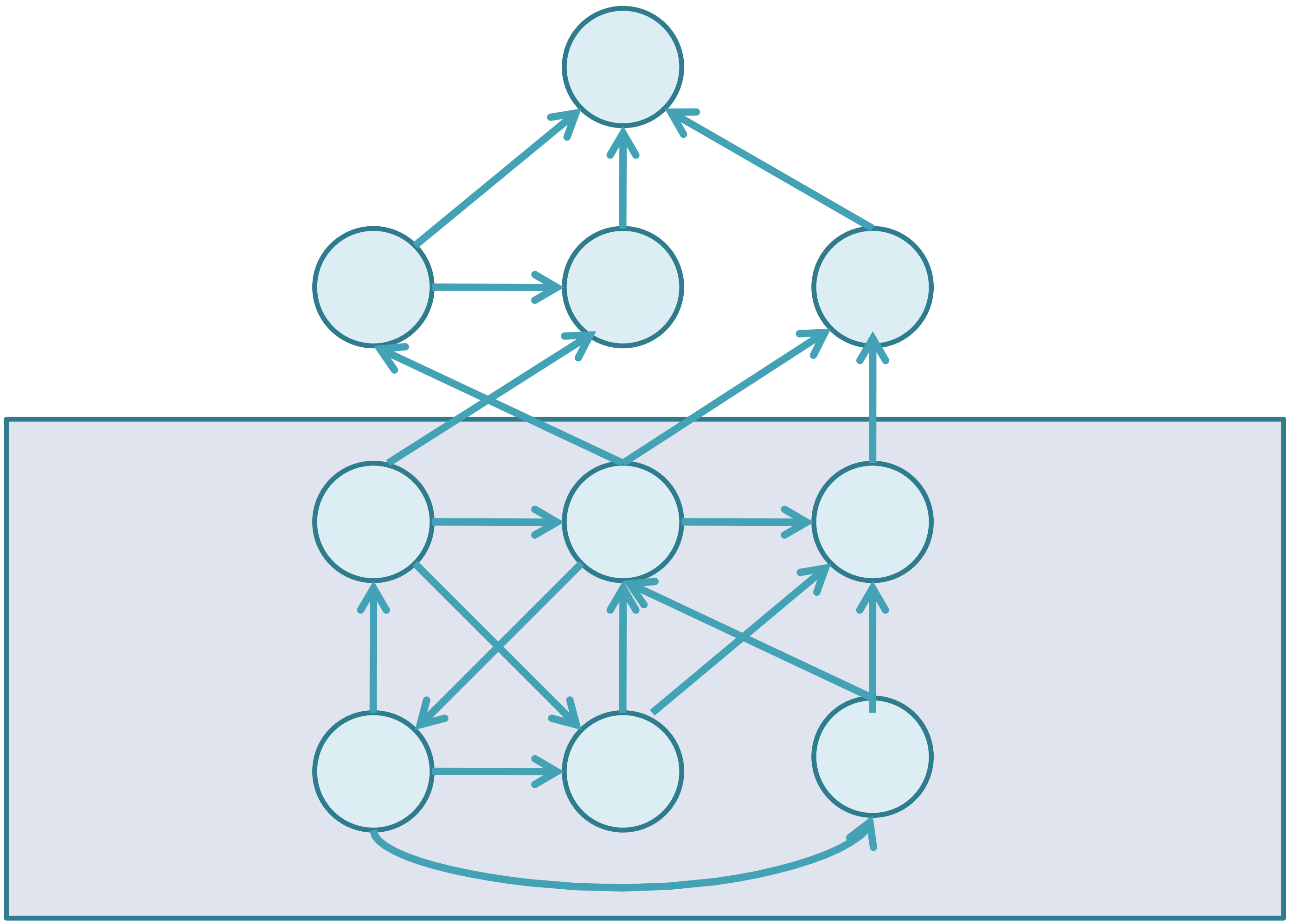
Find-Down-Edges



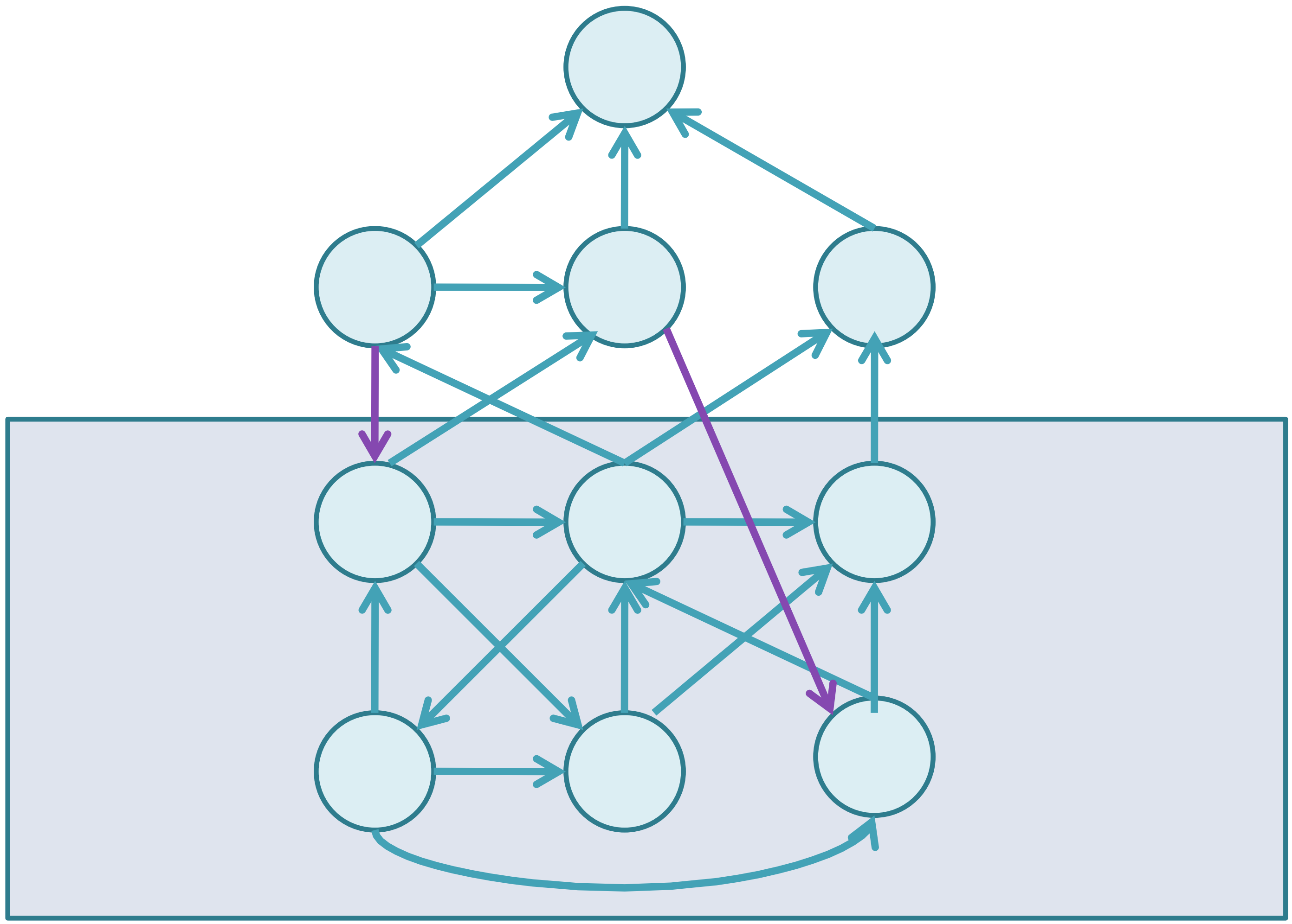
Find-Down-Edges



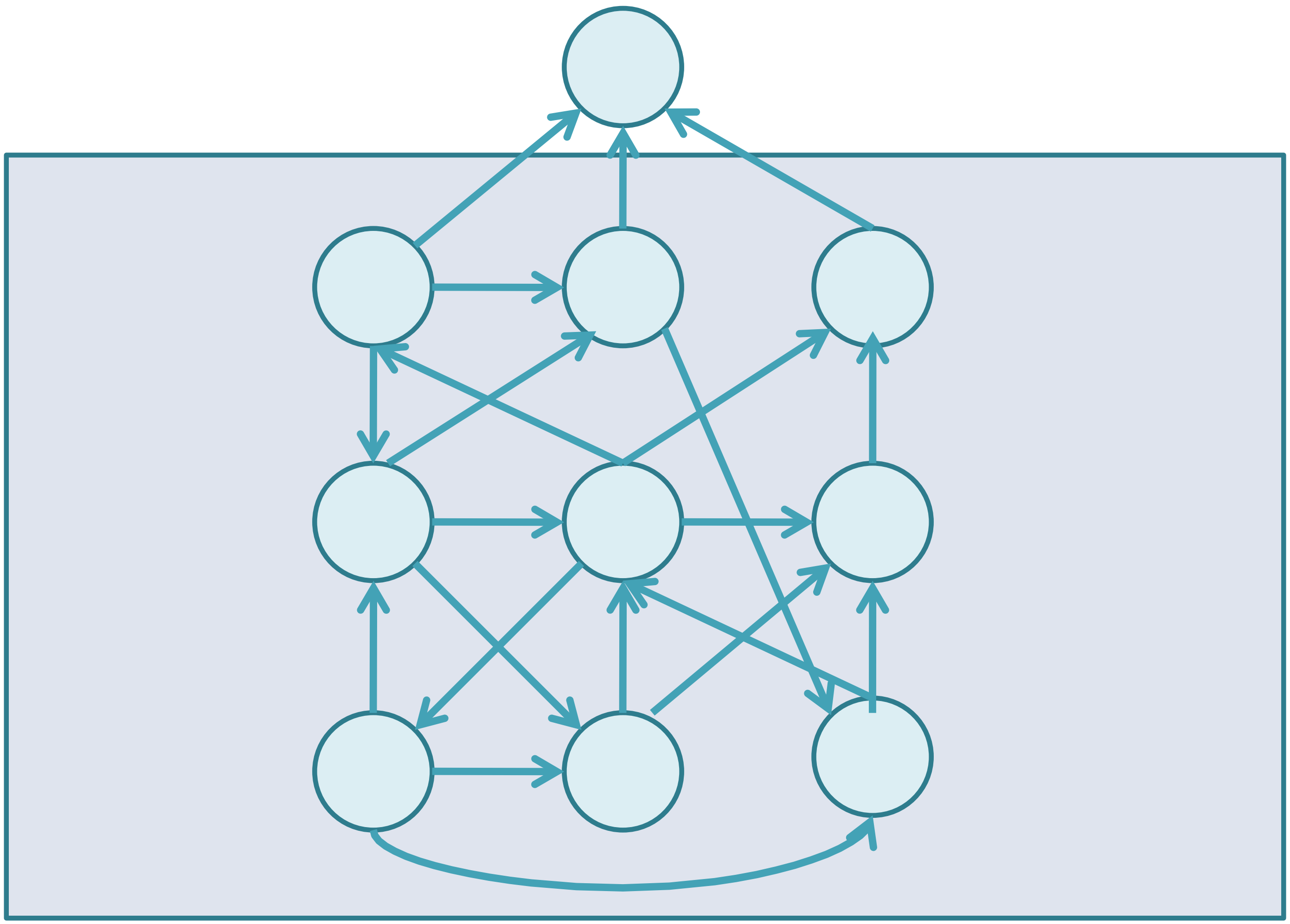
Find-Down-Edges



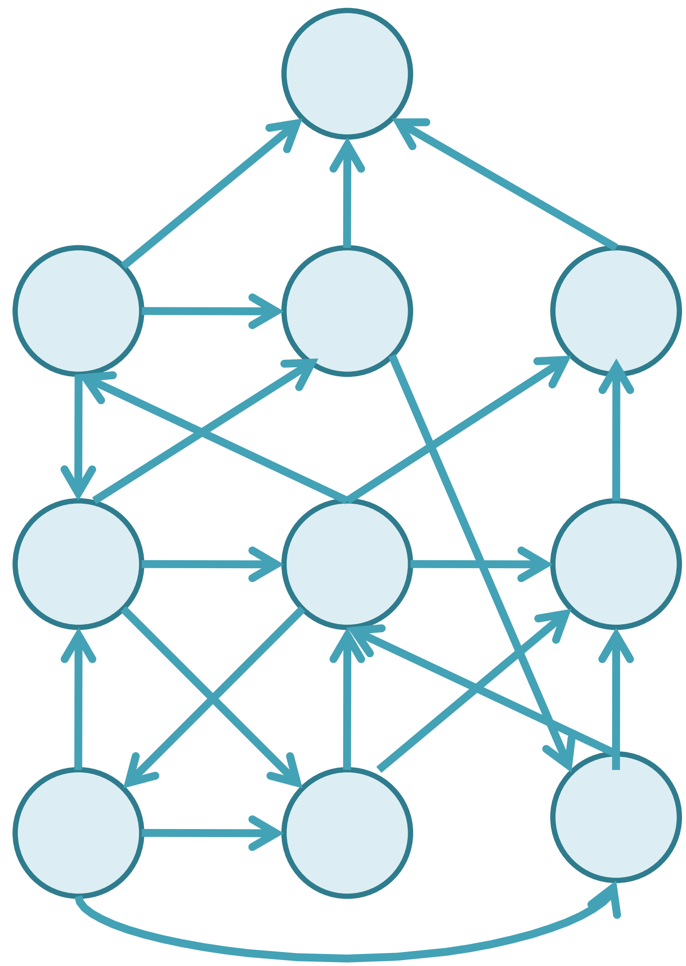
Find-Down-Edges



Find-Down-Edges



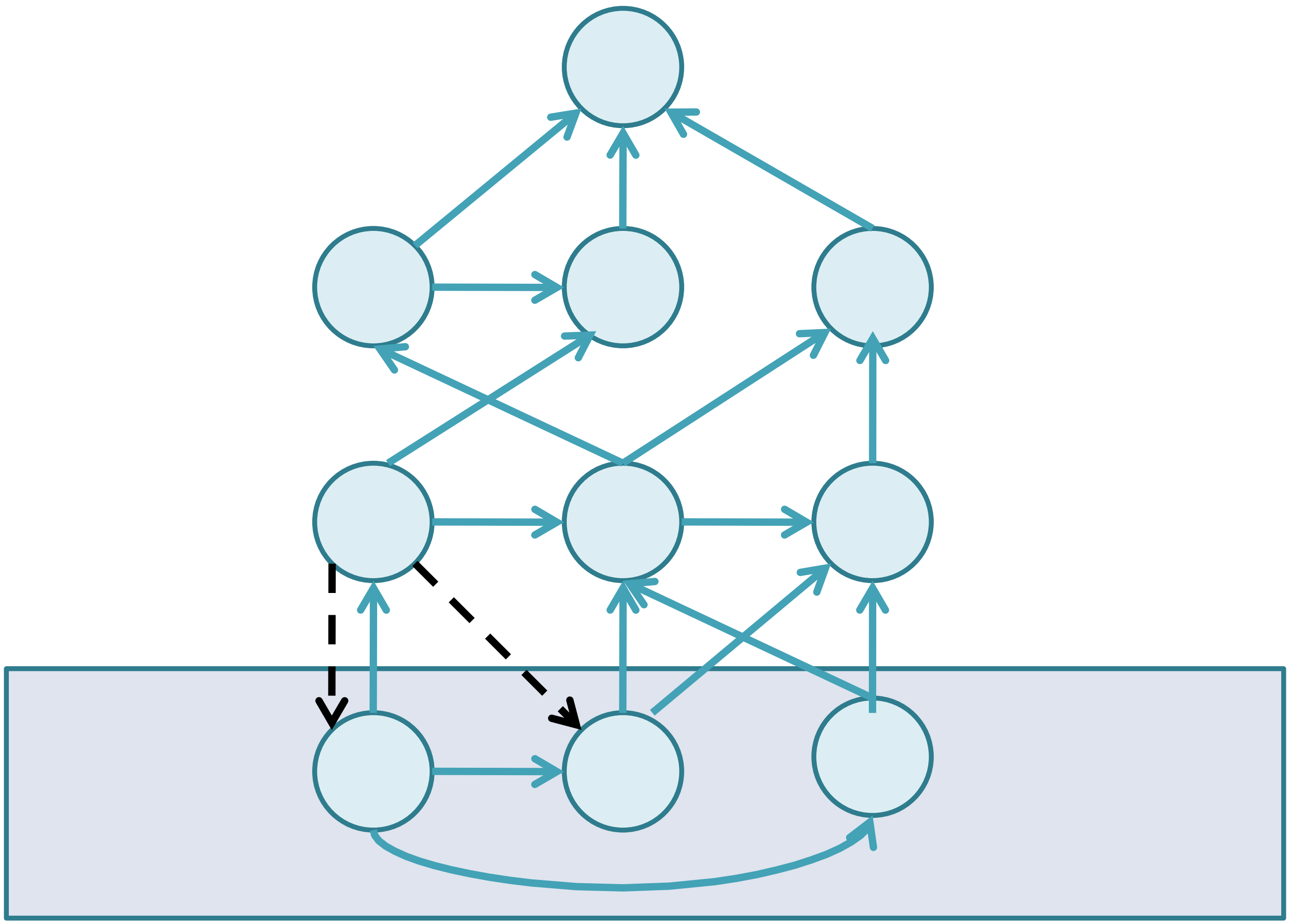
Find-Down-Edges



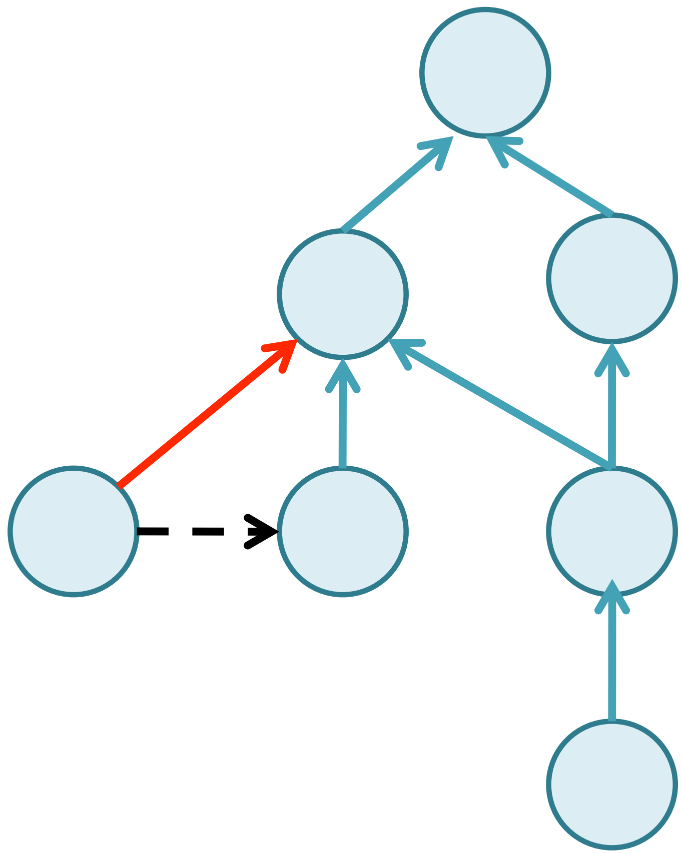
Find-Down-Edges

- For each node u at current level
  - Sort each node v_i in C (complete set) by distance to the root in G – {u}
  - Let v_1 … v_k be the sorted v_is
  - Let pi_1 … pi_k be their corresponding shortest paths to the root in G – {u}
  - For i from 1 to k
    - Do experiment of firing u, leaving pi_i free, and suppressing the rest of the nodes.

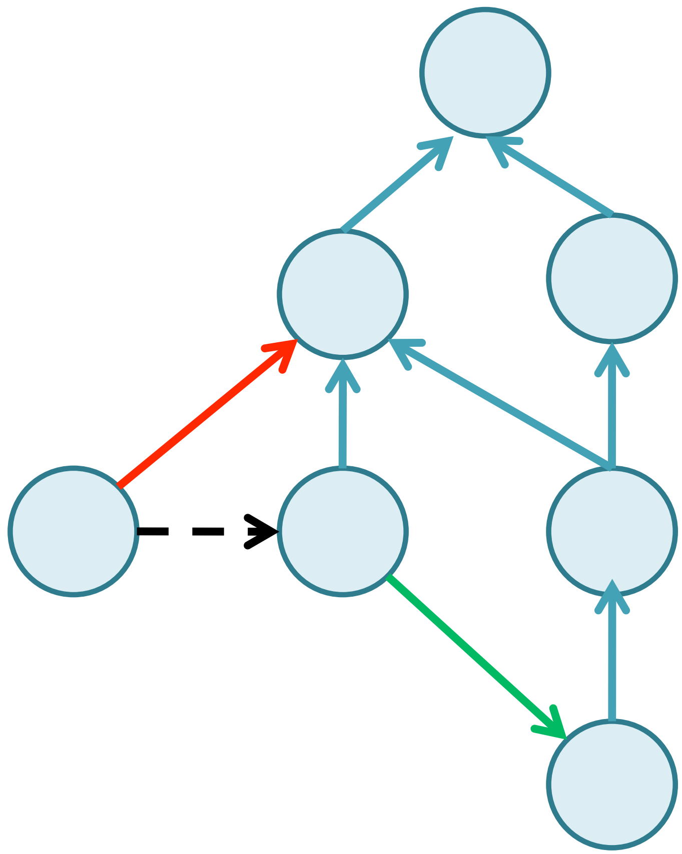
For Example



With Ones – a Problem



With Ones – a Problem



With Ones

- Algorithm gets more complicated
- Level edges and down edges are found in one subroutine
- In looking for down edges from u, need to avoid not just u, but also all nodes reachable from u by 1 edges

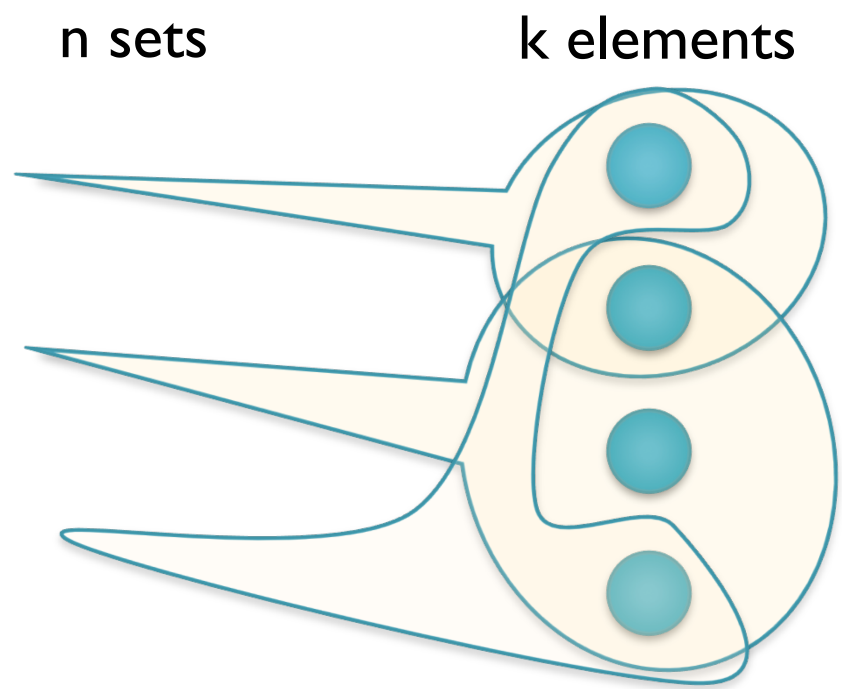
In the End

- We do 1 query per each possible edge, giving an $O(n^2)$ algorithm
- Matches the $\Omega(n^2)$ lower bound

Finding Influential Nodes

- Suppose instead of learning the social network, we wanted to find an influential set of nodes quickly.
- A set of nodes is influential if, when activated, activates the output with probability at least p
- NP Hard to Approximate to log n, even if we know the structure of the network

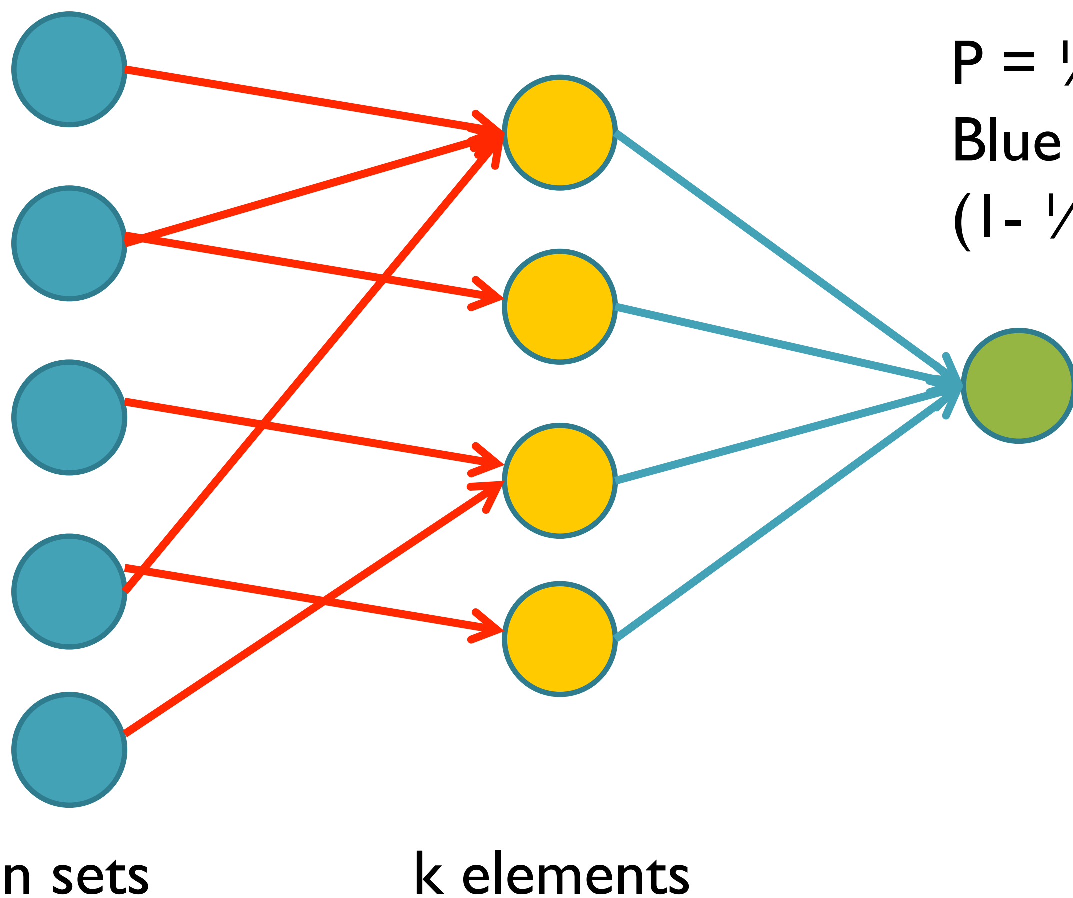
Set Cover



Can assume n = poly(k)

NP Hard to Approximate to log n

Reduction from Set Cover



P = ½
Blue edges have weight
(1- ½ ^(1/k))

An Approximation Algorithm

- Say the optimal solution has m nodes
- Suppose we wanted to fire the output with probability $(p - \varepsilon)$
- Let I be the set of chosen influential nodes.
- Observation: at any point in the algorithm, greedily adding one more node w to I makes

$$S(e_{I \cup \{w\}}) \geq S(e_I) + \frac{p - S(e_I)}{m}$$

Analyzing Greedy

- Using a greedy algorithm, we let k be the number of rounds the algorithm is run

For

$$p\left(1-\frac{1}{m}\right)^k < \epsilon$$

it suffices that

$$e^{-\frac{k}{m}} < \frac{\epsilon}{p}$$

or

$$k > m\log\left(\frac{p}{\epsilon}\right).$$

Summary of the Active Case

- Applies known model to new domain.
- Matching worst-case upper and lower bounds for learning social networks.
- But queries too expensive in most applications…
- Lots of open problems!

What if We Cannot Manipulate the Network?

2009 Cases of Swine Flu

The Constraints

- The social network is an unknown graph, where nodes are agents.
- Let $p_{(u,v)}$ be the a priori probability of an edge between nodes u and v.
- Each observed outbreak induces (or exposes) a constraint.
  - Namely the graph is connected on the induced subset.

Finding the Cheapest Network

- If the prior distribution is independent (and probabilities are small), the maximum likelihood social network maximizes

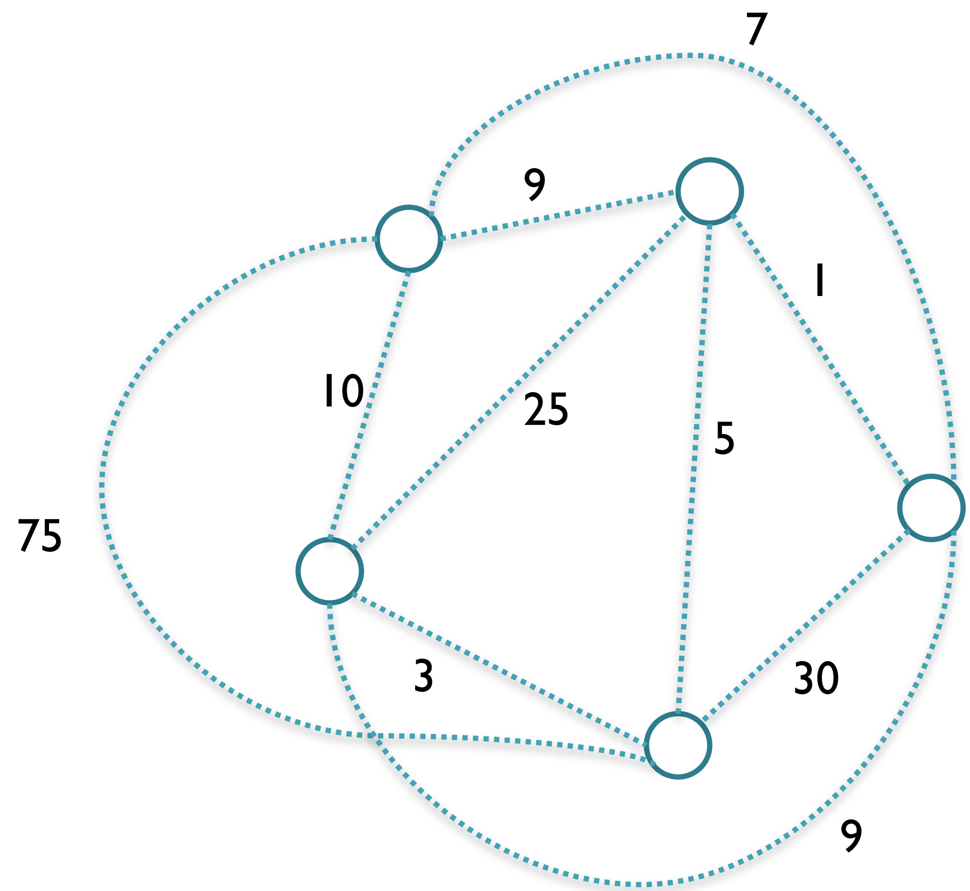
$$\prod_{u,v \in V} p_{(u,v)}$$

- This is equivalent to minimizing the sum of the log-likelihood costs

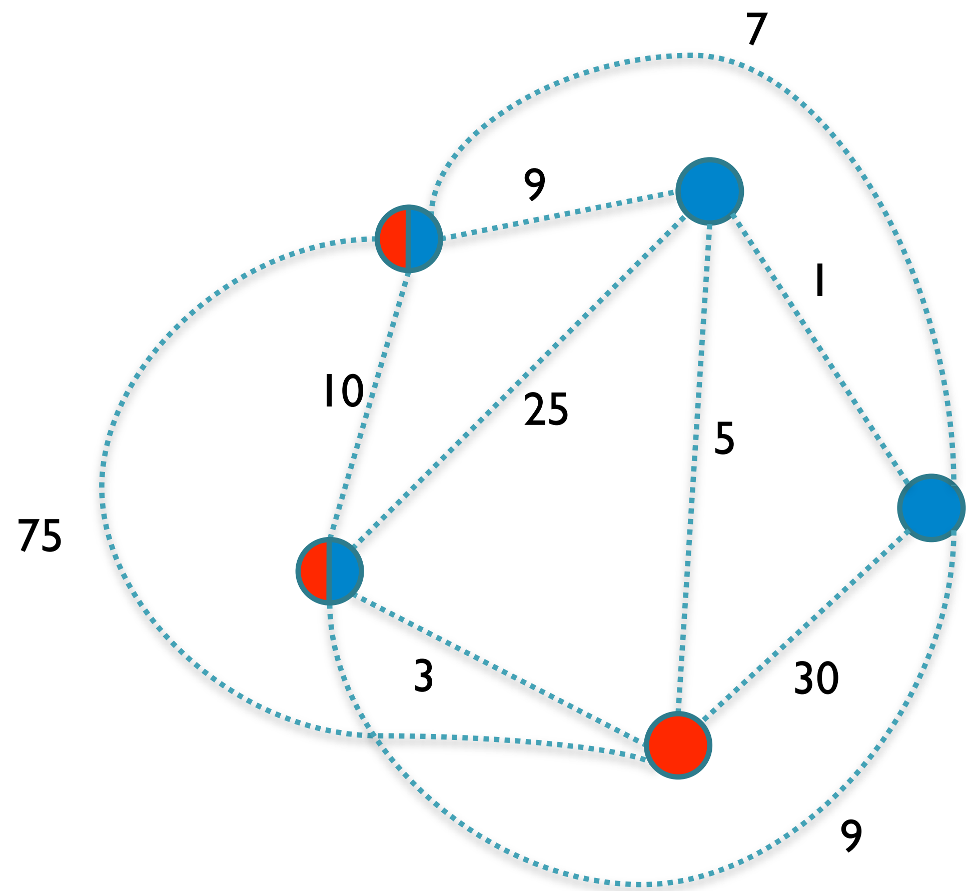
$$\sum_{v,u \in V} - \log(p_{(u,v)})$$

while satisfying the constraints

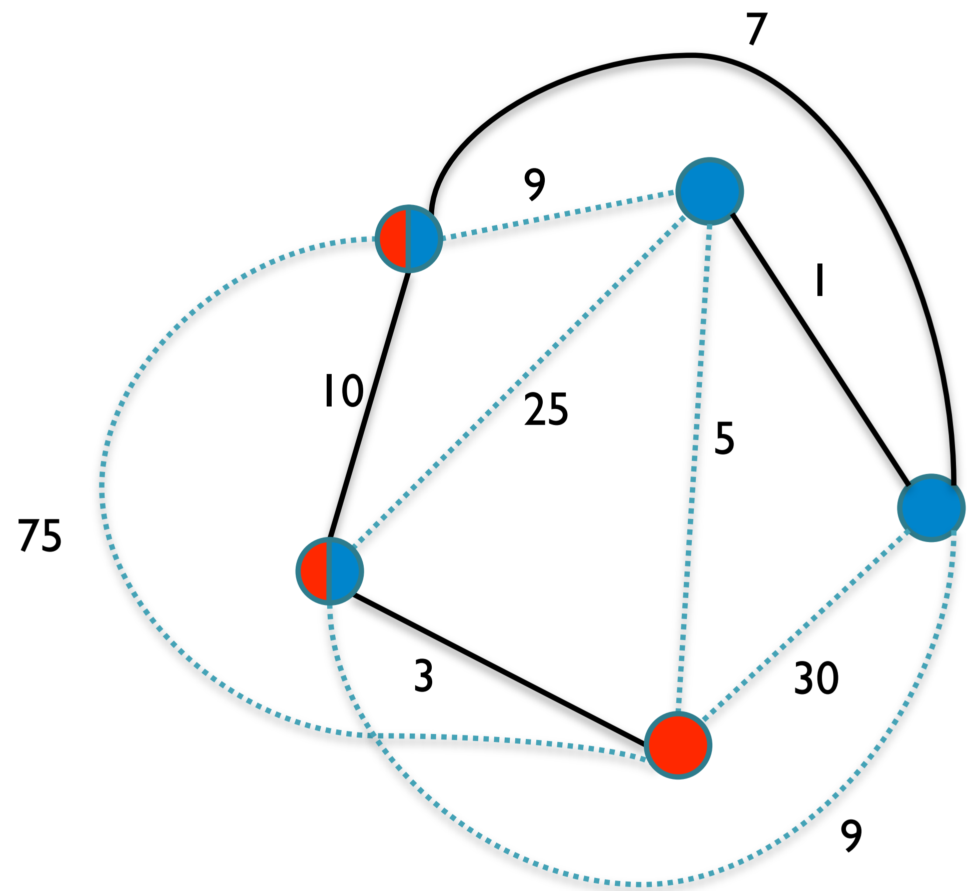
Finding the Cheapest Network Consistent with the Constraints



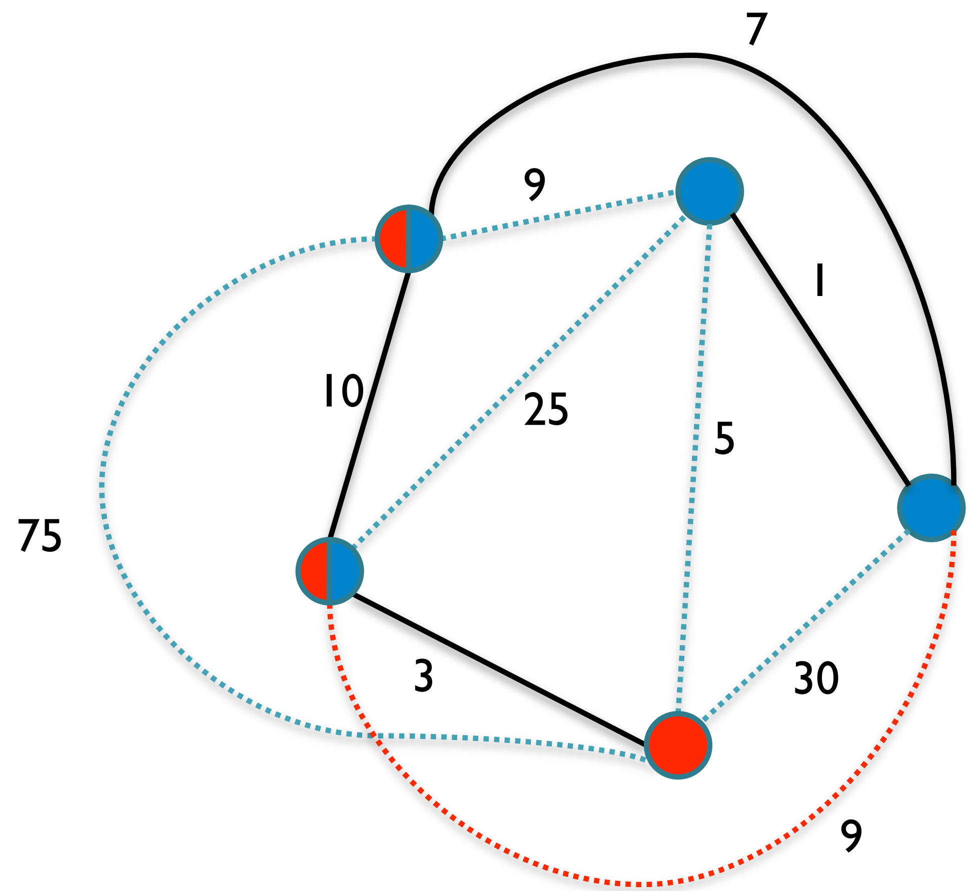
Finding the Cheapest Network Consistent with the Constraints



Finding the Cheapest Network Consistent with the Constraints



Finding the Cheapest Network Consistent with the Constraints



The Network Inference Problem

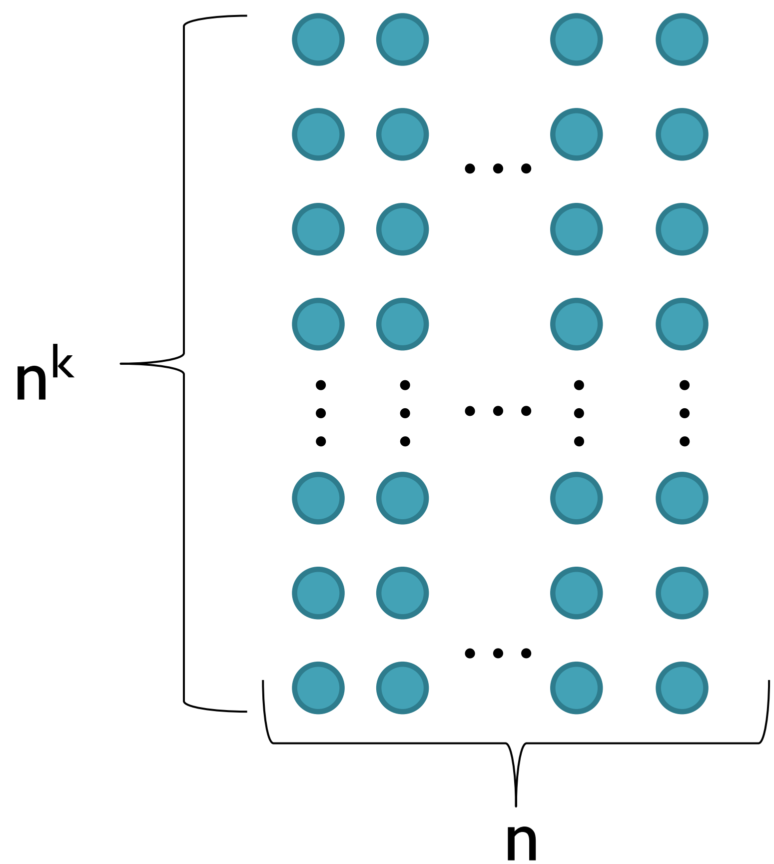
- The Network Inference Problem.
  - Given:
    - a set of vertices $V = \{v_1,\ldots,v_n\}$
    - costs c_e for each edge $e=\{v_i,v_j\}$
    - a constraint set $S = \{S_1,\ldots,S_r\}$, with $S_i \subseteq V$
  - Find: a set E of edges of lowest cost such that each S_i induces a connected subgraph of $G=(V,E)$
- We consider both the offline and online version of this problem. We also consider the arbitrary and uniform cost versions.
- Solved for the case where all constraints can be satisfied by a tree [Korach & Stern '03] – they left the general case open

An Offline Lower Bound

- Theorem: If P ≠ NP, the approximation ratio for the Uniform Cost Network Inference problem is $\Omega(\log n)$.
- Proof (reduction from Hitting Set)
  - $U = \{v_1, v_2, \ldots, v_n\}$
  - $C = \{C_1, C_2, \ldots, C_j\}$, with $C_i \subseteq U$
  - The Hitting Set problem is to minimize $|H|$, where $H \subseteq U \text{ s.t. } \forall C_i\ H \cap C_i \neq \phi$

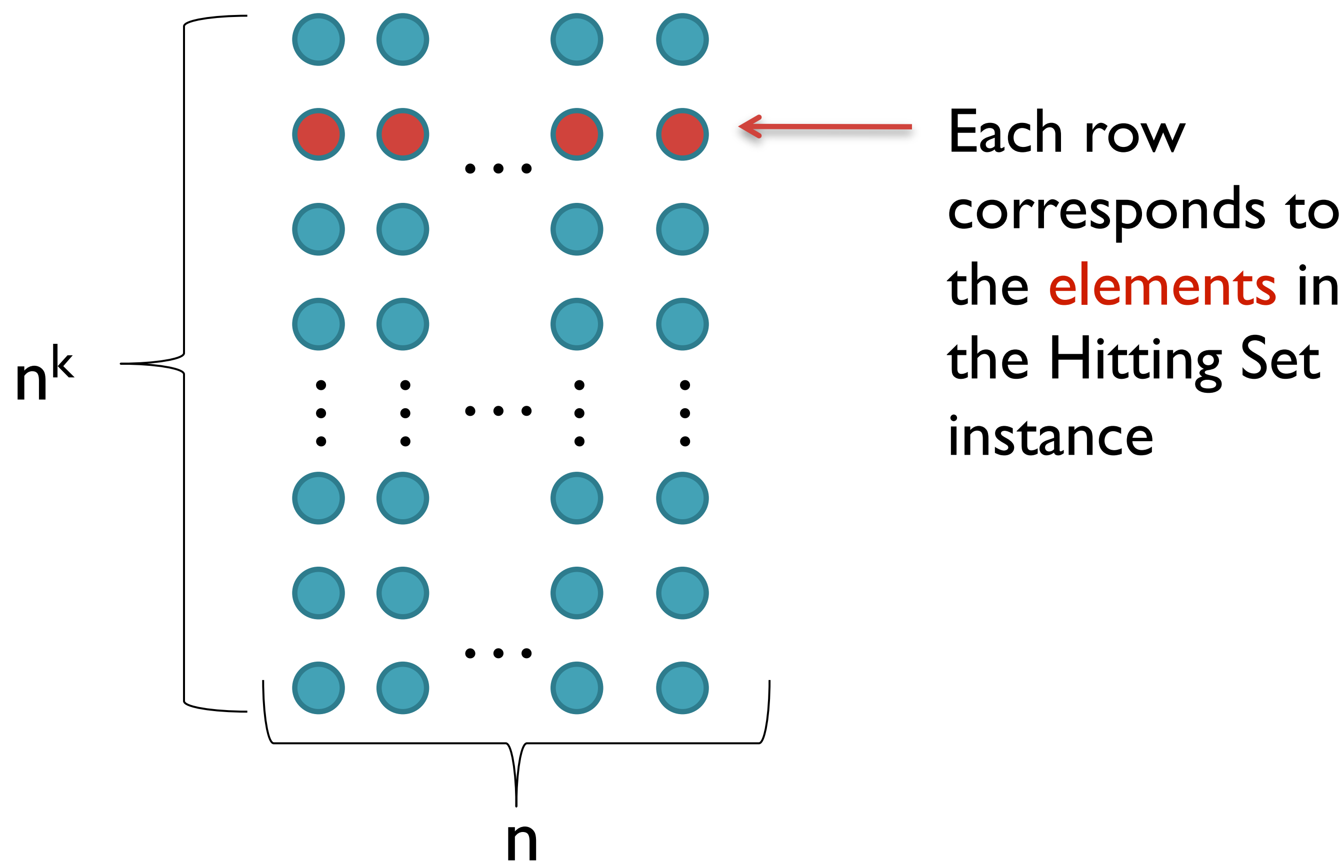
An Offline L.B. continued

- Reduction from Hitting Set
- For a constant k, We make a N.I. instance



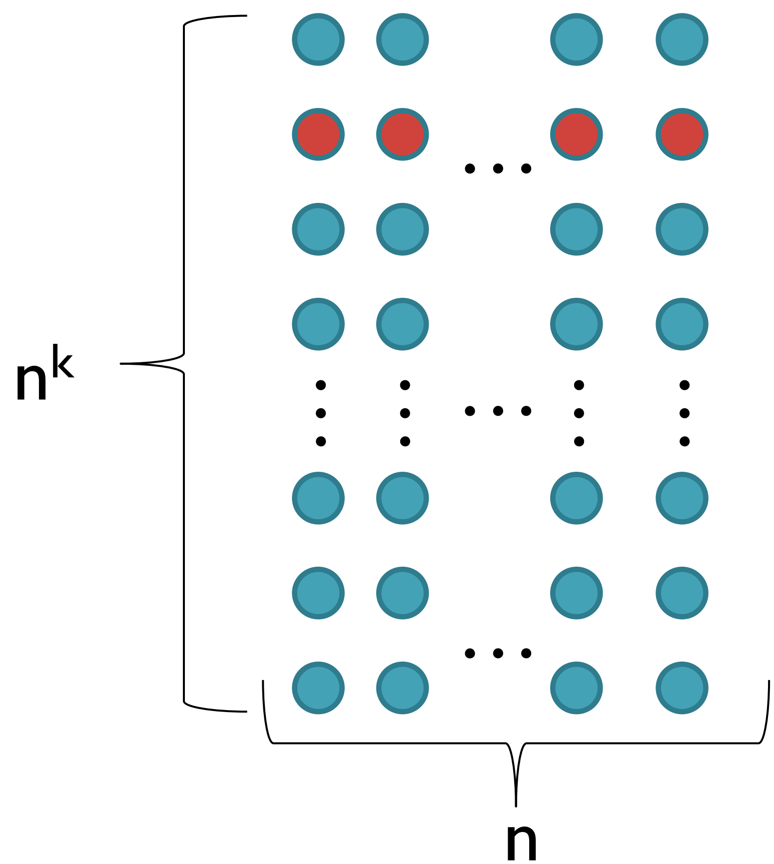
An Offline L.B. continued

- Reduction from Hitting Set
- For a constant k, We make a N.I. instance



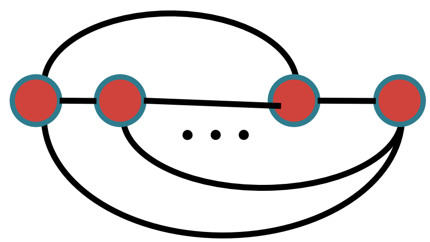
An Offline L.B. continued

- Constraints: first, for each row, give all pairwise constraints:



An Offline L.B. continued

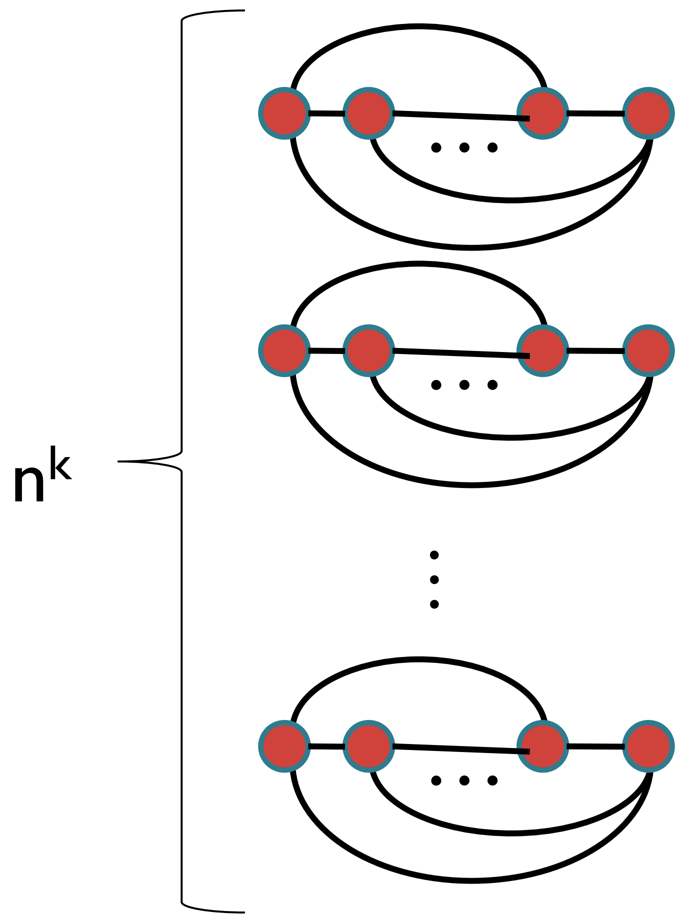
- Constraints: first, for each row, give all pairwise constraints:



- This will force the learner to put down a clique on each row

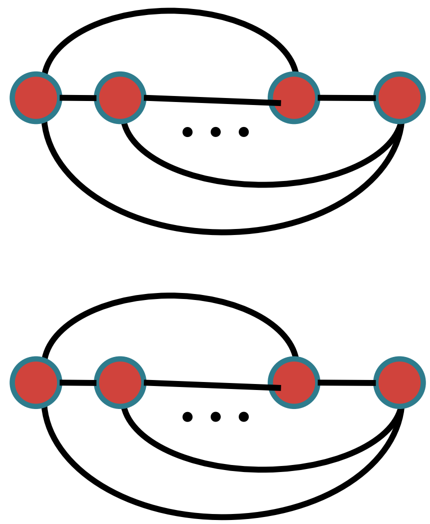
An Offline L.B. continued

- Now we have n^k rows of cliques



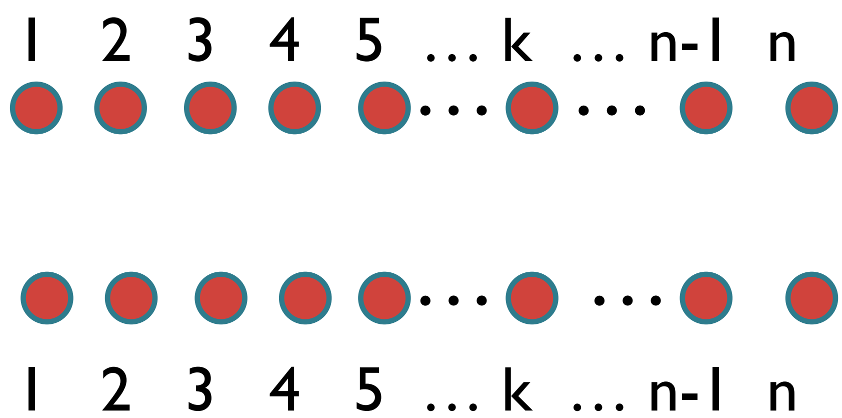
An Offline L.B. continued

- For each pair of rows:



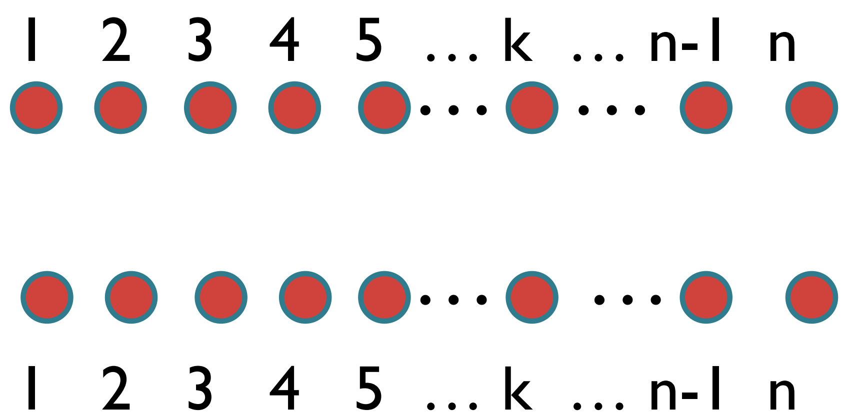
An Offline L.B. continued

- For each pair of rows:



An Offline L.B. continued

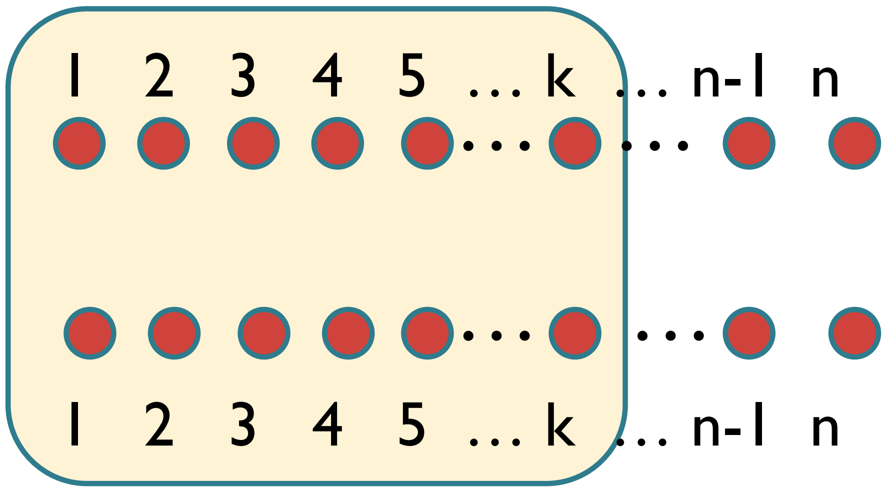
- For each pair of rows:



- w.l.o.g. for the Hitting Set constraint
  - $C_i = \{v_1, v_2, \ldots, v_k\}$
  - we will add the constraint:

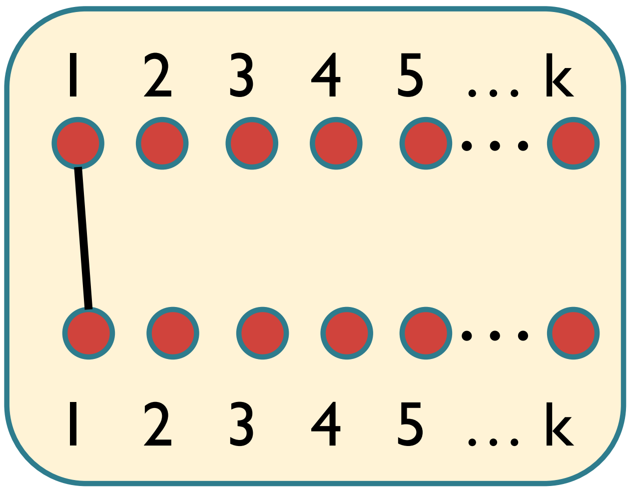
An Offline L.B. continued

- For each pair of rows:

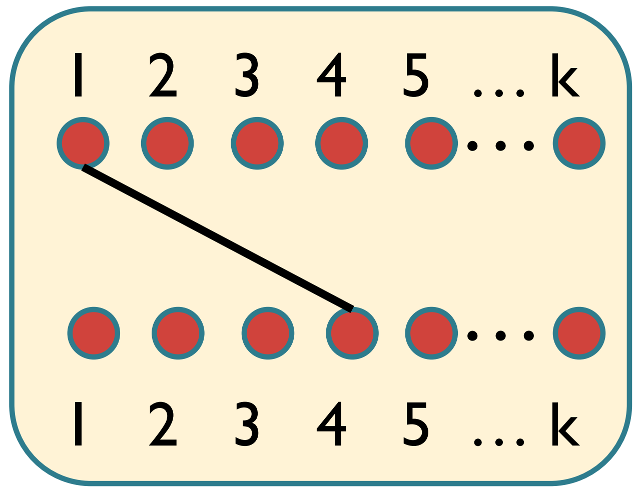


- w.l.o.g. for the Hitting Set constraint
  - $C_i = \{v_1, v_2, \ldots, v_k\}$
  - we will add the constraint:

An Offline L.B. continued

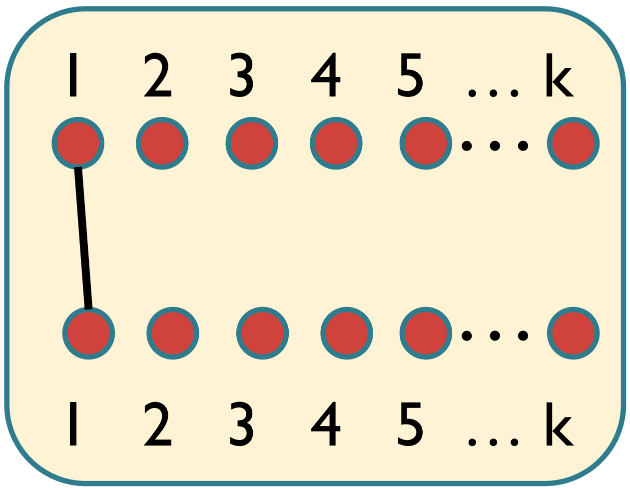


corresponds to adding v_1 to H

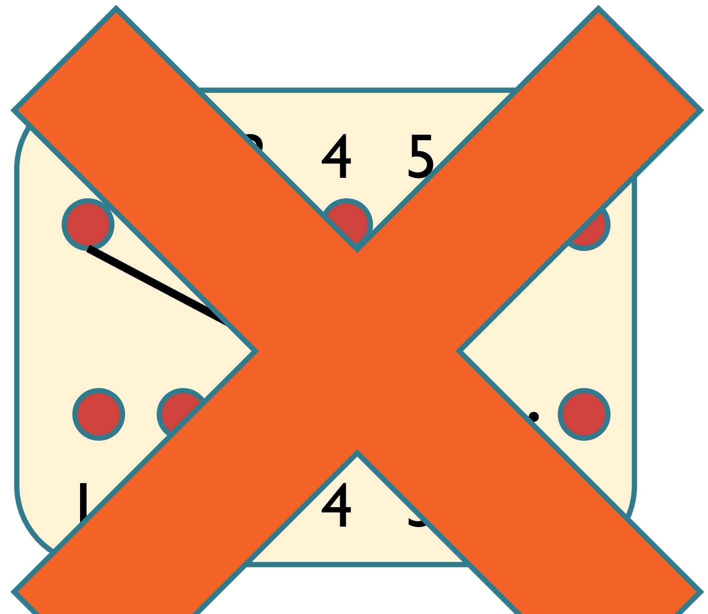


never better

An Offline L.B. continued



corresponds to adding v_1 to H



never better

Finishing the Lower Bound

- Unless P=NP, optimal Hitting Set approximation is Ω(log(n)) [Feige '98].
- The optimal algorithm pays:

$$n^k \binom{n}{2} + \mathrm{OPT}\binom{n^k}{2}$$

- But the learner pays:

$$n^k \binom{n}{2} + \Omega\left(\log(n)\mathrm{OPT}\binom{n^k}{2}\right)$$

- k can be chosen to be arbitrarily large.

Offline Network Inference Algorithm

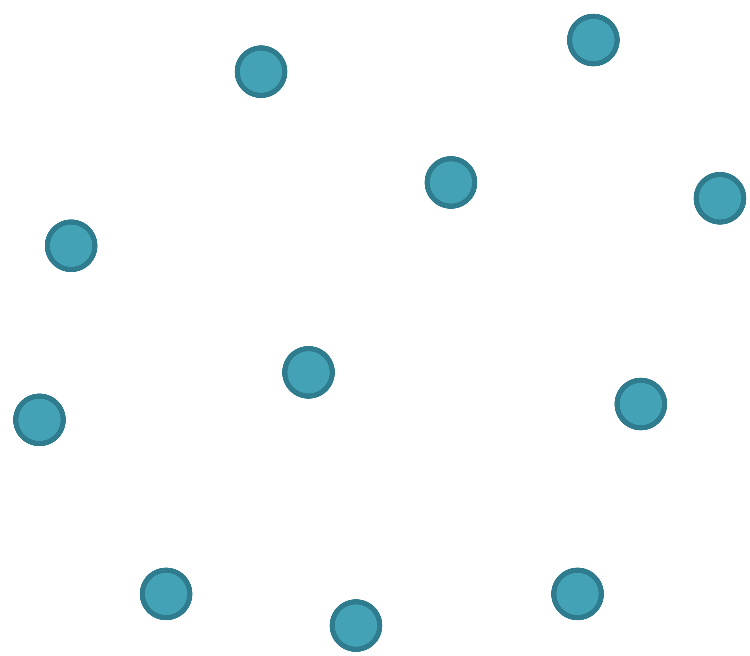
- Theorem: There is a $O(\log(n)+\log(r))$ approximation algorithm to OPT
- Proof:
  - Let **C** sum over all constraints S_i, the number of components S_i induces in G minus 1.
  - Now consider the greedy algorithm: while $C > 0$, add to E the edge that has the lowest ratio of c_e to ΔC.
  - This greedy algorithm gives an approximation of $\log(C_0) = O(\log(n)+\log(r))$

The Online Problem

- Constraints S_i come in online
- Must satisfy each constraint as it comes in.
- Can add but not remove edges.
- Seemingly good ideas like placing a spanning tree on each constraint can perform very badly.

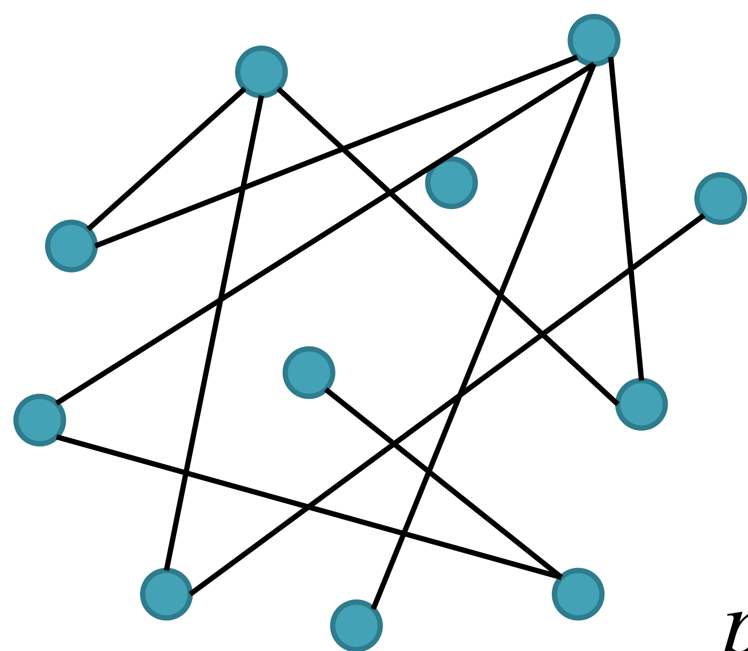
Online Algorithm Against Oblivious Adversary

$O(n^{2/3}\log^{2/3}n)$-competitive algorithm



Online Algorithm Against Oblivious Adversary

$O(n^{2/3}\log^{2/3} n)$-competitive algorithm



$$p = \frac{c\log^{2/3} n}{n^{1/3}}$$

Online Algorithm Against Oblivious Adversary

$O(n^{2/3}\log^{2/3}n)$-competitive algorithm

- All constraints S_i, $|S_i| \geq n^{1/3}\log^{1/3}(n)$ are almost surely connected
- All constraints S_i, $|S_i| < n^{1/3}\log^{1/3}(n)$ that are not already covered, we can put a clique on, and hit at least 1 edge in OPT
- We used $O(n^{5/3}\log^{2/3}(n)+n^{2/3}\log^{2/3}(n)\mathrm{OPT})$ edges in expectation.
- Because OPT = $\Omega(n)$, we are done.

Other Online Results

- The competitive ratio for uniform cost stars and paths is θ(log n).
  - for paths, makes use of pq-trees [Booth and Lueker '76]

- The uniform cost problem has a $\Omega\left(\sqrt{n}\right)$-competitive lower bound

- The arbitrary cost problem has an Ω(n)-competitive lower bound and O(n log n)-competitive algorithm.

Summary

- Lots of other results in this model.
- Passive model does not require interfering in the network.
- Interesting techniques, but gaps left.
- Would be interesting to extend to models incorporating incomplete observations.
- Extend to weaker adversaries or random networks.

Thank You!

Questions?