Active Learning of Interaction Networks

Lev Reyzin

thesis defense

<u>Committee</u>

Dana Angluin (advisor)

James Aspnes

Robert Schapire (Princeton)

Daniel Spielman

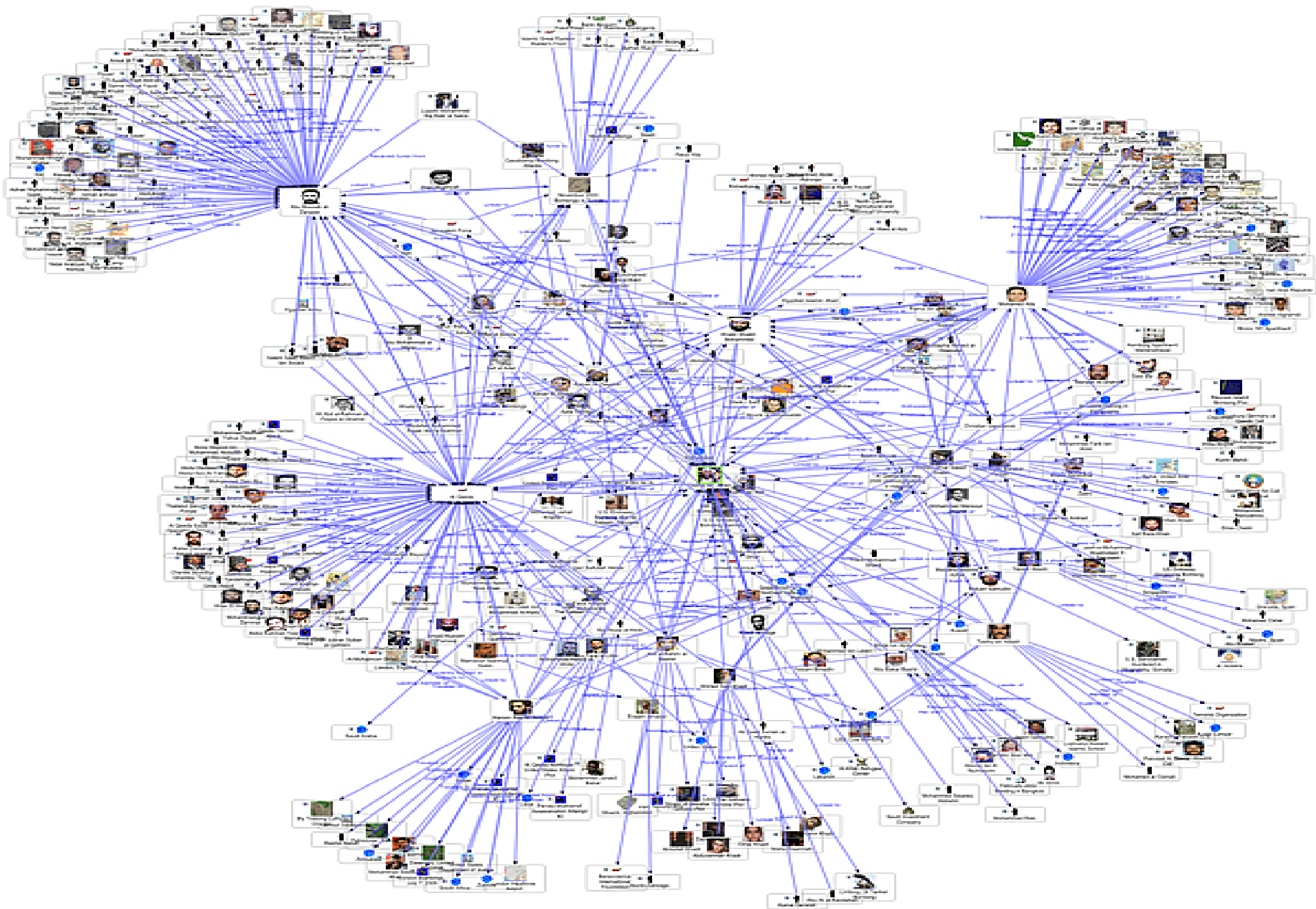
Which Pairs React?

An Experiment

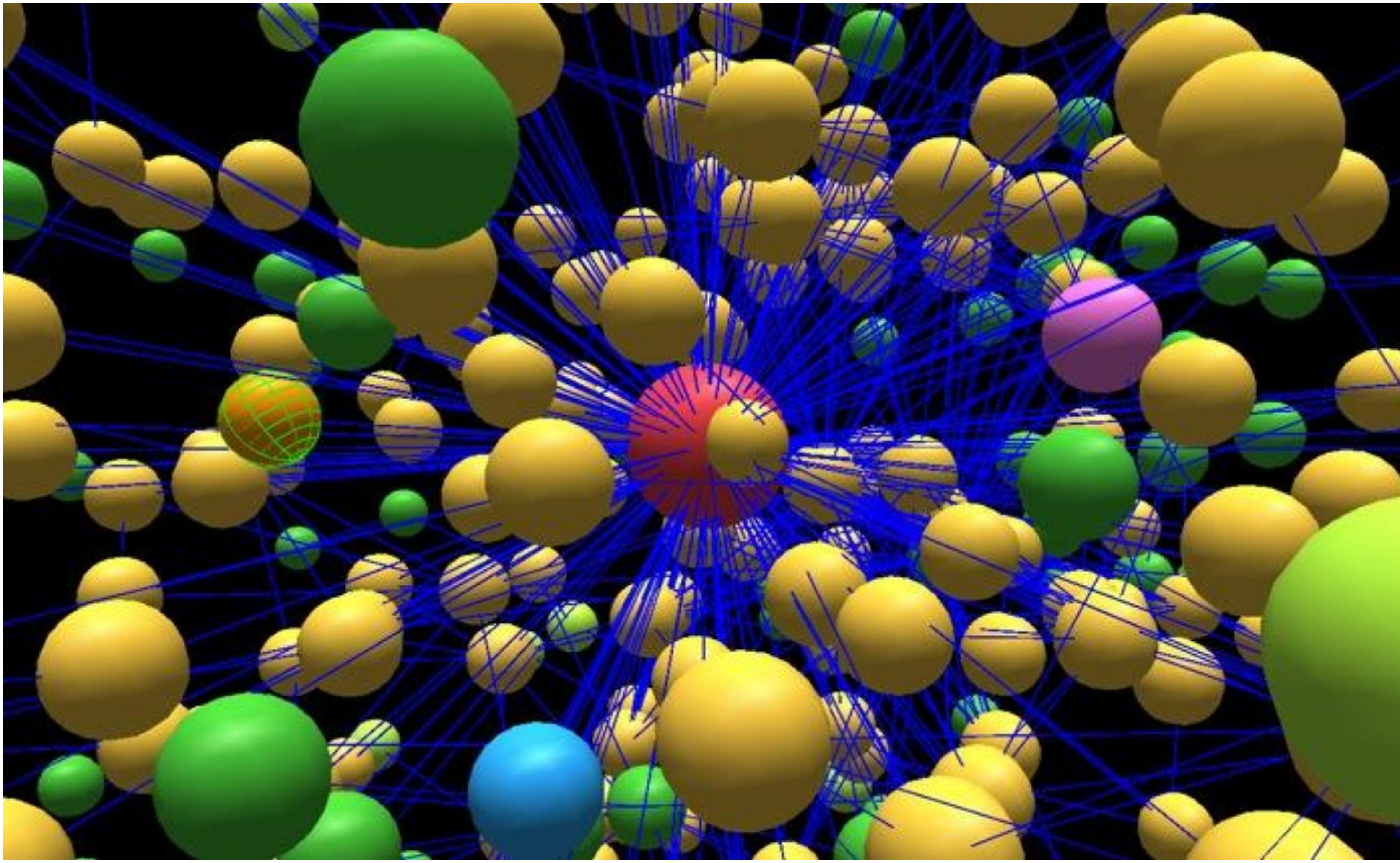
or

How to Mechanize this Process?

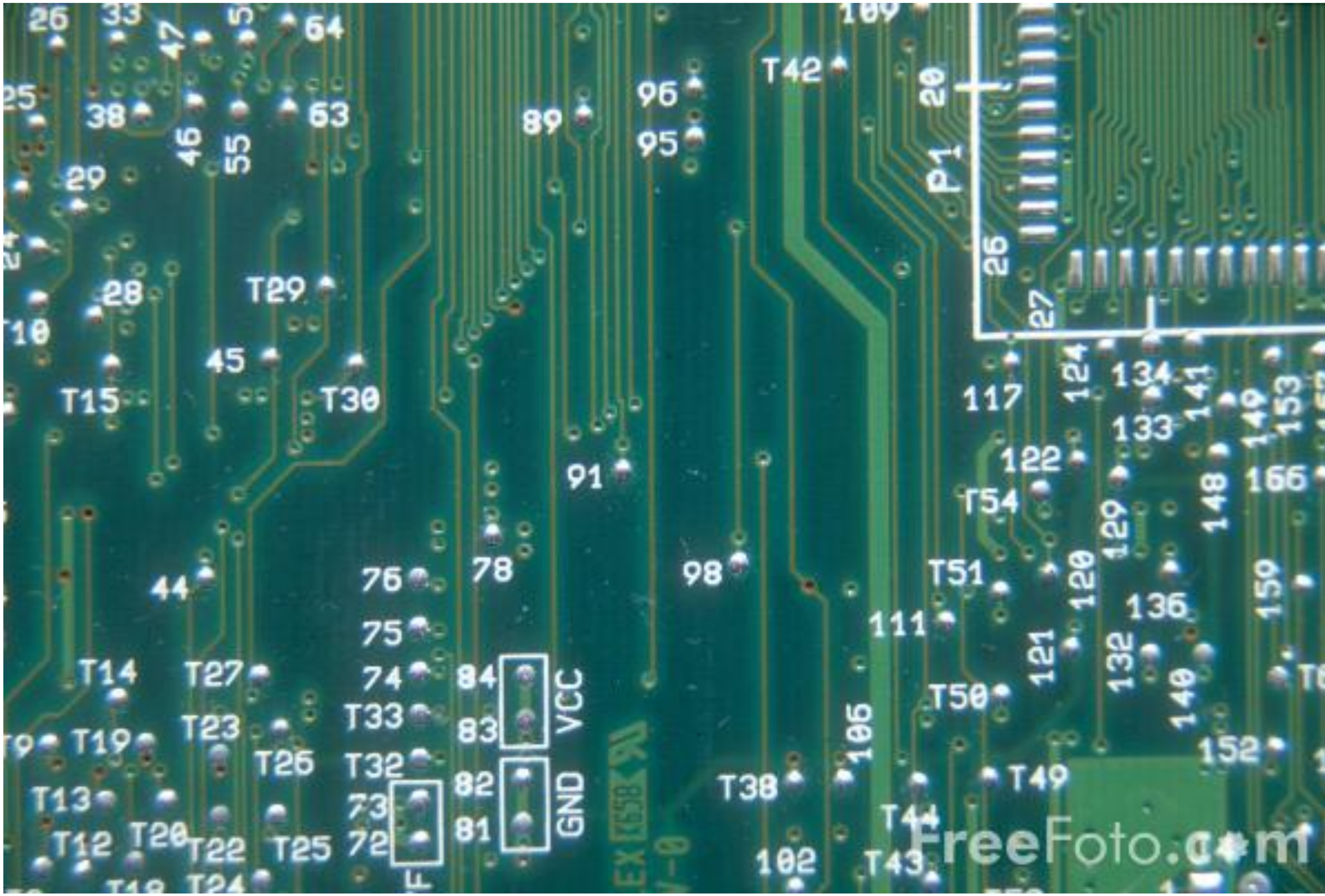
How Do We Learn Social Networks?



Learning Protein Networks



Testing Circuit Connectivity



Interaction Networks are Everywhere

Interaction Networks: finite populations of elements whose state may change as a result of interacting with other elements according to specific rules.

Active Learning

- In active learning, the learning algorithm has some choice in the data it learns from.
- An oracle responds to the learner's queries (questions or experiments) with information.
- **Many problems in discovering interaction networks can be modeled as active learning problems and analyzed from a machine learning viewpoint.**

Papers Covered in this Thesis

- Lev Reyzin and Nikhil Srivastava
  On the Longest Path Algorithm for Reconstructing Trees from Distance Matrices
  In *Information Processing Letters 2007*
- Lev Reyzin and Nikhil Srivastava
  Learning and Verifying Graphs Using Queries with a Focus on Edge Counting
  In *ALT 2007*
- Dana Angluin, James Aspnes, Jiang Chen, and Lev Reyzin
  Learning Large-Alphabet and Analog Circuits with Value Injection Queries
  In *COLT 2007* and *Machine Learning Journal 2008* Special Issue
- Dana Angluin, James Aspnes, and Lev Reyzin
  Optimally Learning Social Networks with Activations and Suppressions
  In *ALT 2008* and to appear in *Theoretical Computer Science* Special Issue
- Dana Angluin, James Aspnes, and Lev Reyzin
  Network Construction with Subgraph Connectivity Constraints
  Under submission to *SODA 2010*
- Dana Angluin, Leonor Becerra-Bonache, Adrian Horia Dediu, and Lev Reyzin
  Learning Finite Automata Using Label Queries
  To appear in *ALT 2009*

Papers Covered in this Thesis

Learning Evolutionary Trees

Learning Graphs (for DNA sequencing)

Learning Circuits (Gene Regulatory Networks)

Actively Learning Social Networks

Passively Inferring Social Networks

Learning Finite State Automata

Papers Covered in this Thesis

Learning Evolutionary Trees

Learning Graphs (for DNA sequencing)

Learning Circuits (Gene Regulatory Networks)

Actively Learning Social Networks

Passively Inferring Social Networks

Learning Finite State Automata

Papers Covered in this Thesis

Learning Evolutionary Trees

Learning Graphs (for DNA sequencing)

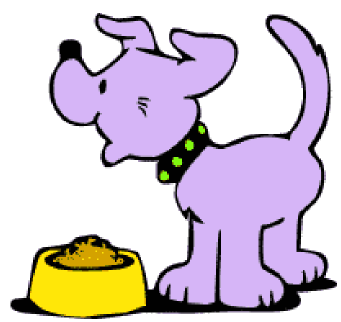
Learning Circuits (Gene Regulatory Networks)

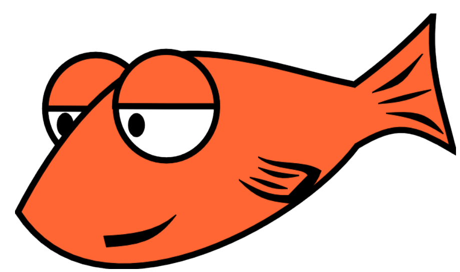
Actively Learning Social Networks

Passively Inferring Social Networks

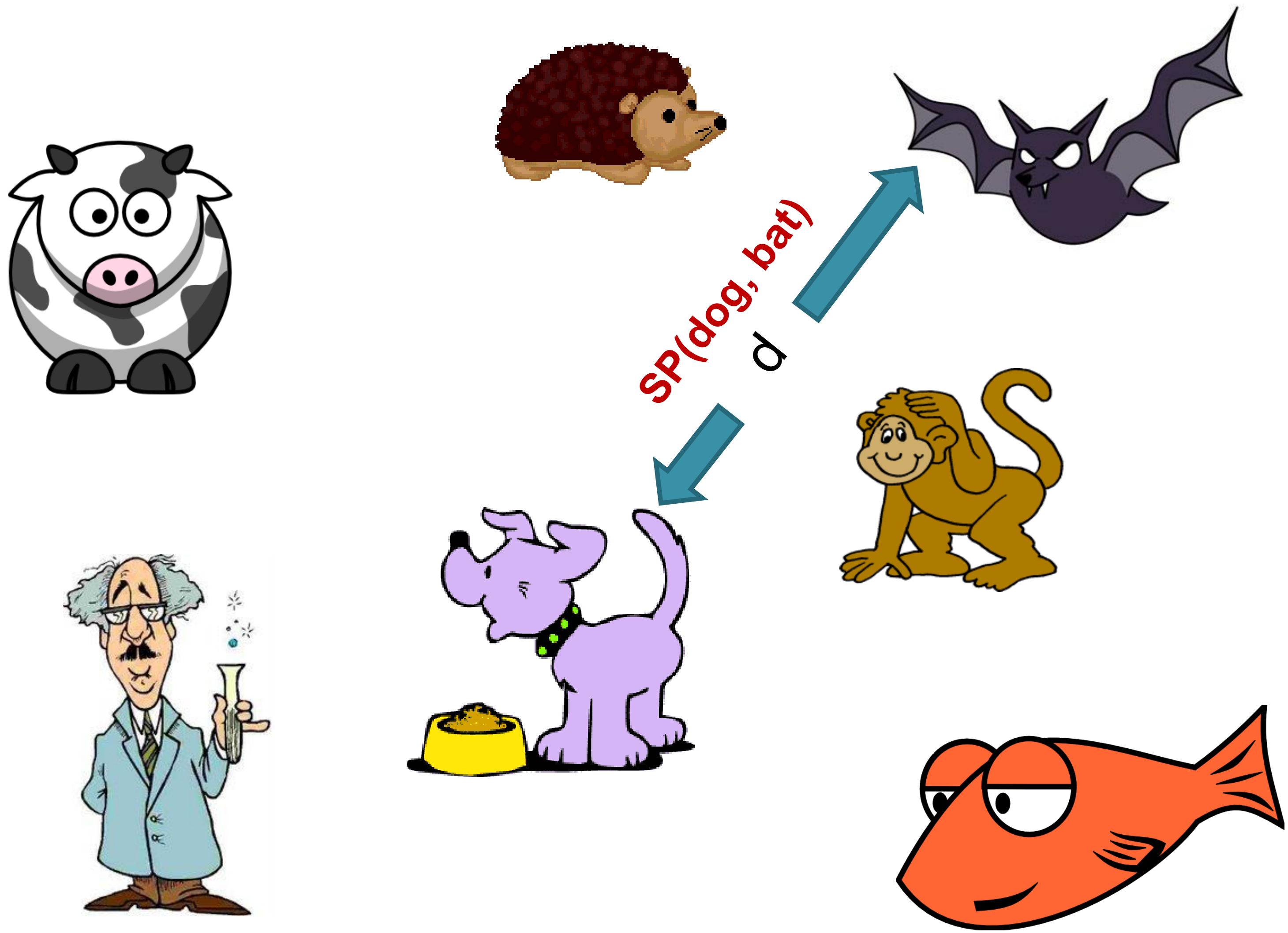
Learning Finite State Automata

Learning Evolutionary Trees

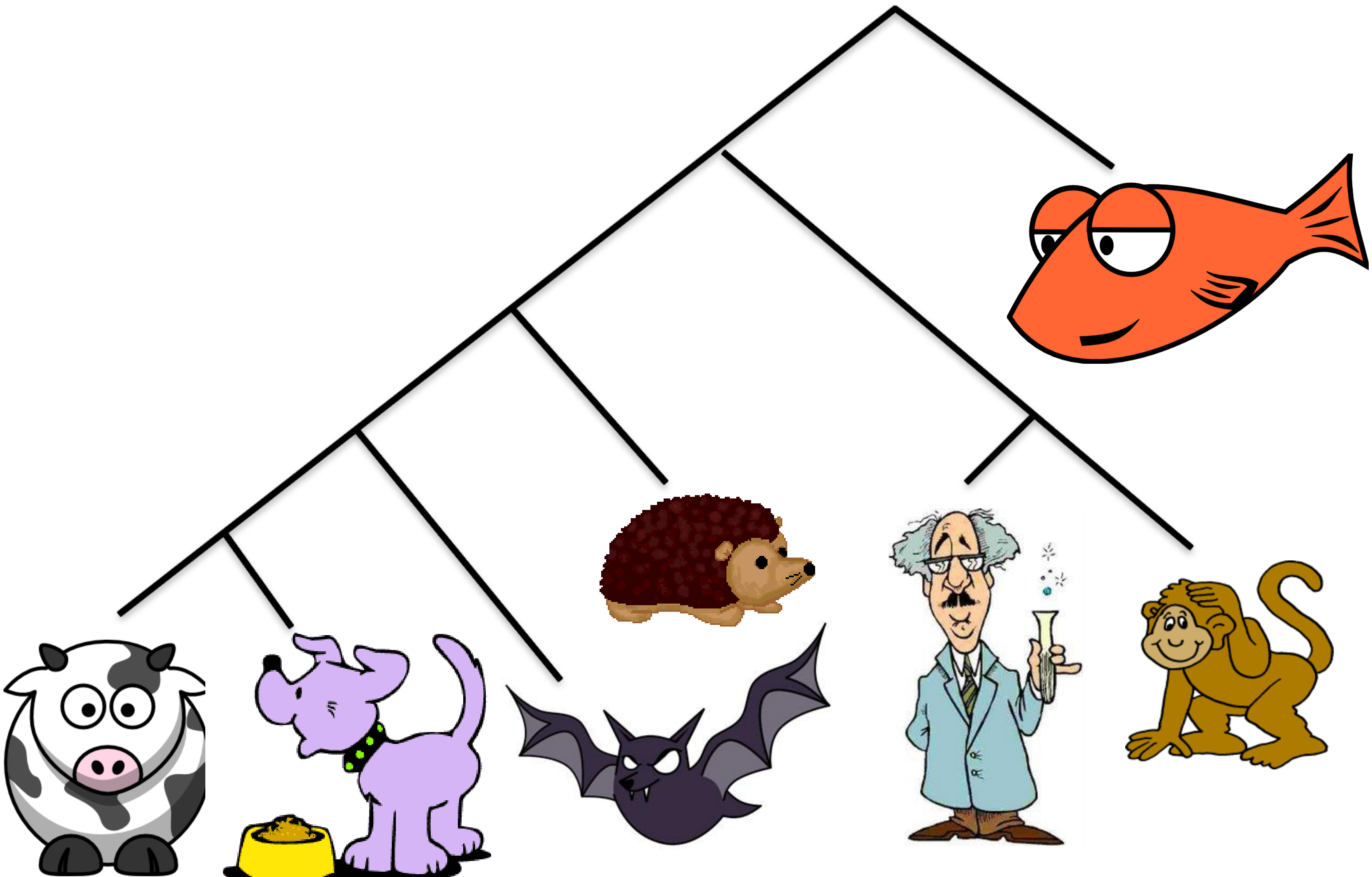




Test Genetic Distance



The Evolutionary Tree



For Degree Restricted Trees

- degree d trees can be learned using $O(dn \log_d n)$ queries [Hein '89].
  - This is also a lower bound [King *et al.* '03]

- The Longest Path algorithm [Culberson & Rudnicki '89] is often used for tree reconstruction and is widely cited.
  - We give the first correct analysis of Longest Path and show it uses $\Theta\left(n^{3/2}\sqrt{d}\right)$ queries.

Papers Covered in this Thesis

Learning Evolutionary Trees

Learning Graphs (for DNA sequencing)

Learning Circuits (Gene Regulatory Networks)

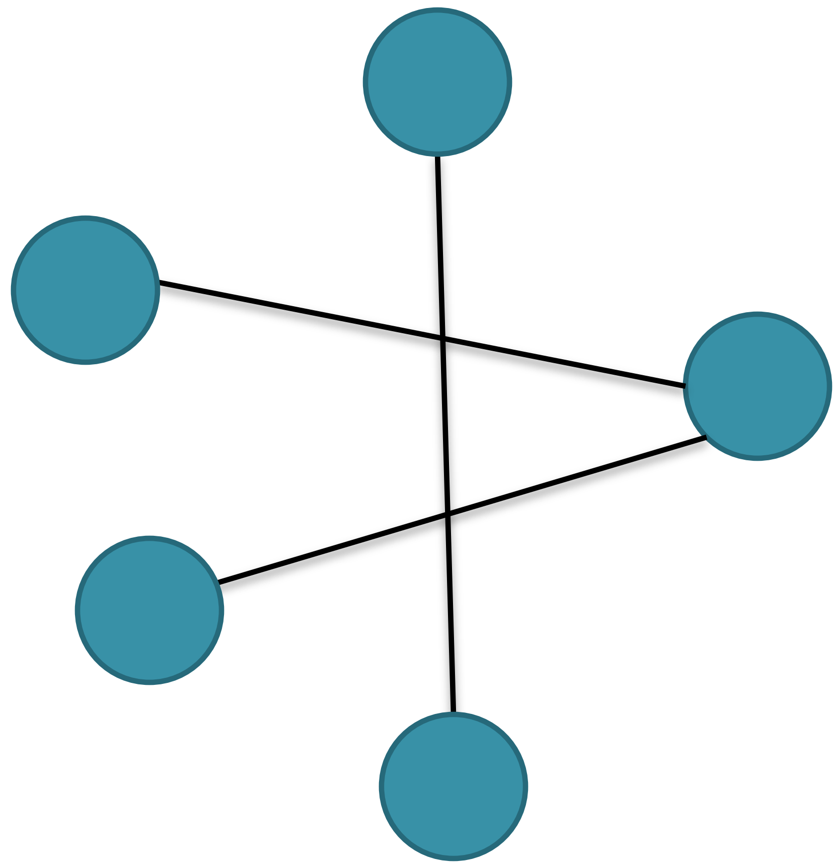
Actively Learning Social Networks

Passively Inferring Social Networks

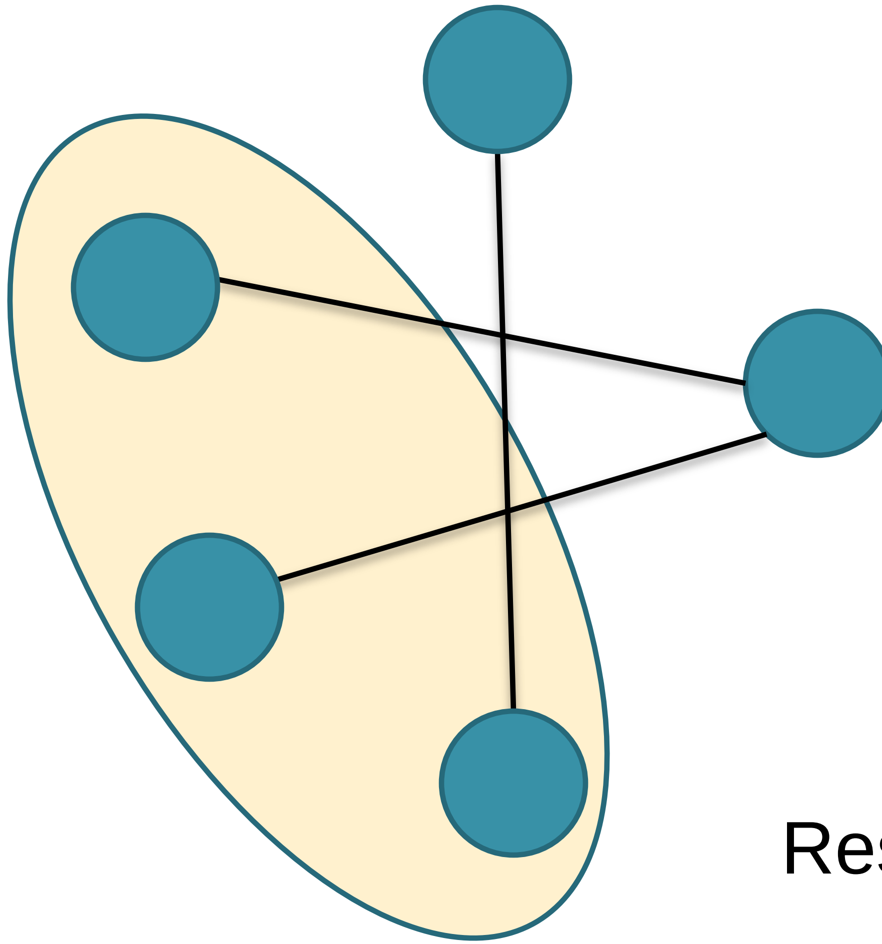
Learning Finite State Automata

Back to the Chemicals

Edge Detecting Queries

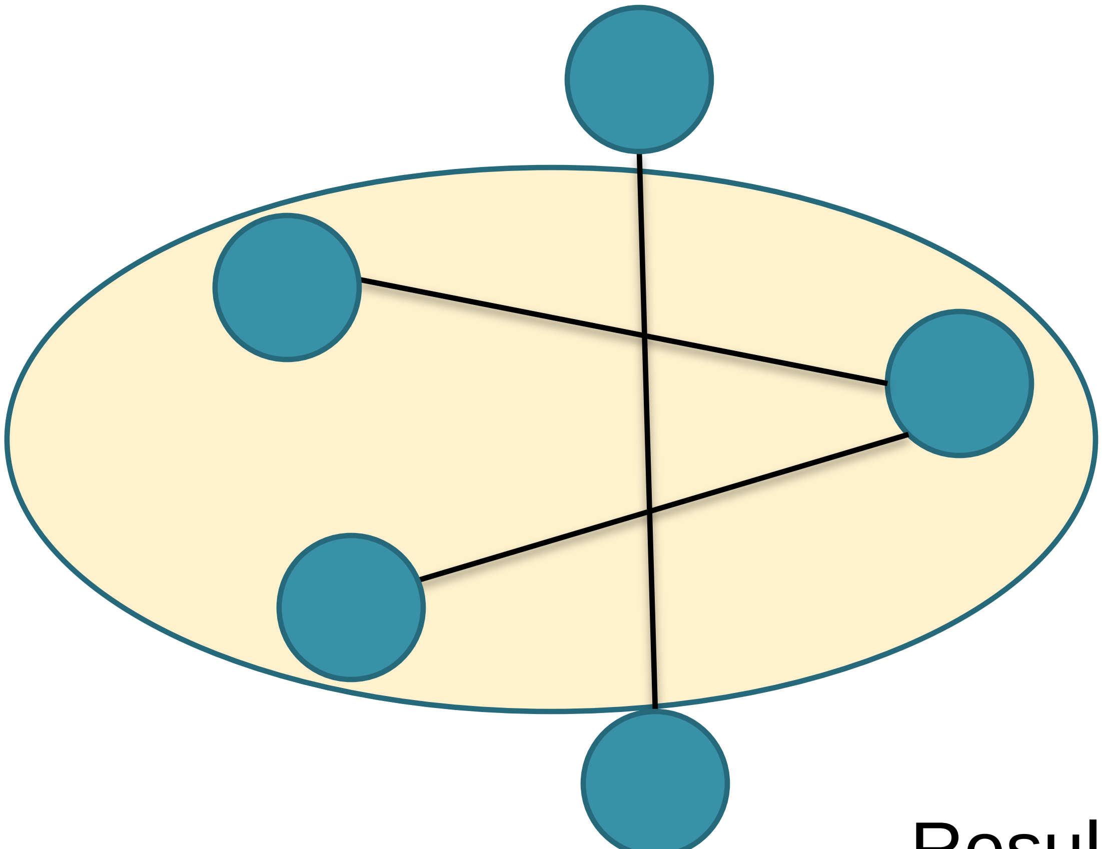


Edge Detecting Queries



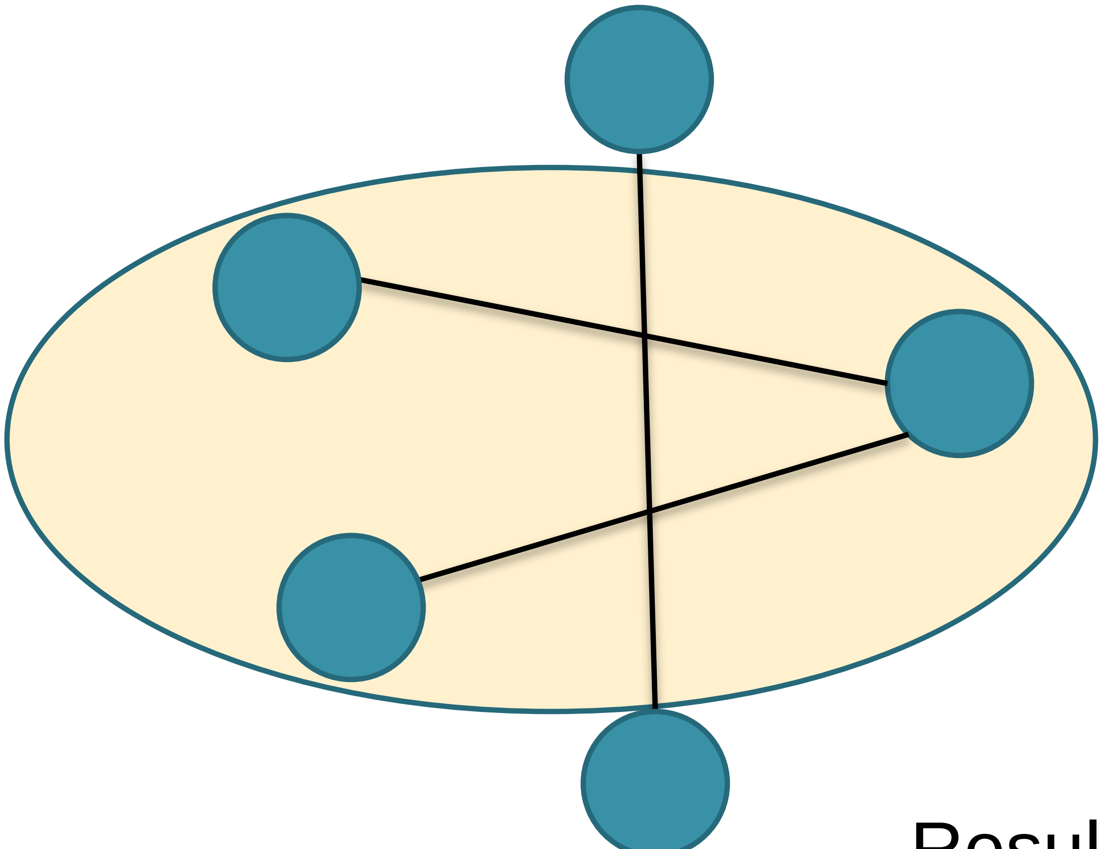
Result = 0

Edge Detecting Queries



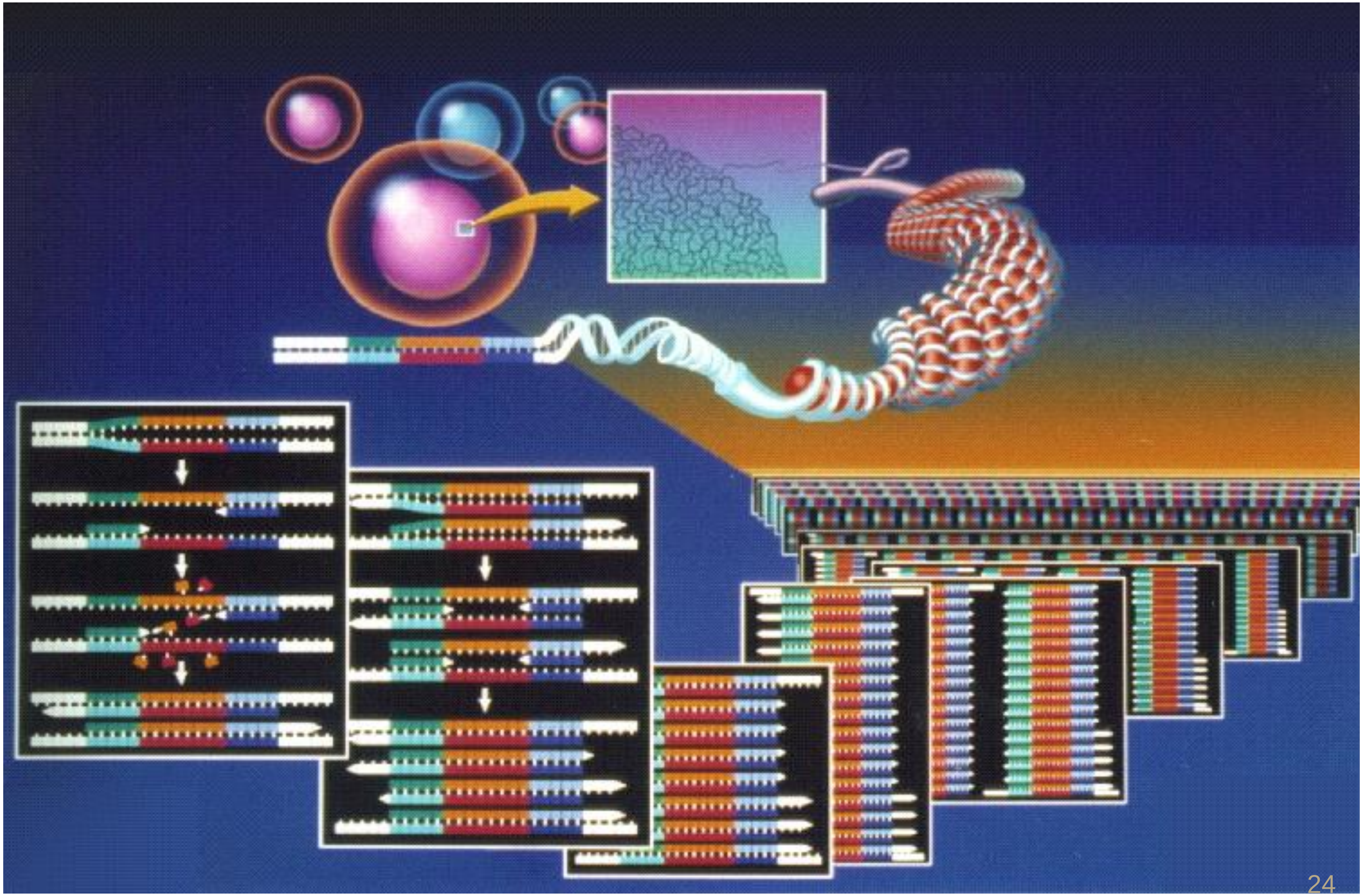
Result = 1

Edge Counting Queries



Result = 2

Multiplex PCR



Edge Detecting vs Counting Queries

G = (V,E), learner is given V and must discover E

Edge Detecting Query:

$S \subseteq V$

$$\mathrm{ED}(S) = \begin{cases} 1 & \text{if } \exists\, e \in S \\ 0 & otherwise \end{cases}$$

Edge Counting Query:

$S \subseteq V$

$$\mathrm{EC}(S) = \text{number of edges in } S$$

Edge Detecting vs Counting Queries

G = (V,E), learner is given V and must discover E

<u>*Edge Detecting Query:*</u>

$S \subseteq V$

$$\mathrm{ED}(S) = \begin{cases} 1 & \text{if } \exists\, e \in S \\ 0 & otherwise \end{cases}$$

arbitrary graphs:
[Angluin and Chen ’04]
hidden matching:
[Alon *et al.* ’04]
Hamiltonian Cycle:
[Grebinski and Kucherov ’98]

<u>*Edge Counting Query:*</u>

$S \subseteq V$

$$\mathrm{EC}(S) = \text{number of edges in } S$$

trees, degree bounded graphs:
[Grebinski Kucherov ’00]
optimal algorithm:
[Choi and Kim ’08]
k-degenerate graphs + survey:
[Bouvel *et al.* ’05]

Results for Poly-time Algorithms

finding connected components ↓

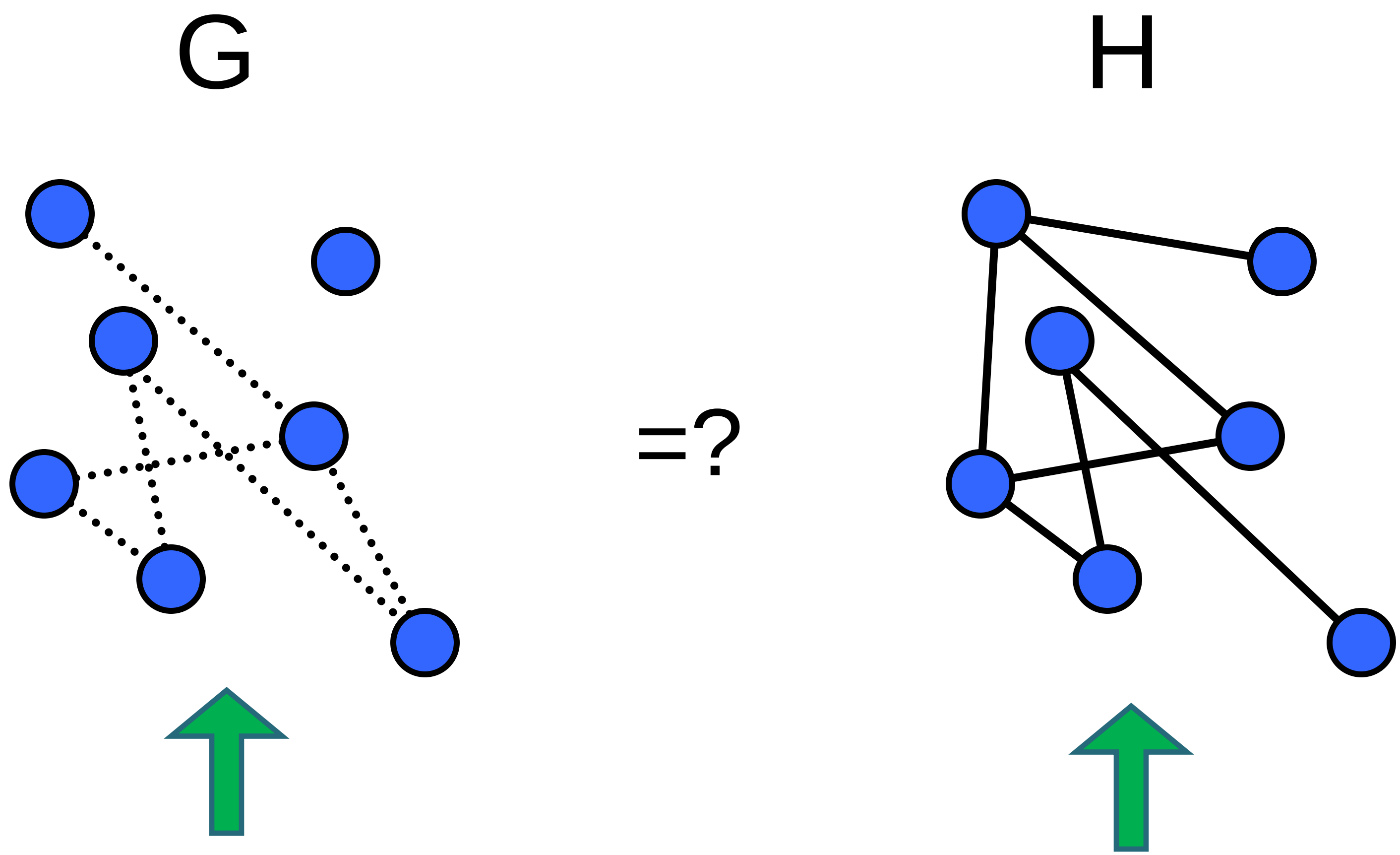
arbitrary graphs ↓

when the target is a tree ↓

	partition	graph	tree
ED	$\Theta(n^2)$	$\Theta(\|E\| \lg n)$ $\Theta(n^2)$	$\Theta(n \lg n)$
EC	$O(n\log n)$ $\Omega(n)$	$O(\|E\| \lg n)$* $\Theta(dn)$ $\Theta(n^2/\lg n)$	$\Theta(n)$
SP	$\Theta(nk)$	$\Theta(n^2)$	$\Theta(n^2)$* $\Theta(dn \lg_d n)$

k = number of components, *= some contribution

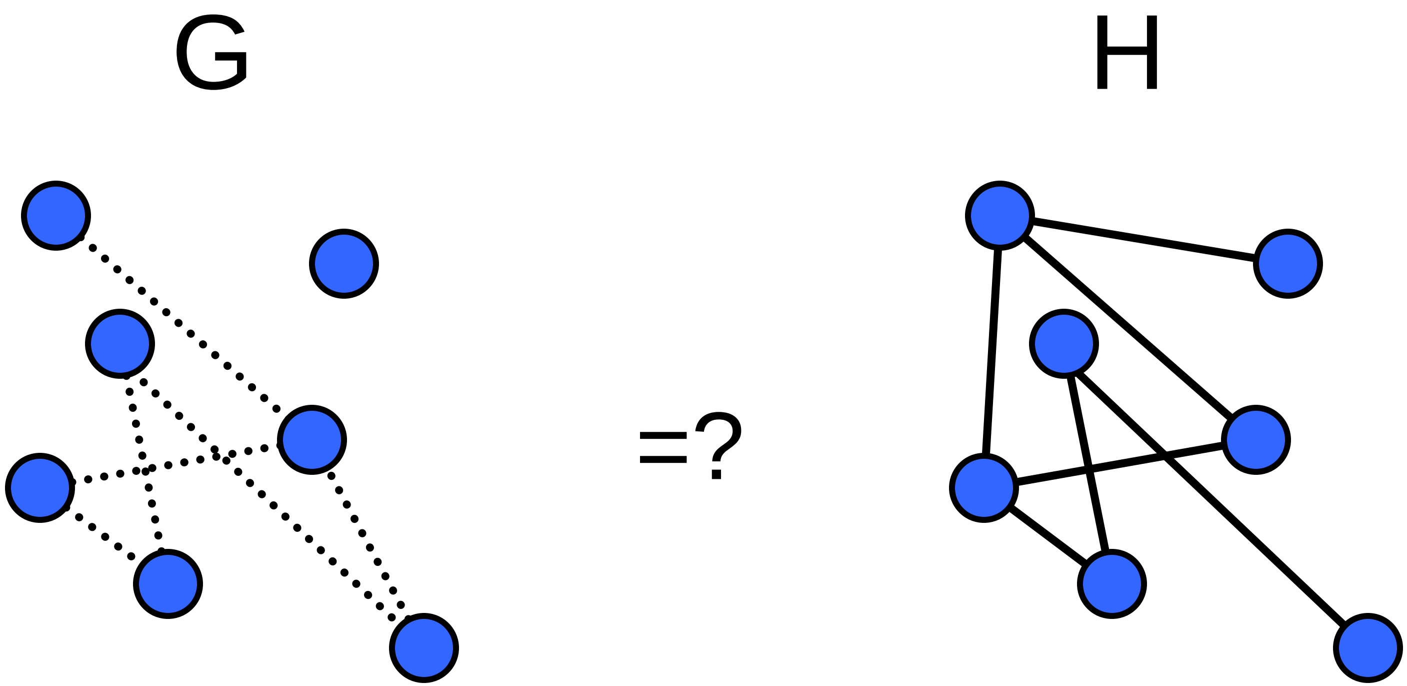
Verification with EC Queries



Have EC query access to this graph

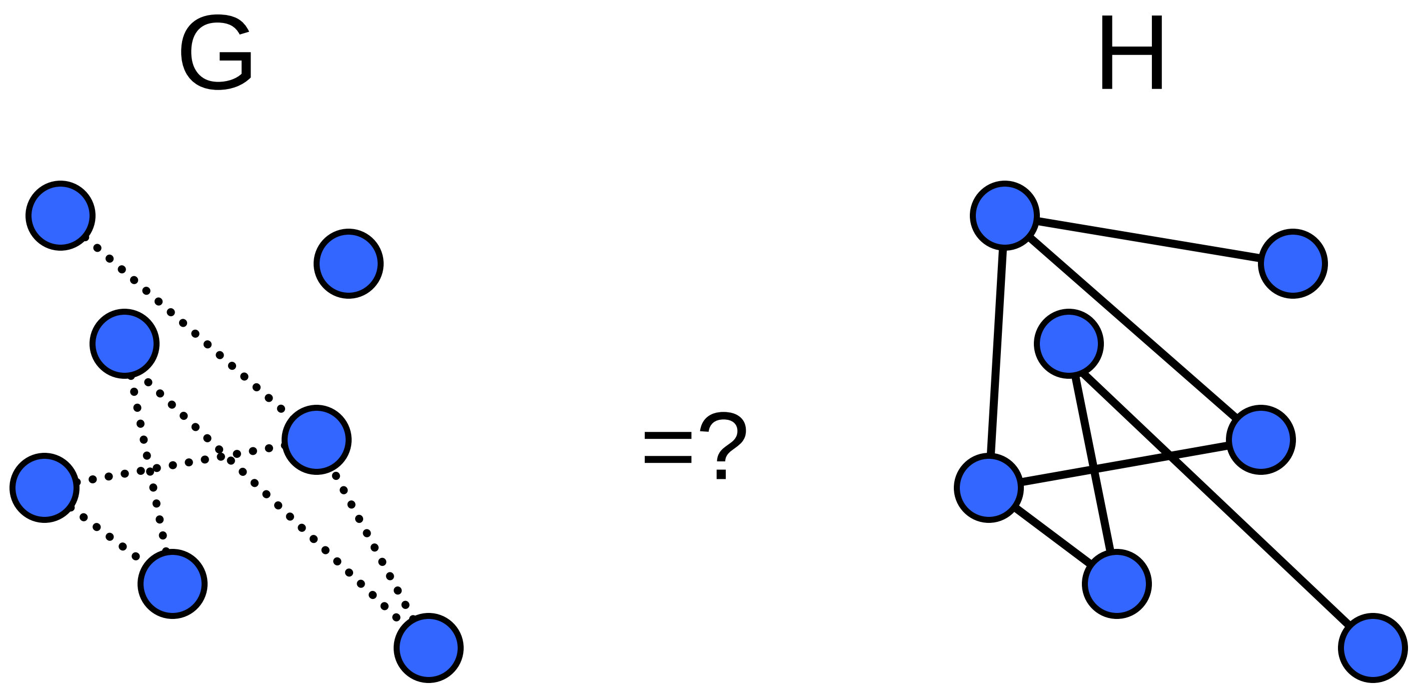
Given this graph

Verification with EC Queries



motivation: check for errors in learning

Verification with EC Queries



no harder than learning

Verification with EC Queries

Theorem: If $G \neq H$ then with prob. $\geq$ ¼ $EC_G(S) \neq EC_H(S)$ for a random subset S

Verification with EC Queries

Theorem: If $G \neq H$ then with prob. $\geq$ ¼ $EC_G(S) \neq EC_H(S)$ for a random subset S

To prove this theorem, we will first need to prove the following lemma:

Lemma: A random subset of vertices of a non-empty graph induces an **odd** number of edges w.p. at least 1/4.

Verification with EC Queries

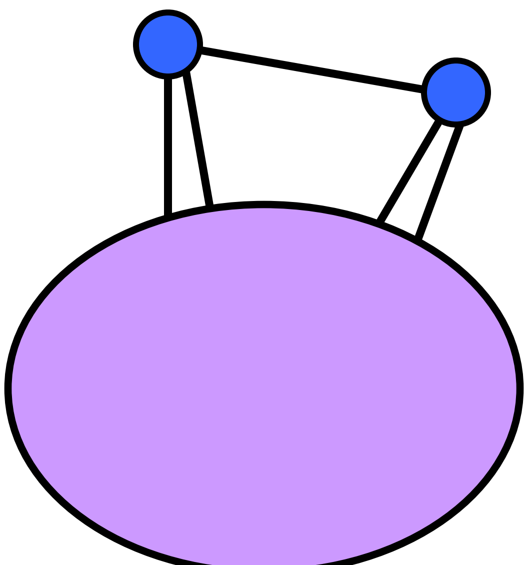
Lemma: A random subset of vertices of a non-empty graph induces an **odd** number of edges with probability at least 1/4.

Verification with EC Queries

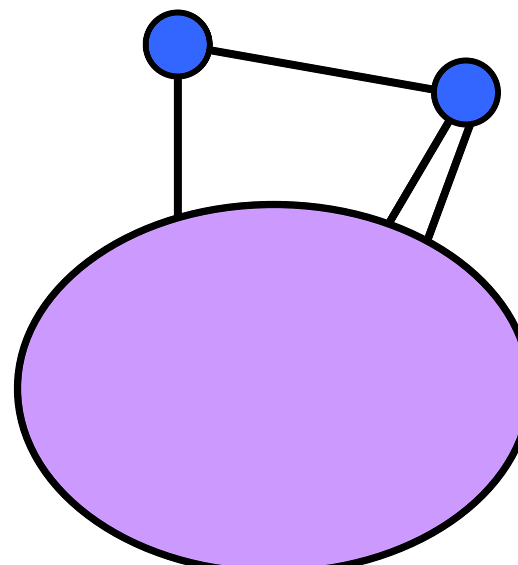
- Proof of Lemma: Order the vertices $\{v_1 \ldots v_n\}$ so $(v_{n-1}, v_n) \in E$. Choose in order with Pr. ½.

Verification with EC Queries

- Proof of Lemma: Order the vertices $\{v_1 \ldots v_n\}$ so $(v_{n-1}, v_n) \in E$. Choose in order with Pr. ½.

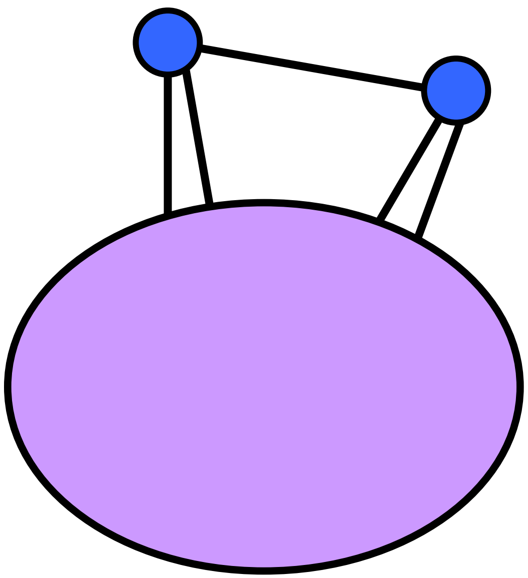


2 cases



Verification with EC Queries

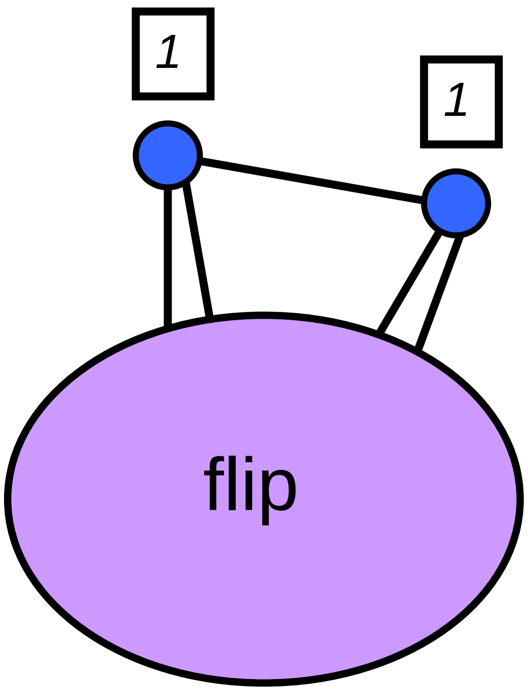
- Proof of Lemma: Order the vertices $\{v_1 \ldots v_n\}$ so $(v_{n-1}, v_n) \in E$. Choose in order with Pr. ½.



even case

Verification with EC Queries

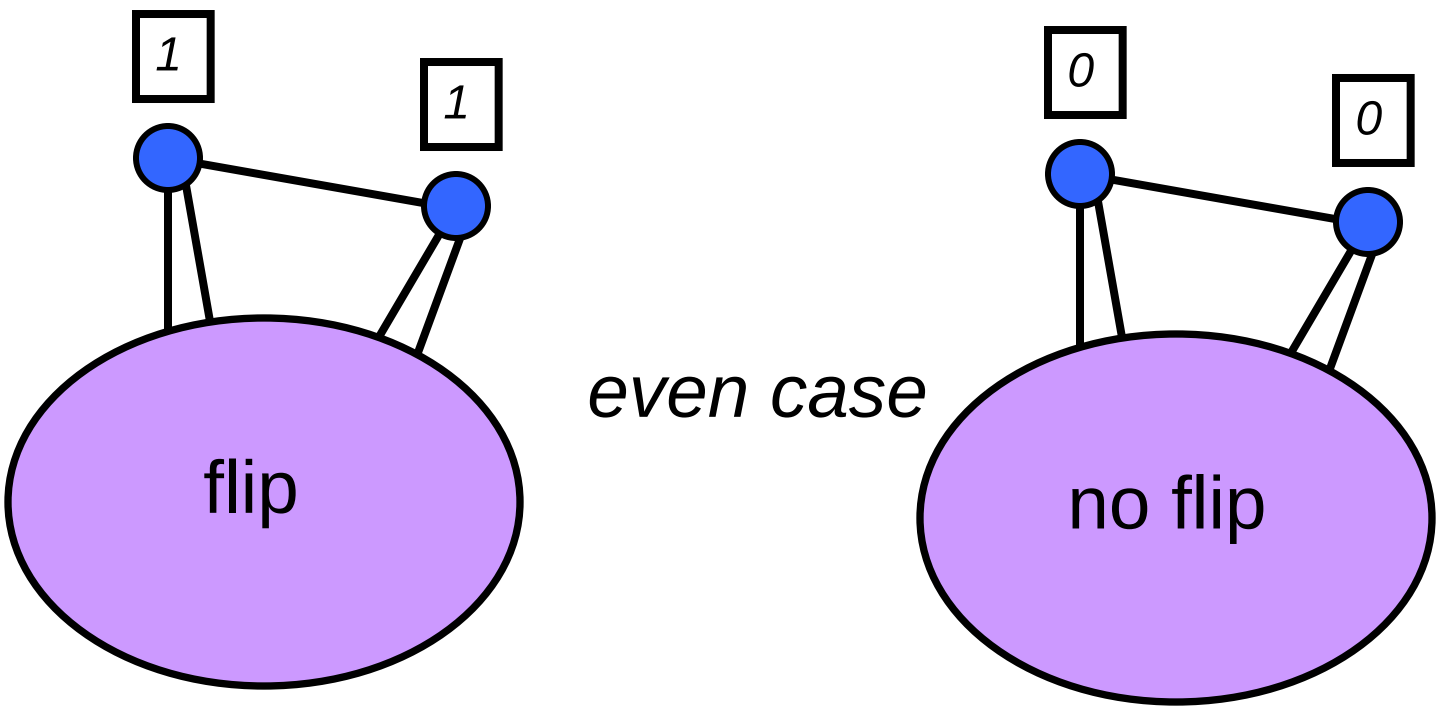
- Proof of Lemma: Order the vertices $\{v_1 \dots v_n\}$ so $(v_{n-1}, v_n) \in E$. Choose in order with Pr. ½.



even case

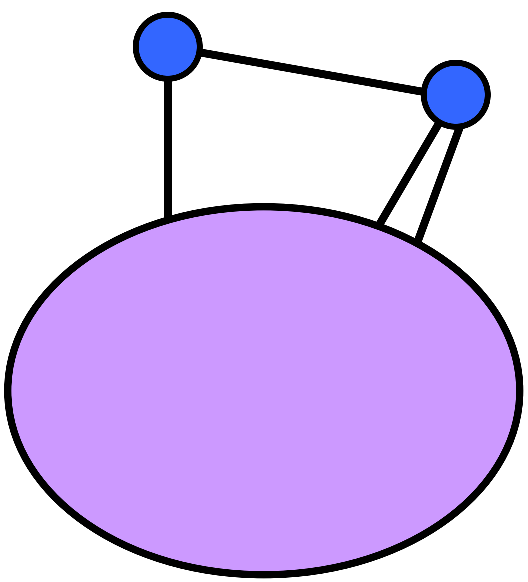
Verification with EC Queries

- Proof of Lemma: Order the vertices $\{v_1 \ldots v_n\}$ so $(v_{n-1}, v_n) \in E$. Choose in order with Pr. ½.



Verification with EC Queries

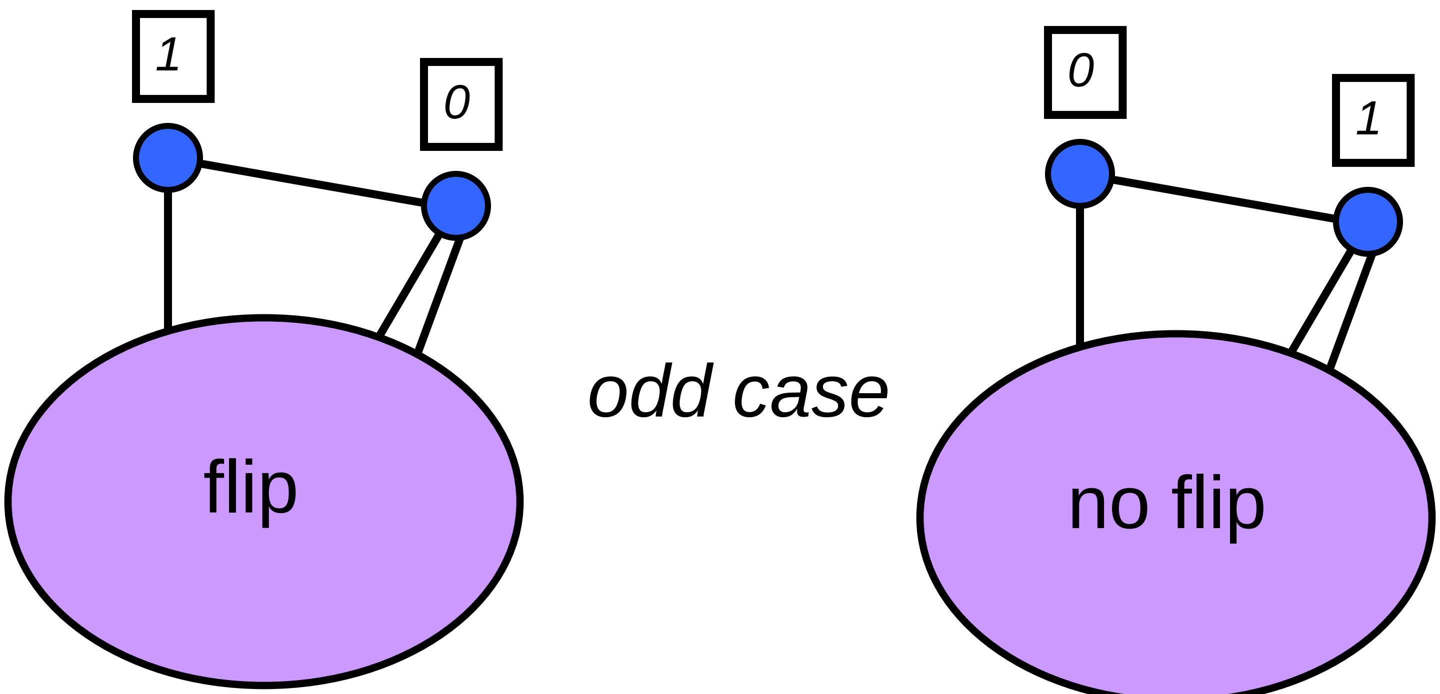
- Proof of Lemma: Order the vertices $\{v_1 \ldots v_n\}$ so $(v_{n-1}, v_n) \in E$. Choose in order with Pr. ½.



odd case

Verification with EC Queries

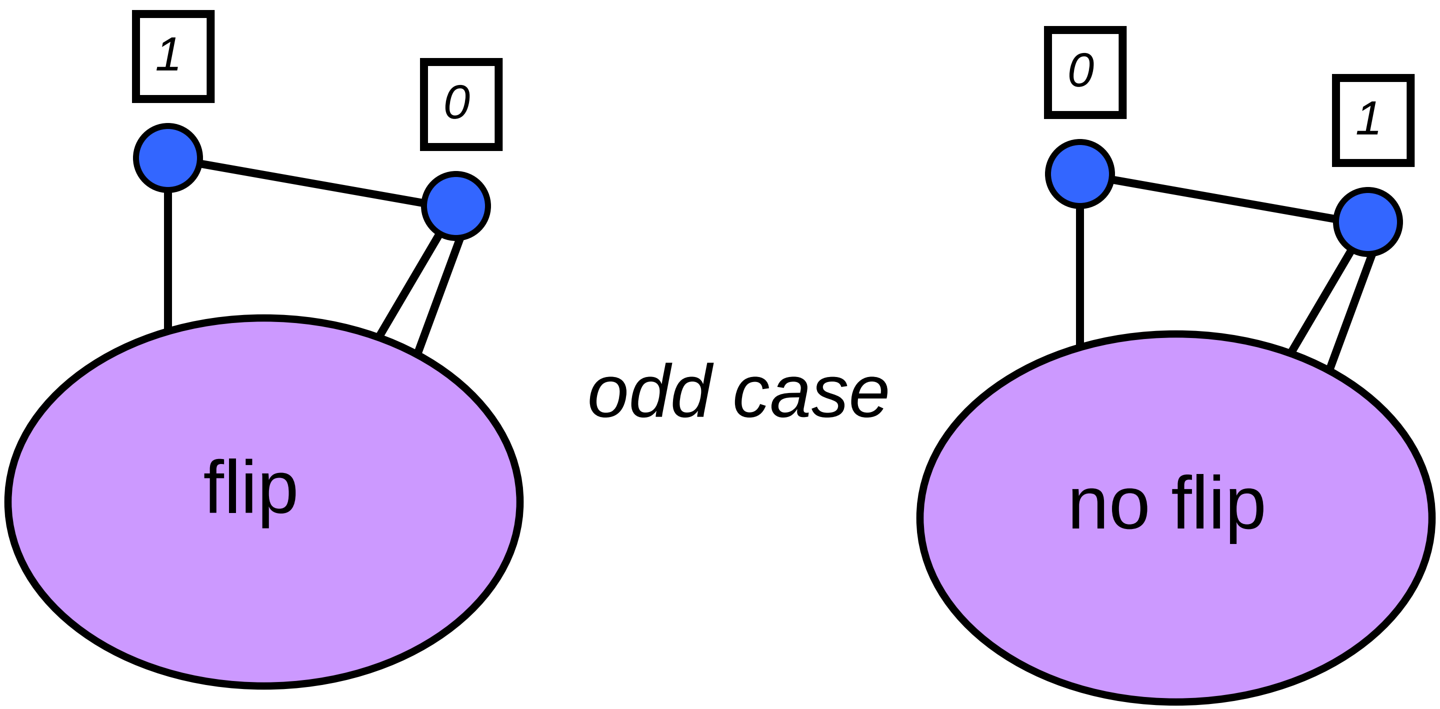
- Proof of Lemma: Order the vertices $\{v_1 \ldots v_n\}$ so $(v_{n-1}, v_n) \in E$. Choose in order with Pr. ½.



odd case

Verification with EC Queries

- Proof of Lemma: Order the vertices $\{v_1 \dots v_n\}$ so $(v_{n-1}, v_n) \in E$. Choose in order with Pr. ½.



□

Verification with EC Queries

- Theorem: If $G \neq H$ then with prob. $\geq$ ¼ $EC_G(S) \neq EC_H(S)$ for a random subset S

- proof:
  - If $G \neq H$, then $G \Delta H \neq \emptyset$
  - If $EC_{G\Delta H}(S)$ is odd, then $EC_G(S) \neq EC_H(S)$
  - If $G\Delta H \neq \emptyset$, and S is chosen uniformly at random, then with probability $\geq$ ¼, $EC_{G\Delta H}(S)$ has an odd number of edges.
    - by the lemma
  - So if $G \neq H$ a random query (which we perform on G and simulate on H) will expose the difference with probability $\geq 1/4$ $\square$

Verification with EC Queries

- Can boost the probability by repeating the random queries
  - Any graph can be verified by a randomized algorithm with error ε using $O(\log(1/\varepsilon))$ EC queries.

- Has a relationship to matrix fingerprinting [Freivalds '77]:
  - For a large class of matrices, we can fingerprint with less randomness.

Papers Covered in this Thesis

Learning Evolutionary Trees

Learning Graphs (for DNA sequencing)

Learning Circuits (Gene Regulatory Networks)

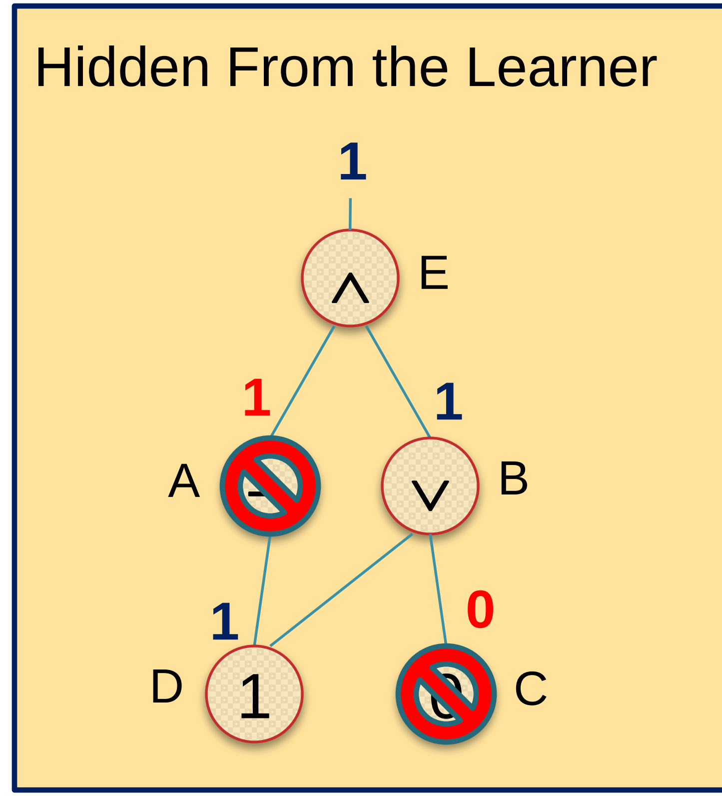
Actively Learning Social Networks

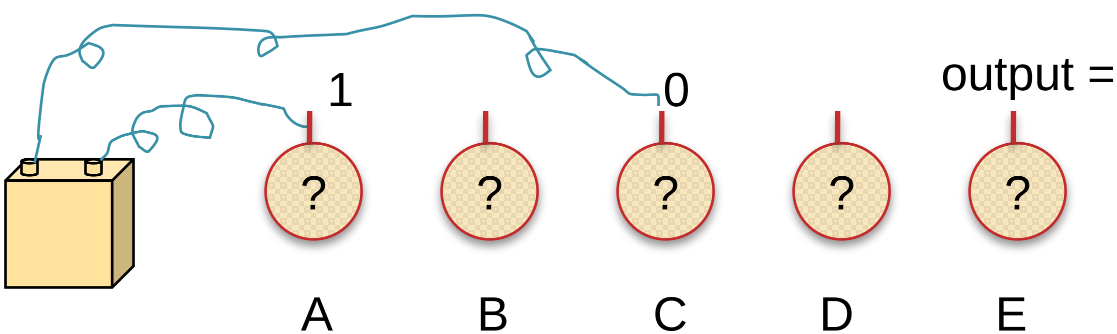
Passively Inferring Social Networks

Learning Finite State Automata

The Value Injection Query Model

- [Angluin *et al.* ’06]
- Experiments on a hidden Circuit.
  - a gate output may be fixed
  - a gate may be left free
- Query
  - given an experiment, we can observe its output
- Example:





Large Alphabet Circuit Results

- Theorem: An algorithm for learning log depth circuits polynomial in the number of wires and alphabet size would imply fixed parameter tractability for all problems in W[1]
- Theorem: There exists an algorithm that learns the class of circuits having n wires, alphabet size s, fan-in bound k, and shortcut width bounded by b, using $ns^{O(k+b)}$ value injection queries and time polynomial in the number of queries.
- Theorem: There exists a polynomial time algorithm that learns up to ε-equivalence any analog circuit of n wires, depth log(n), constant fan-in, Lipshitz gate functions, and shortcut width bounded by a constant.

Papers Covered in this Thesis

Learning Evolutionary Trees

Learning Graphs (for DNA sequencing)

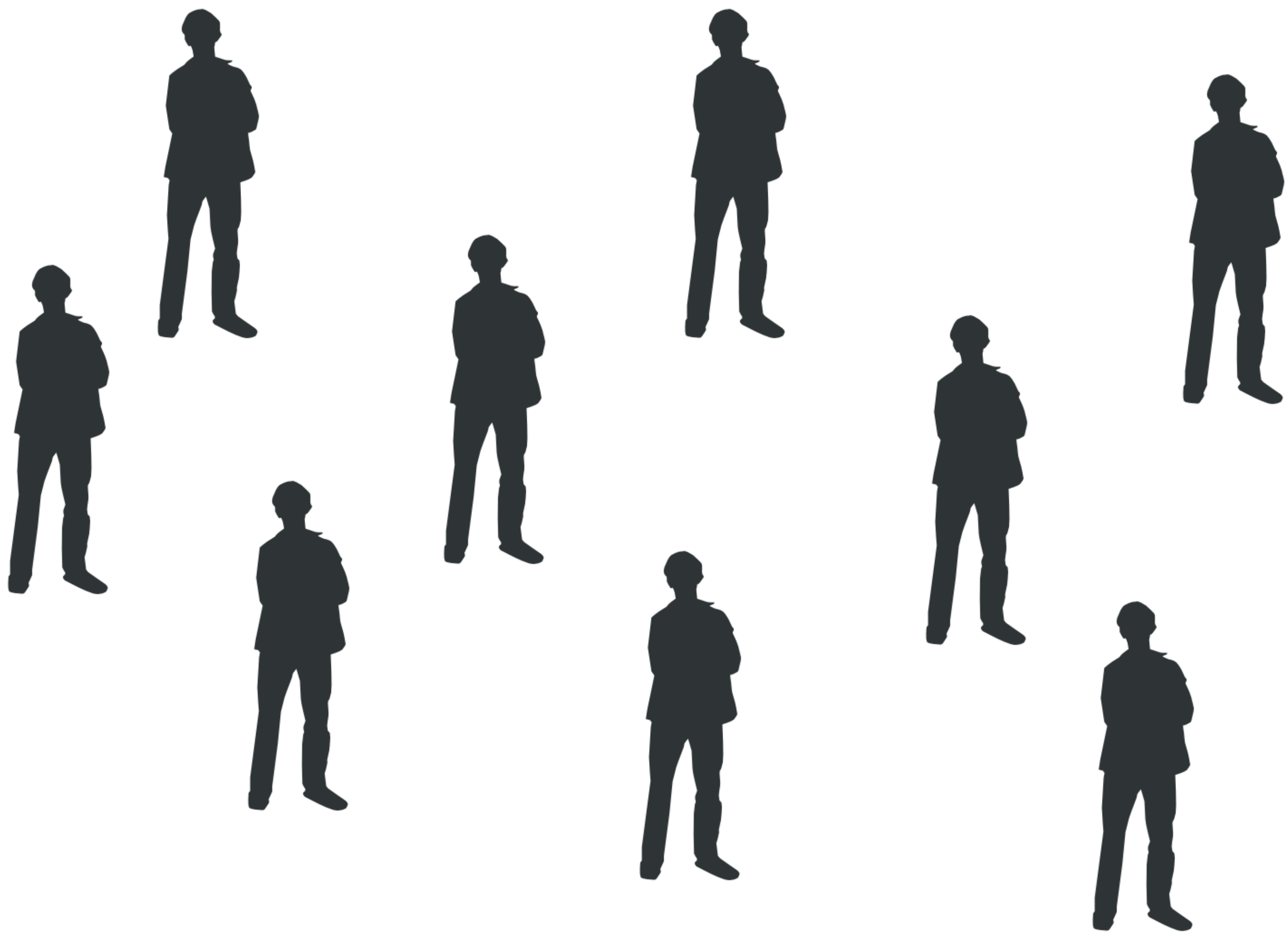
Learning Circuits (Gene Regulatory Networks)

Actively Learning Social Networks

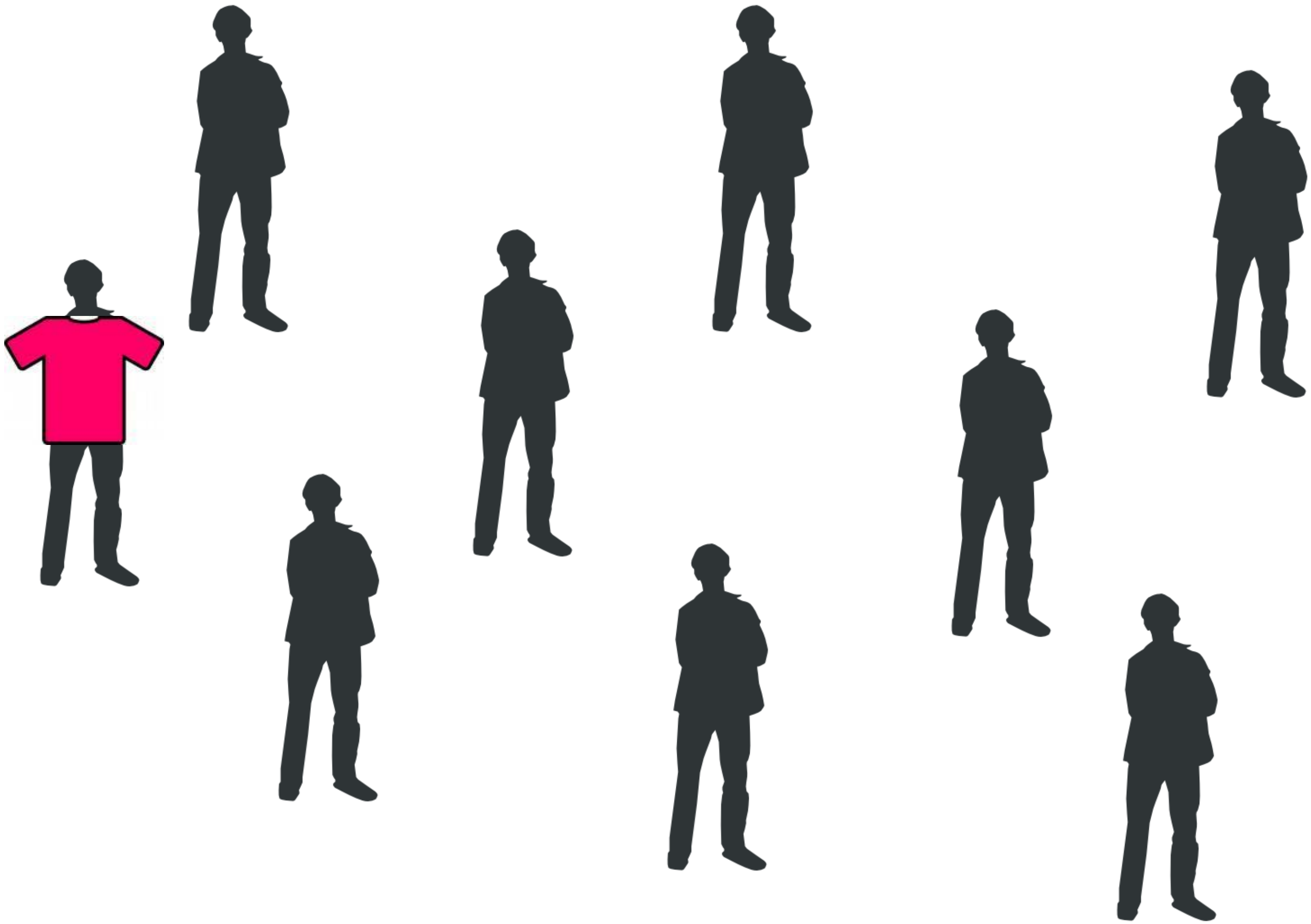
Passively Inferring Social Networks

Learning Finite State Automata

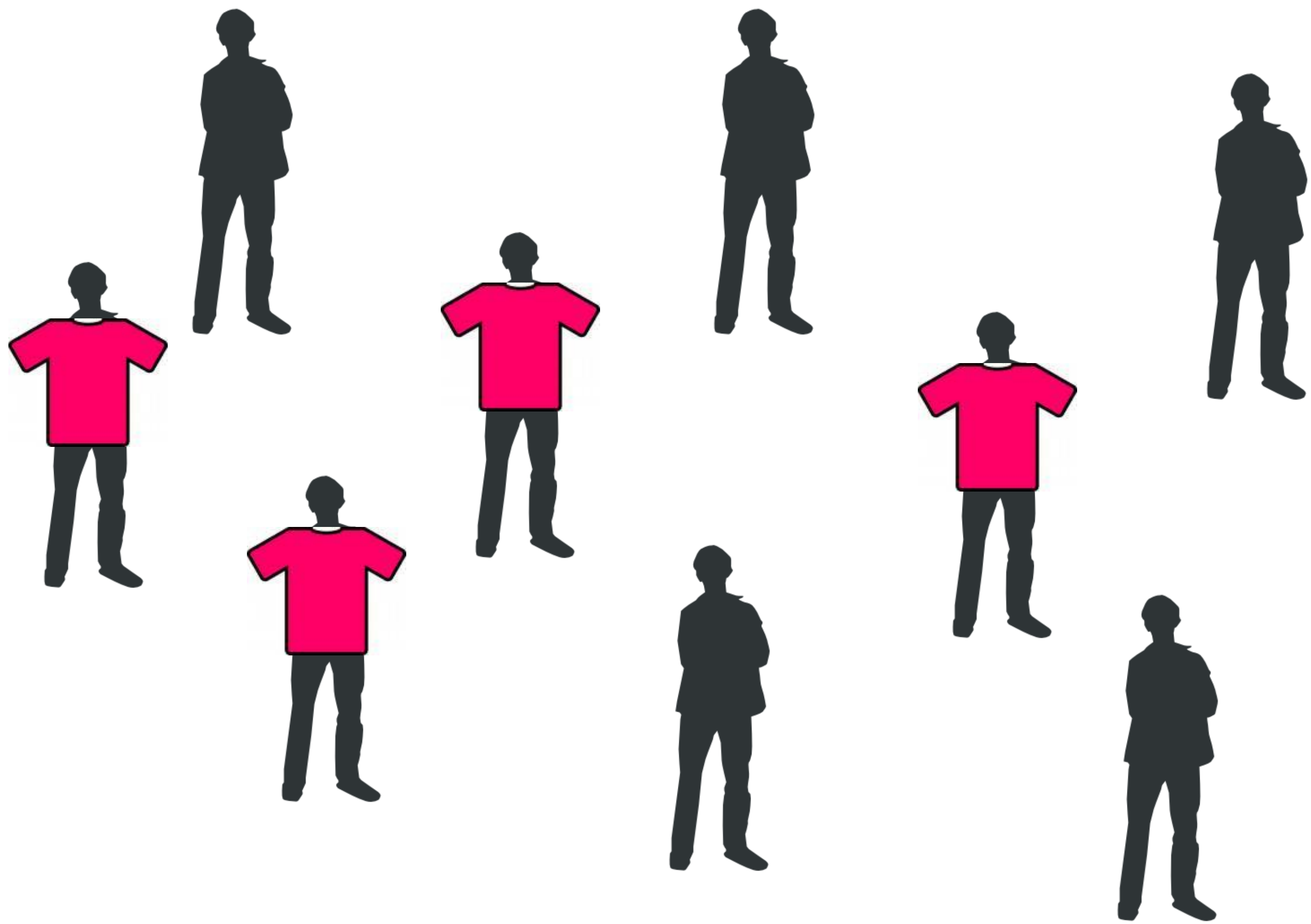
Trends Spreading through a Social Network



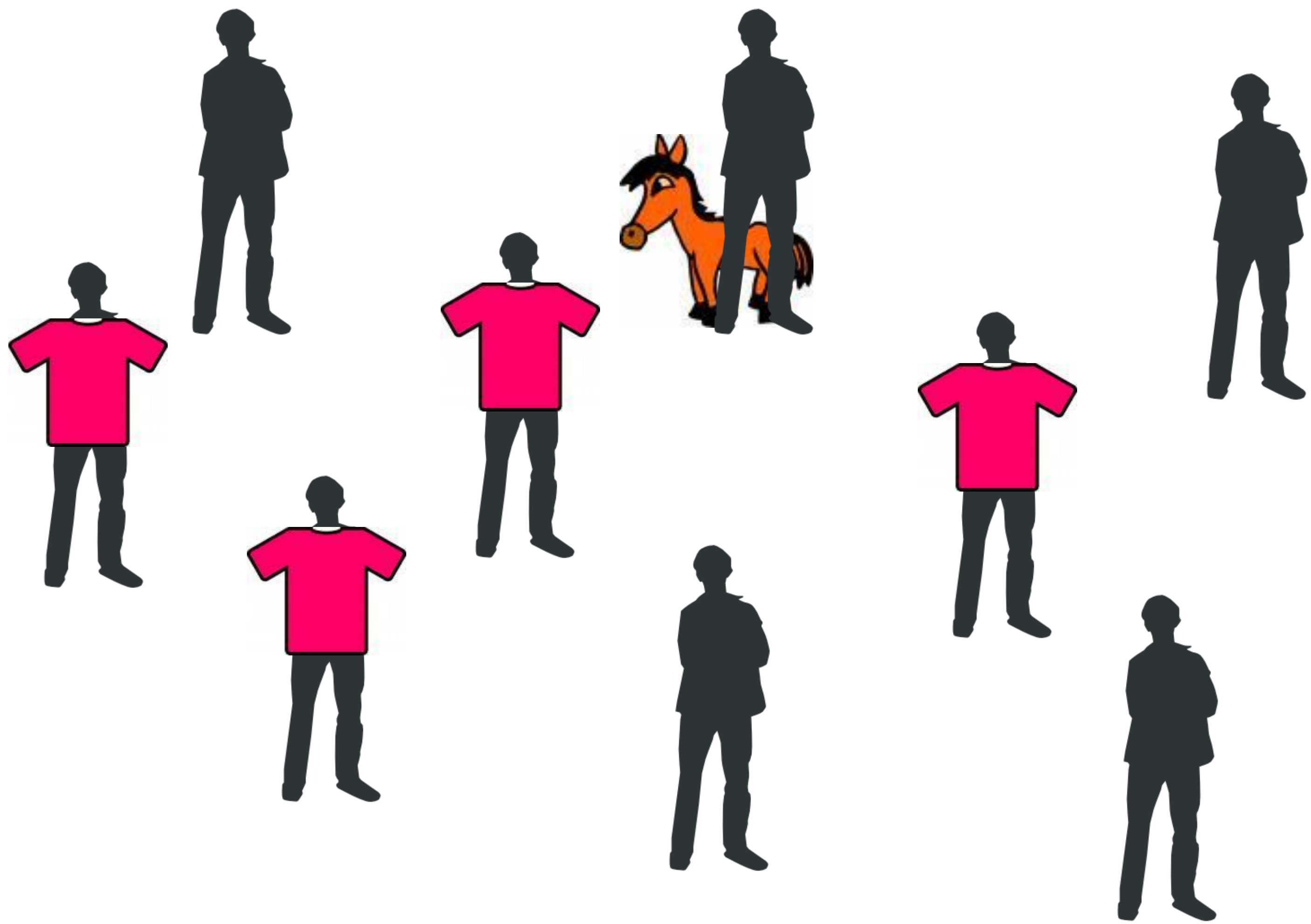
Trends Spreading through a Social Network



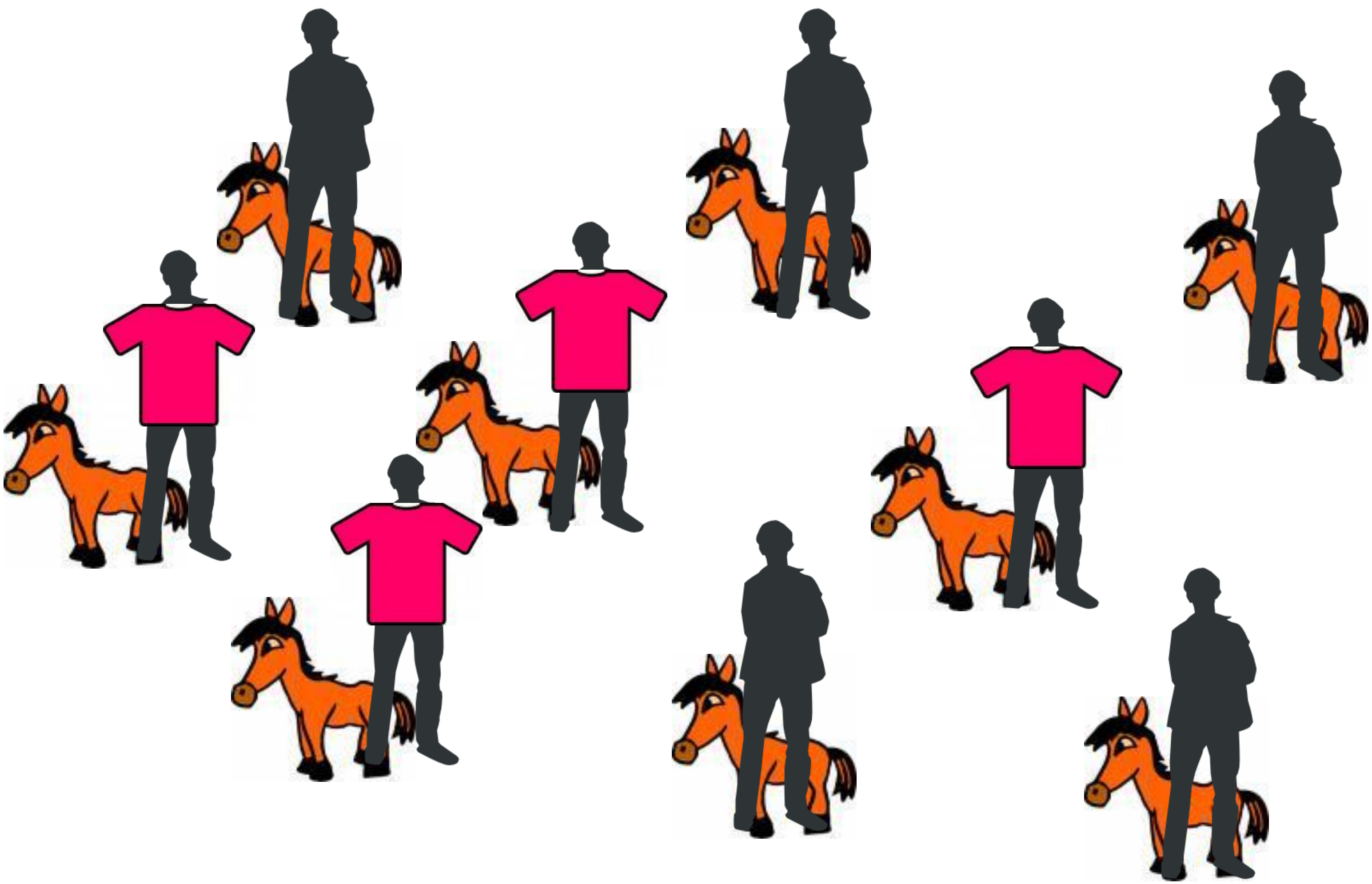
Trends Spreading through a Social Network



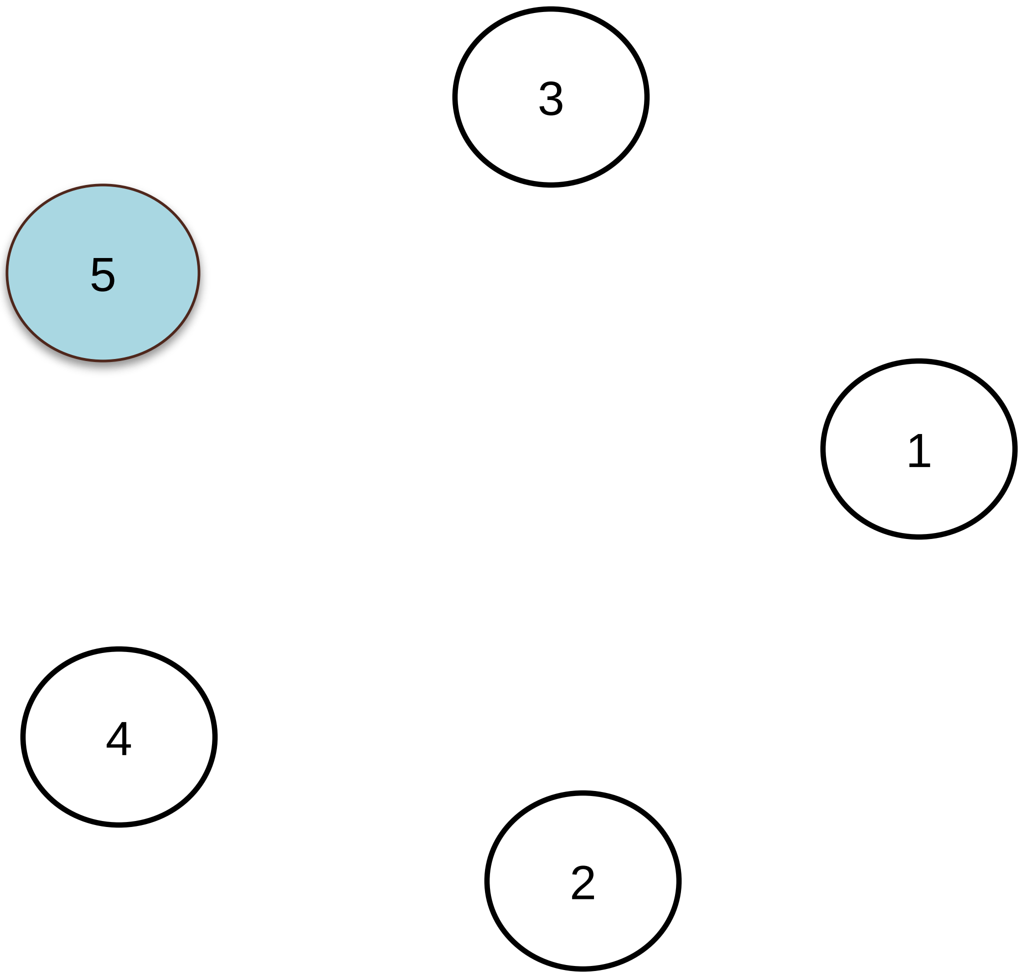
Trends Spreading through a Social Network



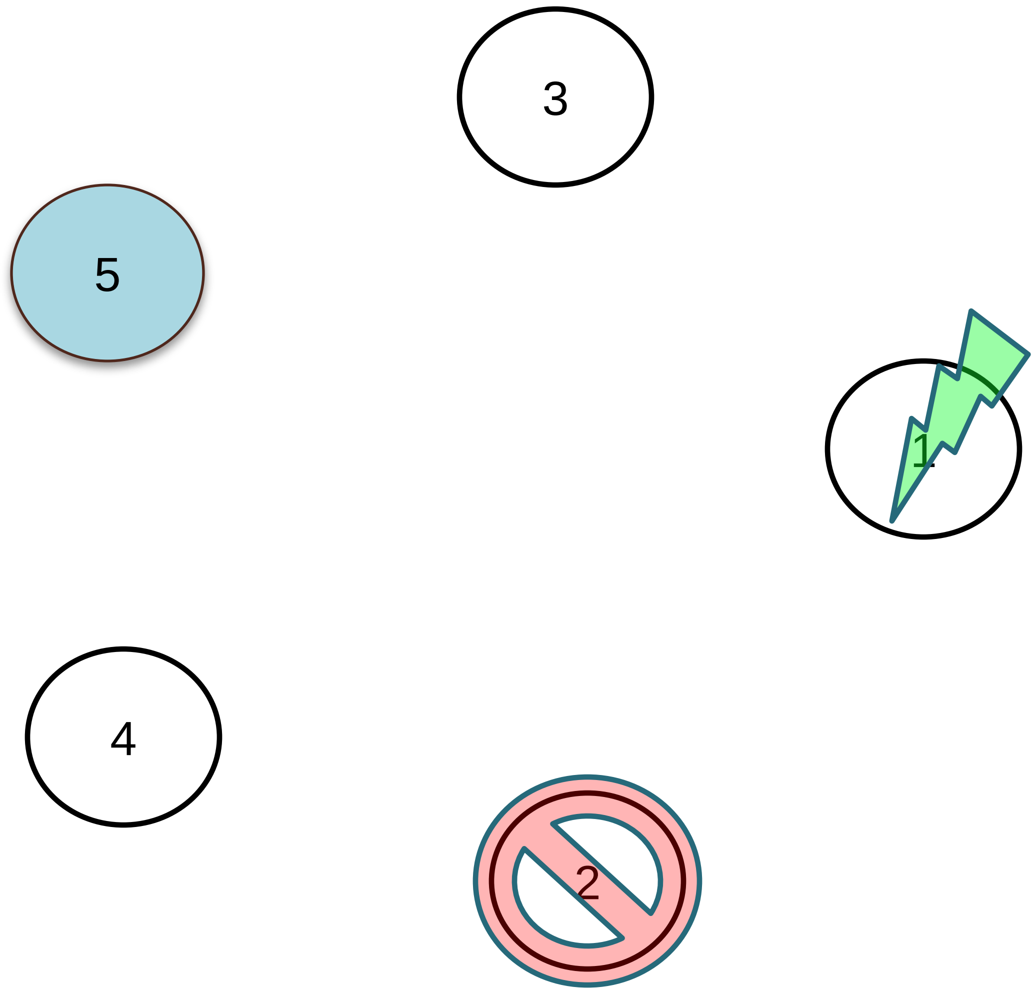
Trends Spreading through a Social Network



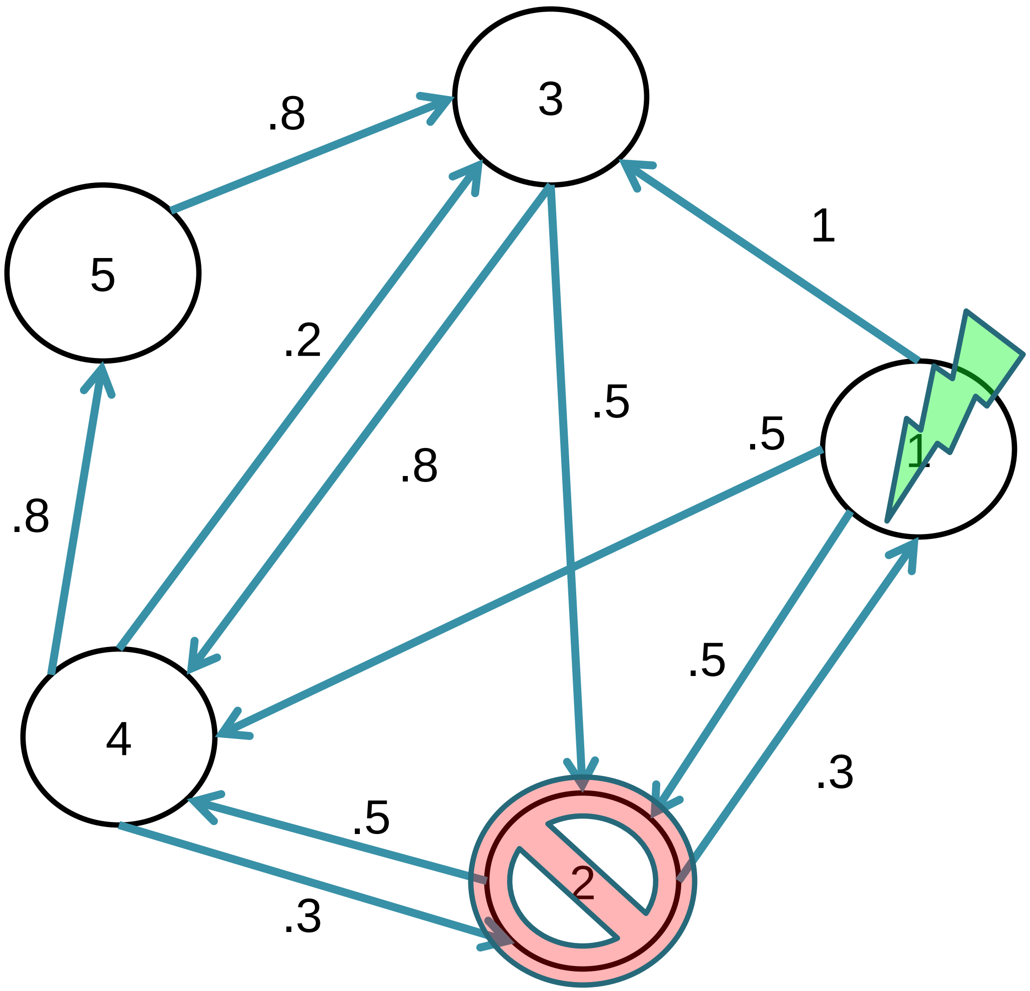
What the Learner Sees



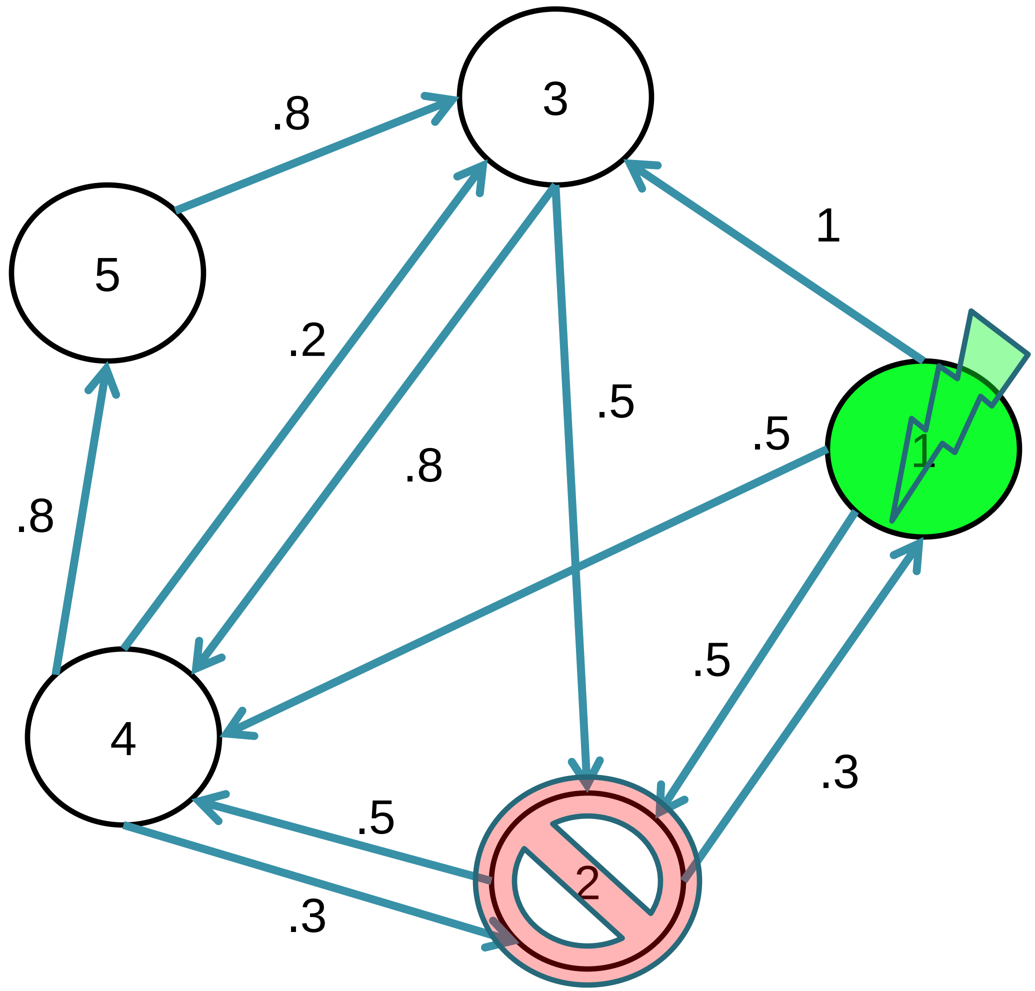
Activations and Suppressions



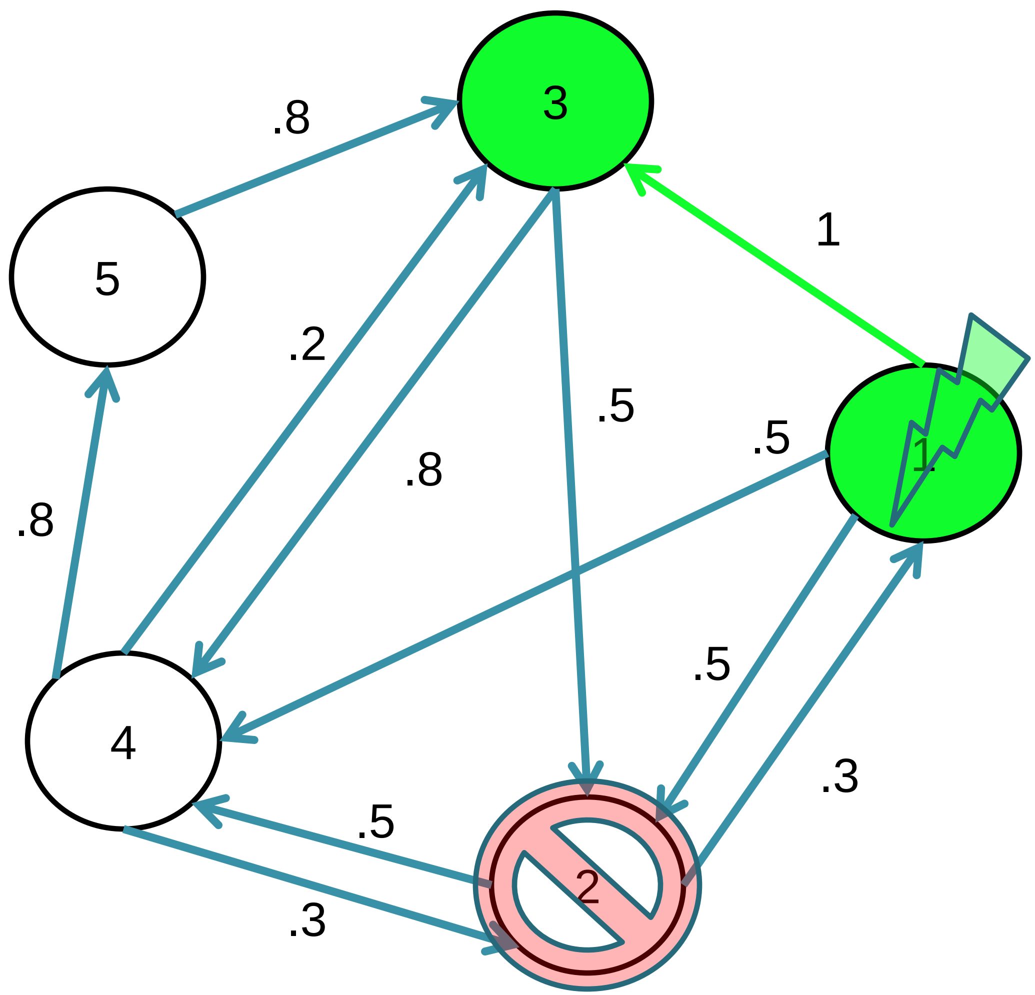
Activations and Suppressions



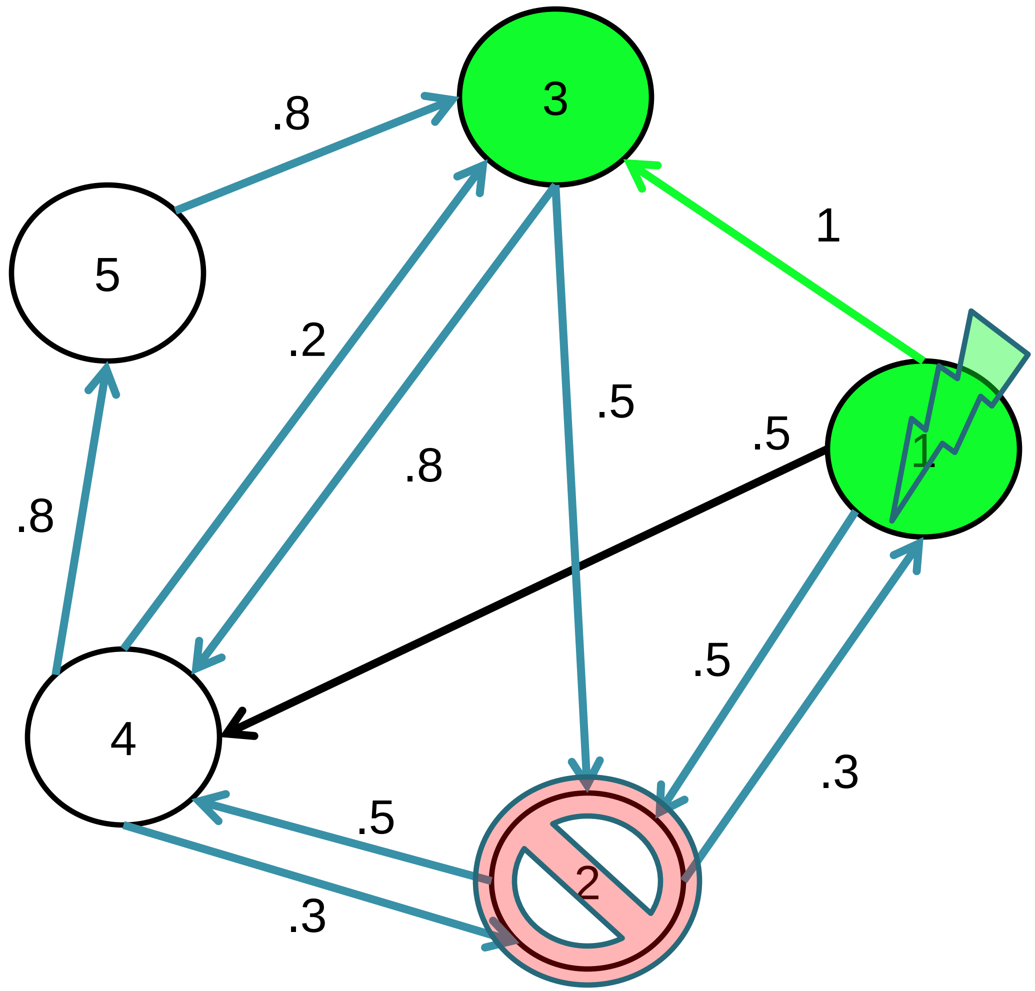
Activations and Suppressions



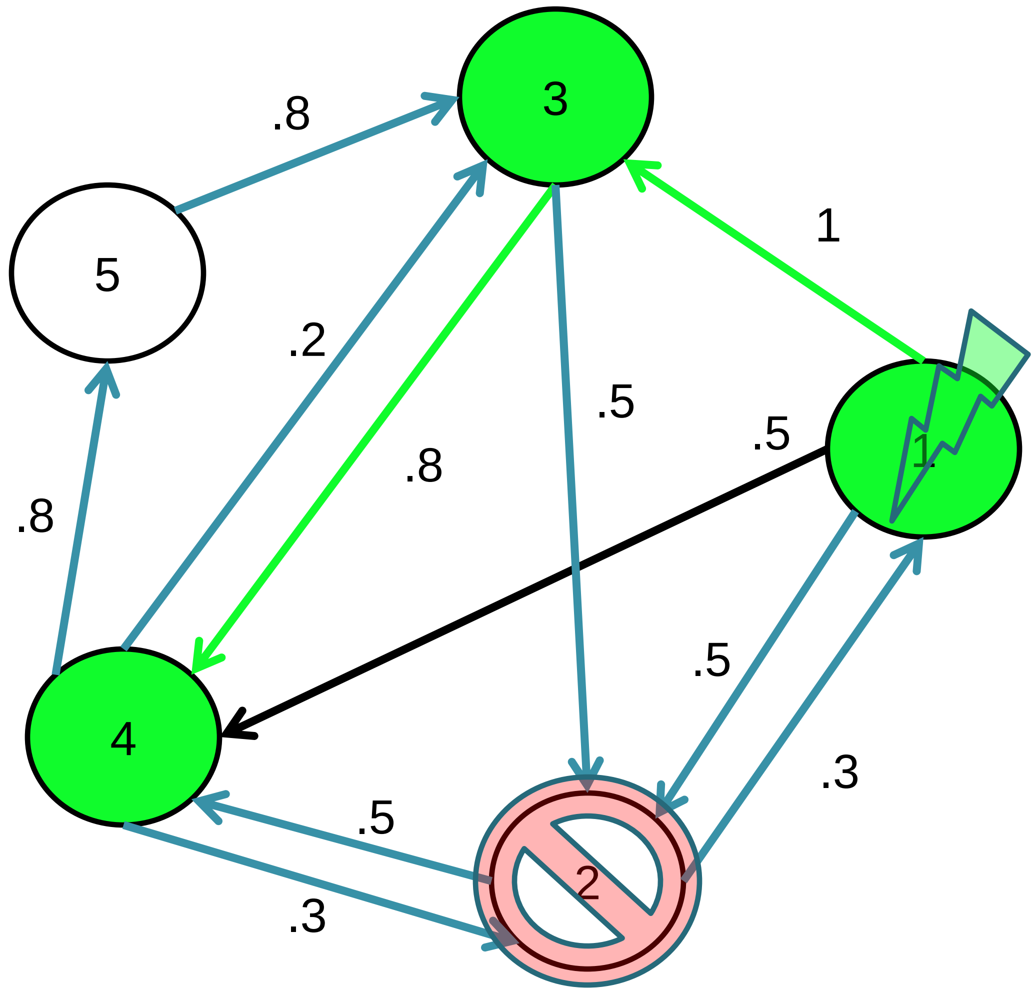
Activations and Suppressions



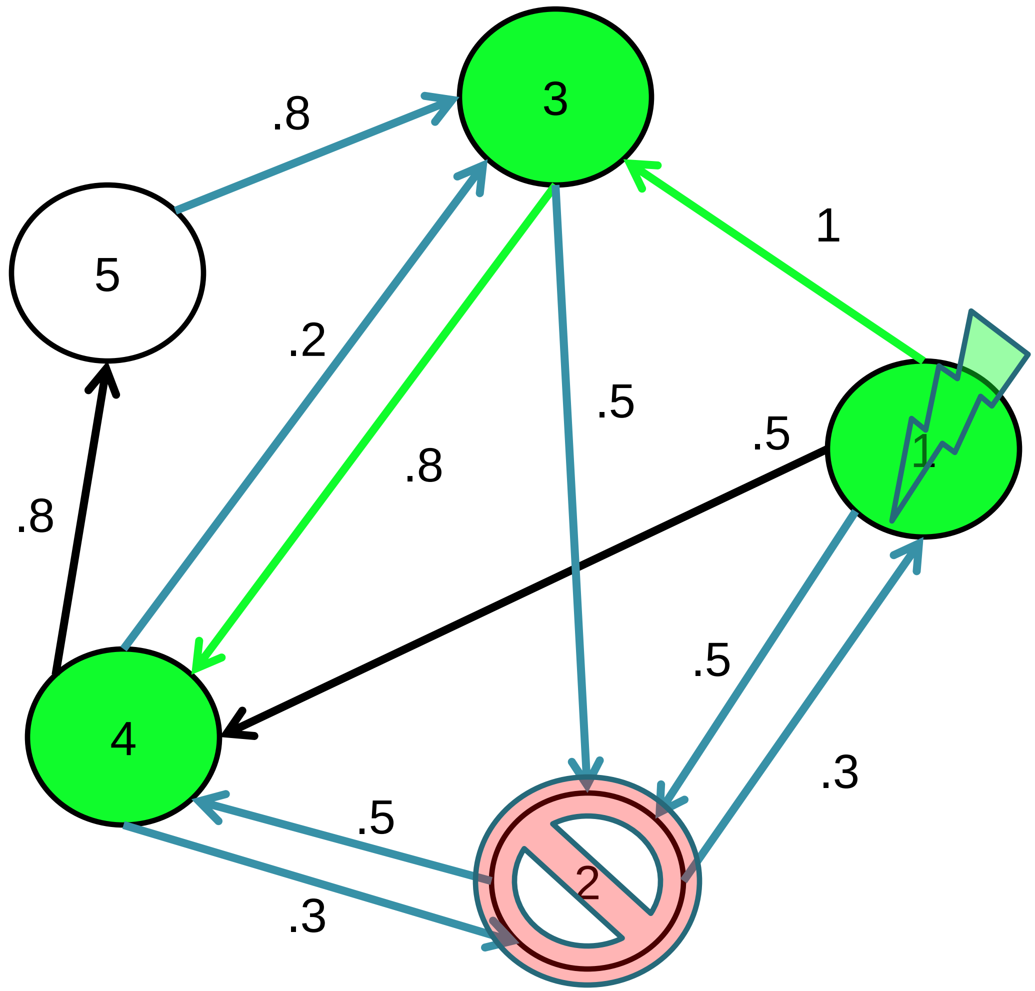
Activations and Suppressions



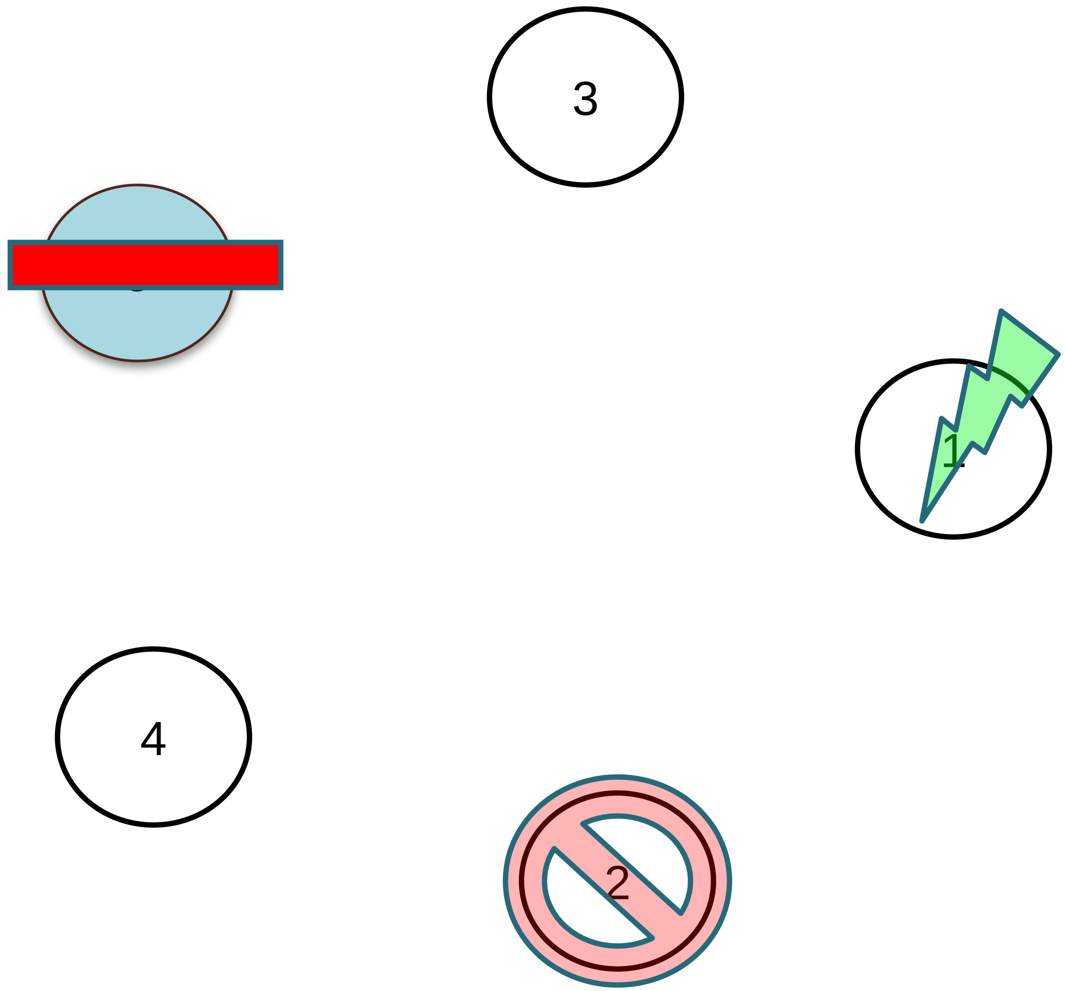
Activations and Suppressions



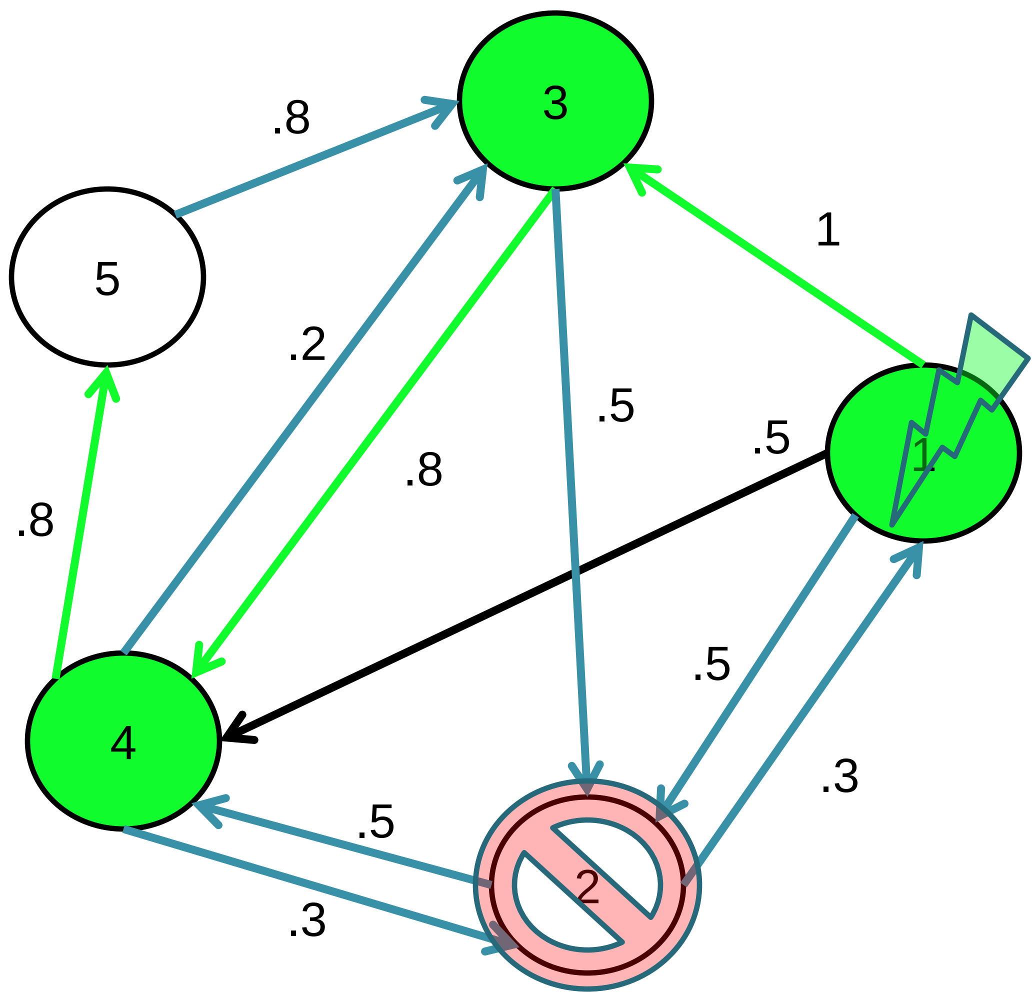
Activations and Suppressions



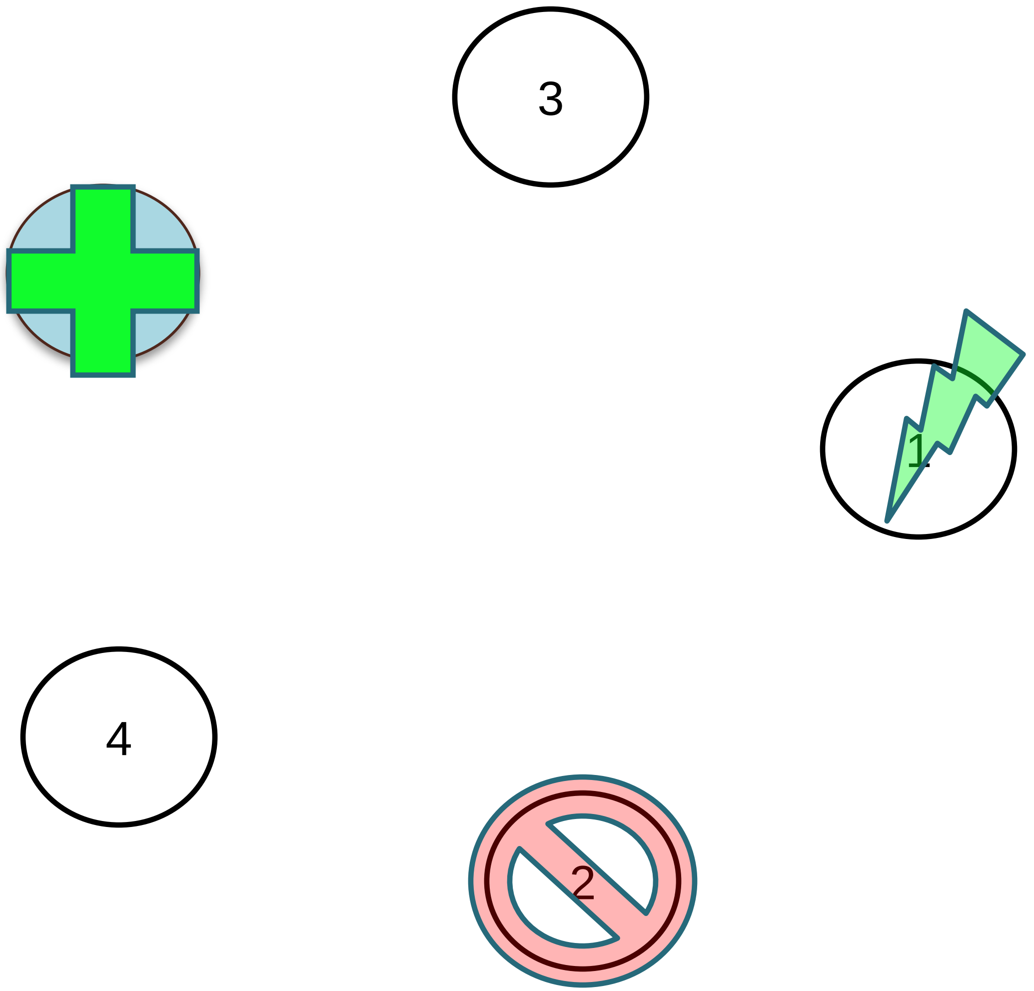
Activations and Suppressions



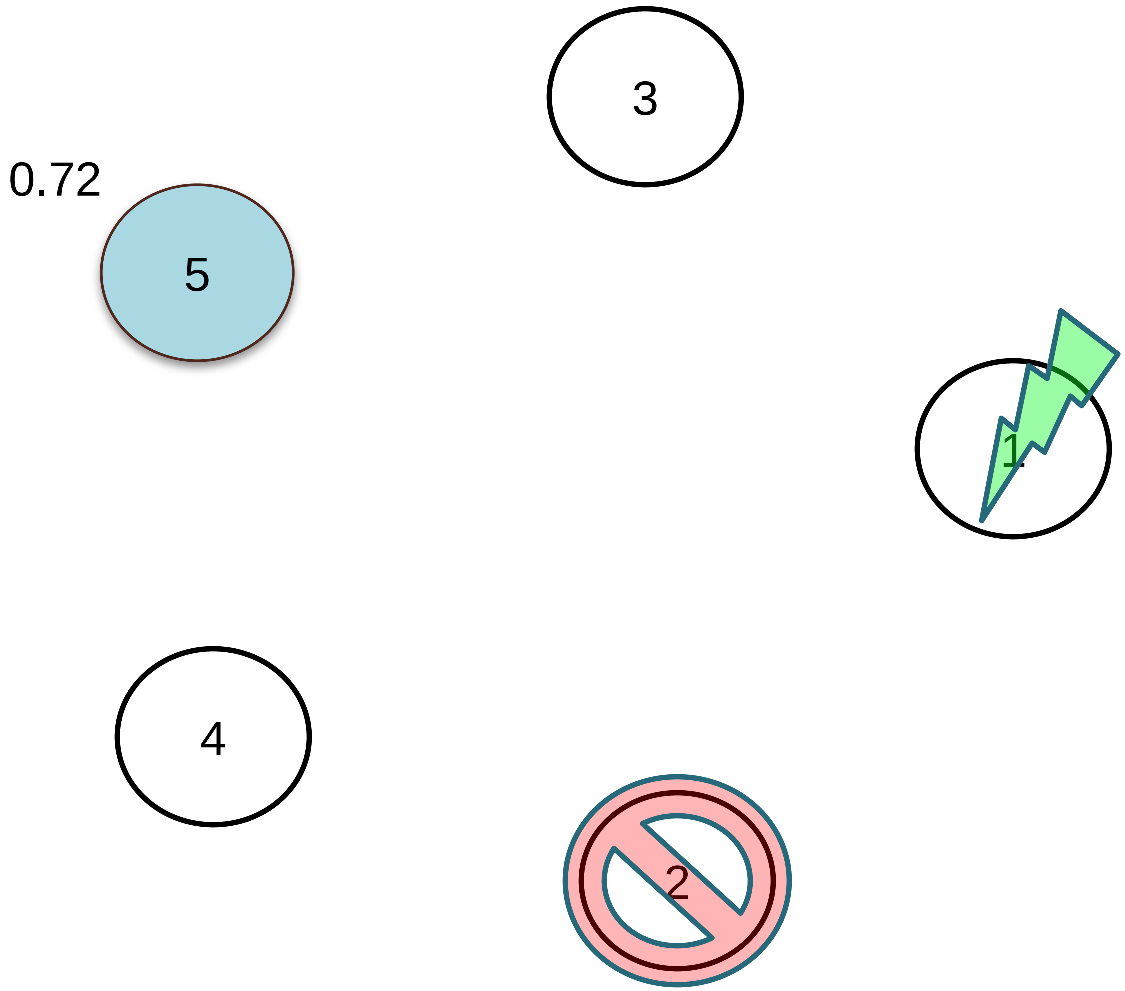
Activations and Suppressions



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)

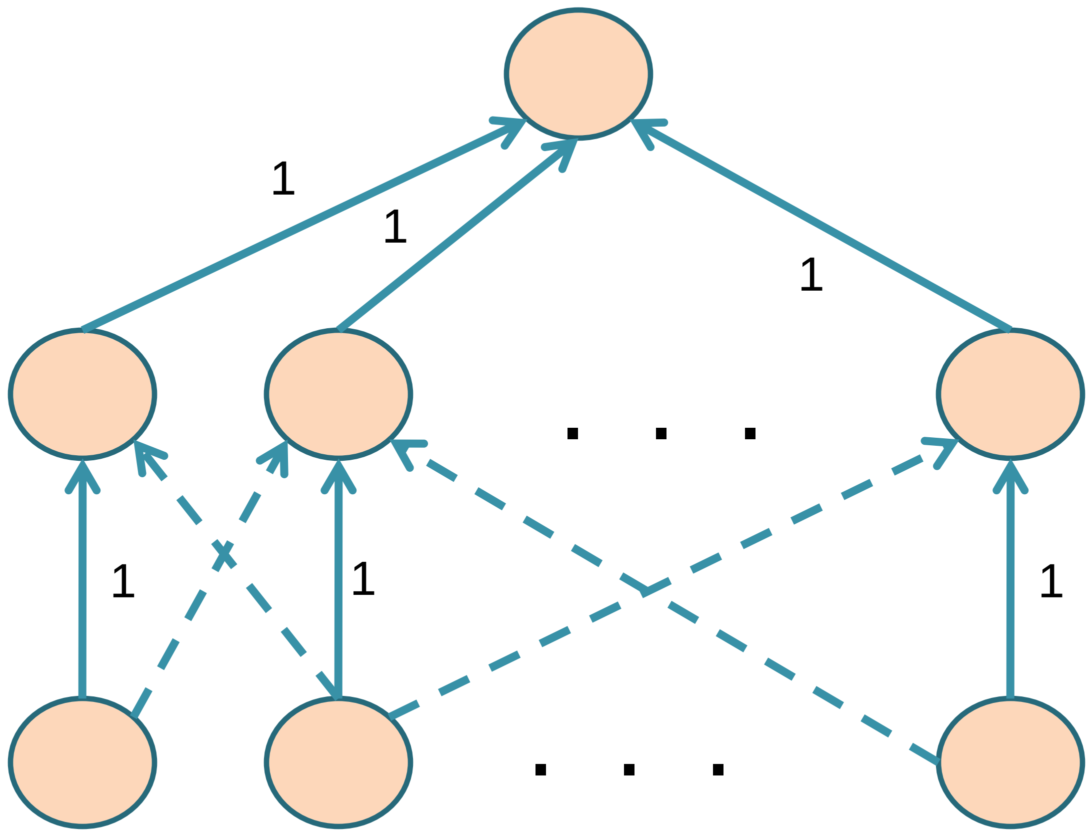
- Given access to a hidden social network S*, the learning problem is to find a social network S behaviorally equivalent to S* using value injection queries.

The Percolation Model

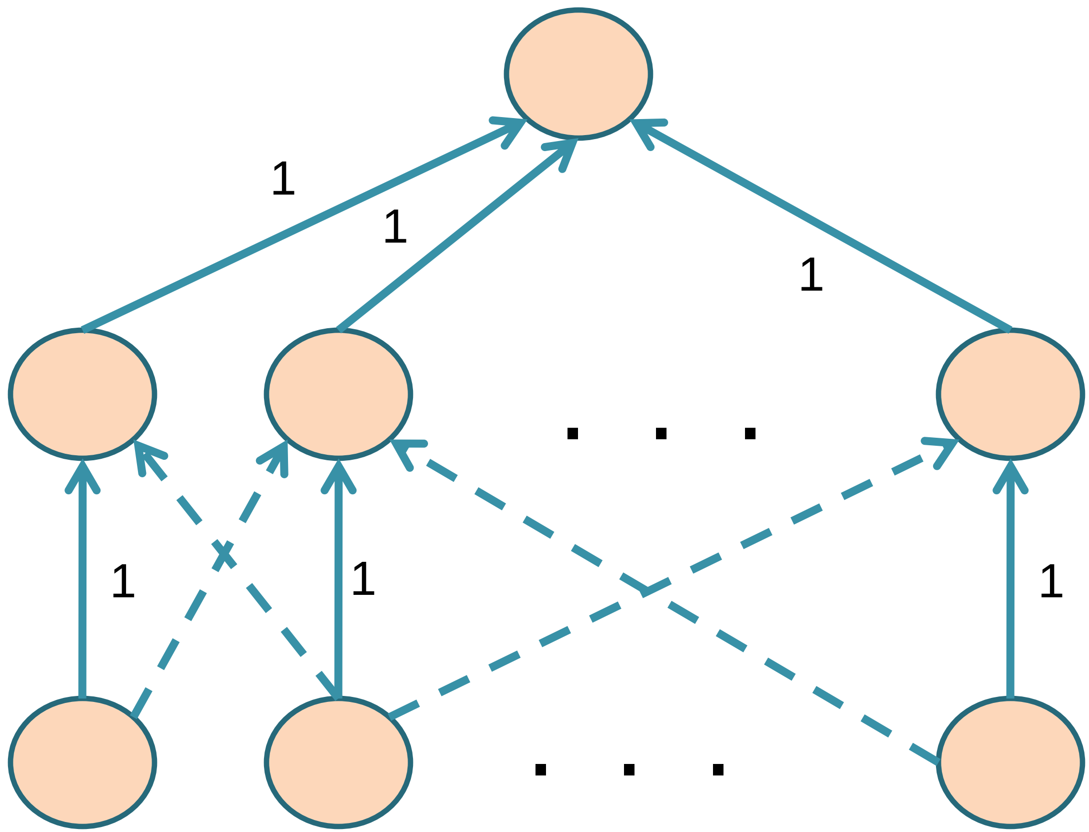
Given a network S and a VIQ

- 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

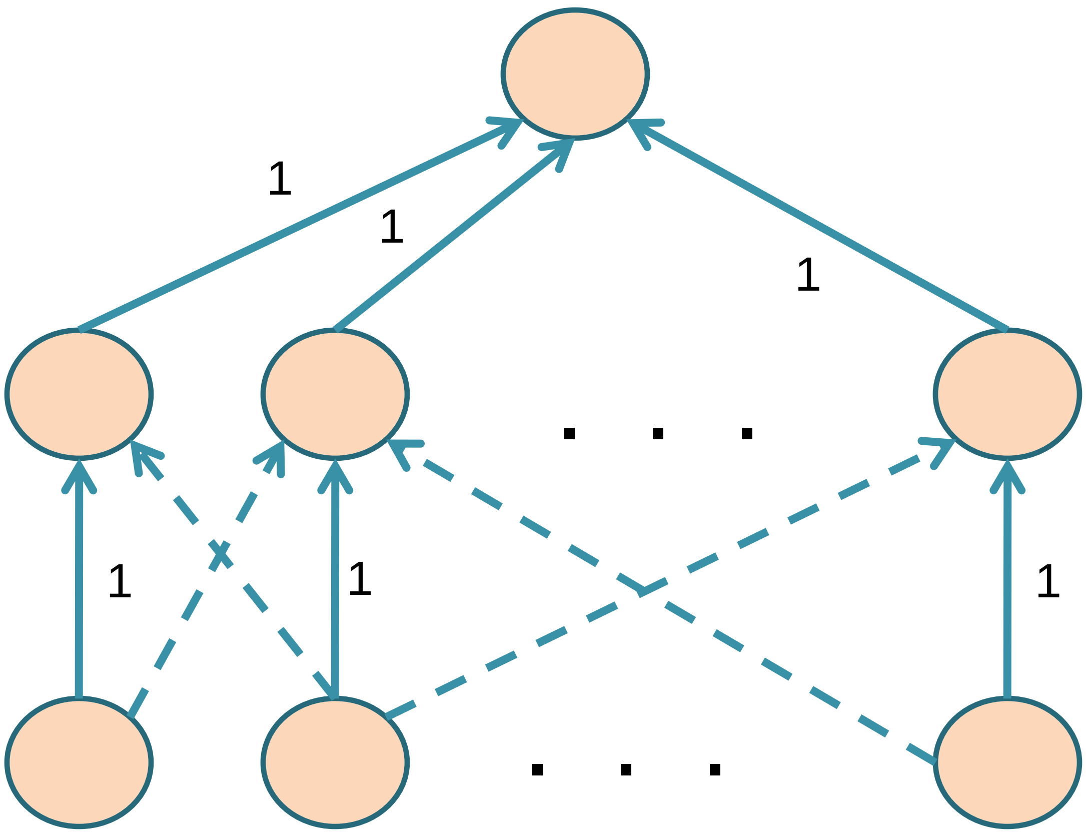


A Lower Bound



All queries give 1-bit answers

A Lower Bound



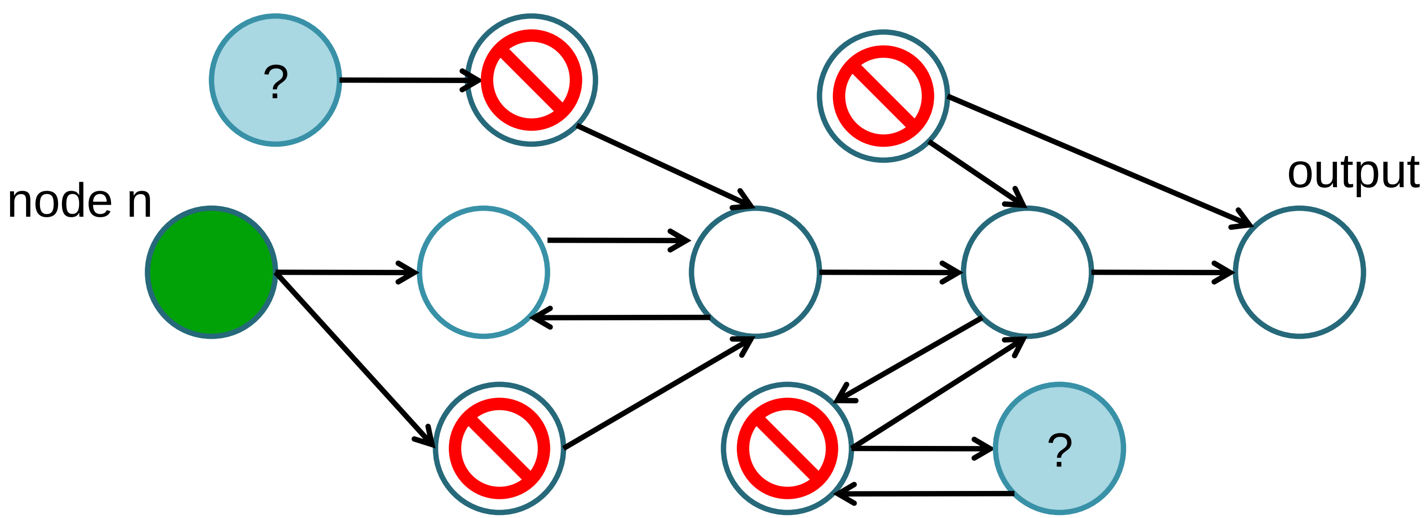
$2^{\Omega(n^2)}$ such graphs, $\Omega(n^2)$ l.b.

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

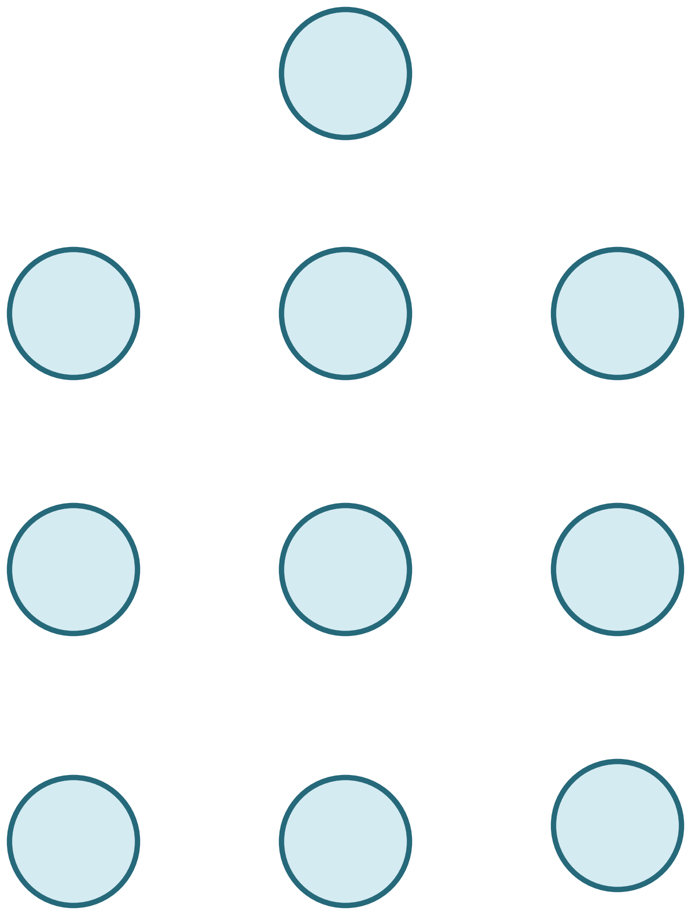
- An excitation path for a node n is a VIQ in which a subset of the free agents form a simple directed path from n to the output. All agents not on the path with inputs into the path are suppressed.
- We also have a shortest excitation path



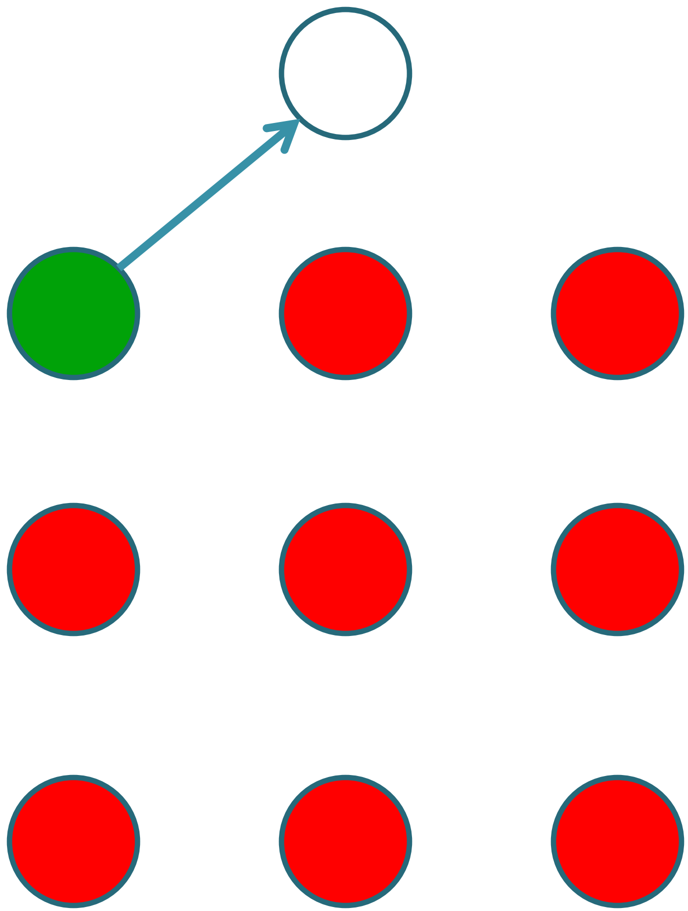
The Learning Algorithm For Networks Without 1 Edges

- 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

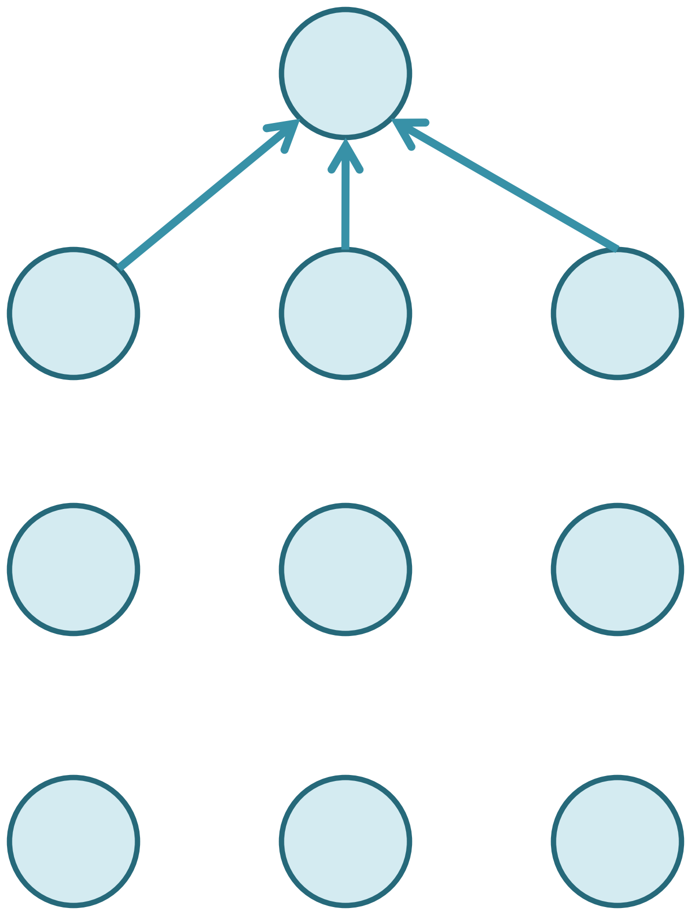
Find-Up-Edges



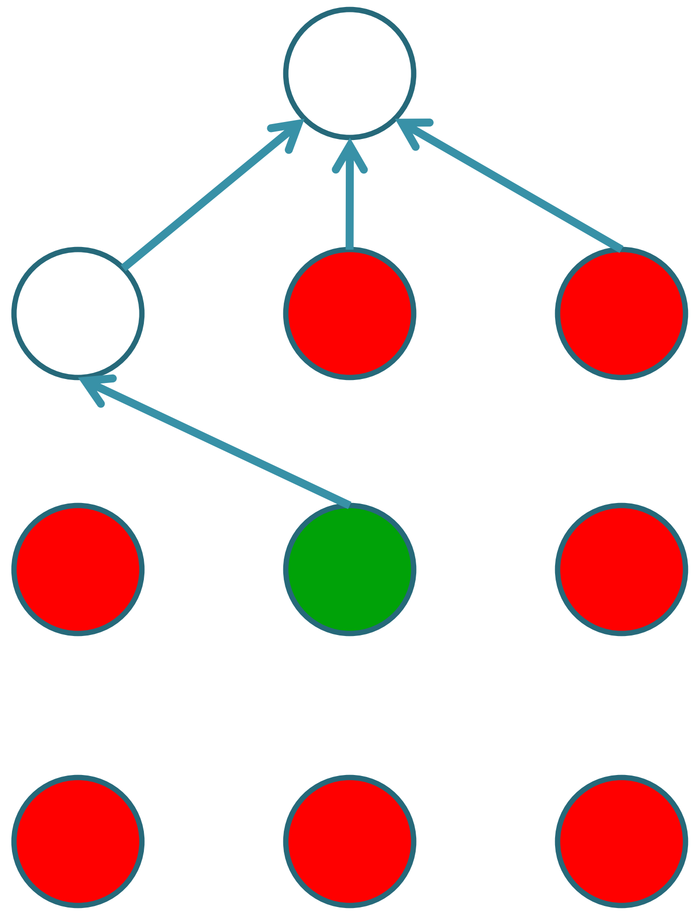
Find-Up-Edges



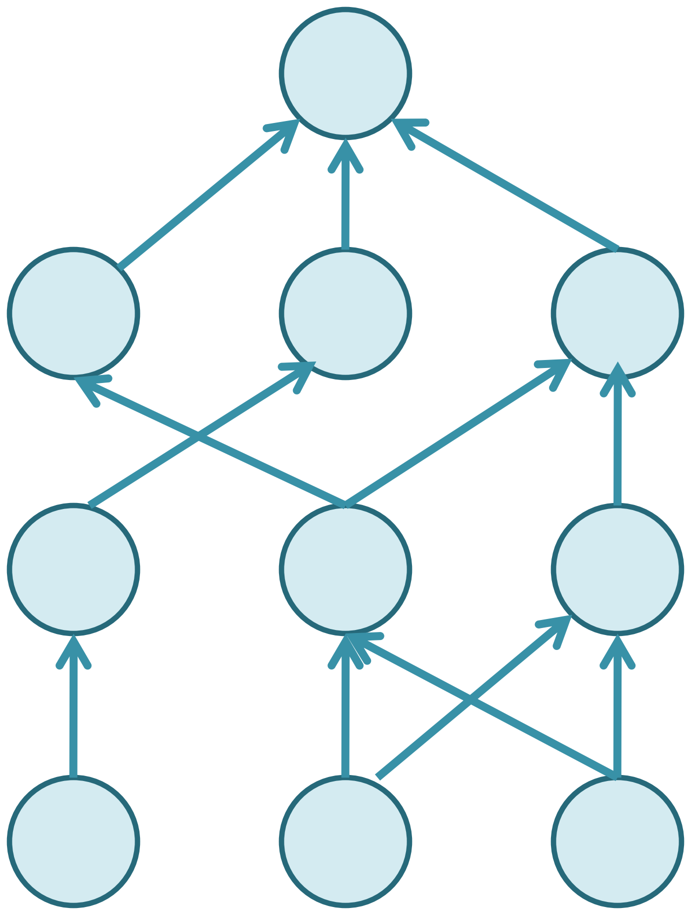
Find-Up-Edges



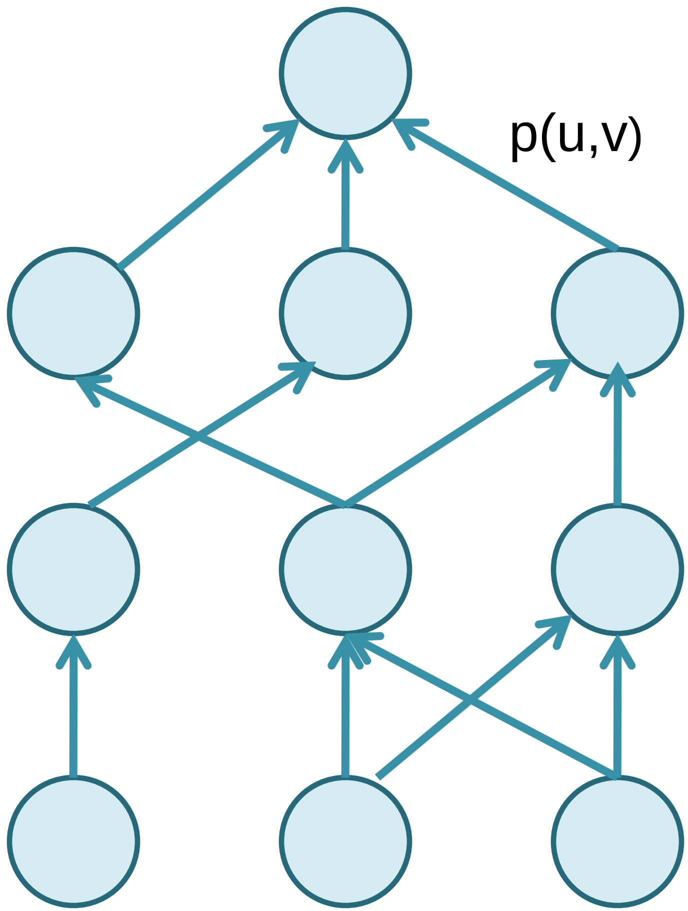
Find-Up-Edges



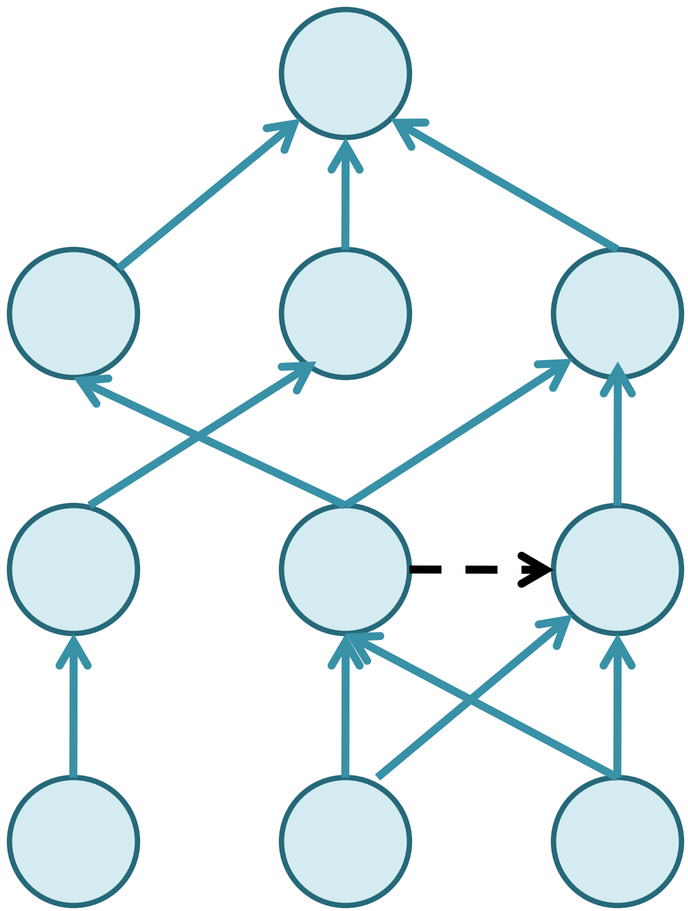
Find-Up-Edges



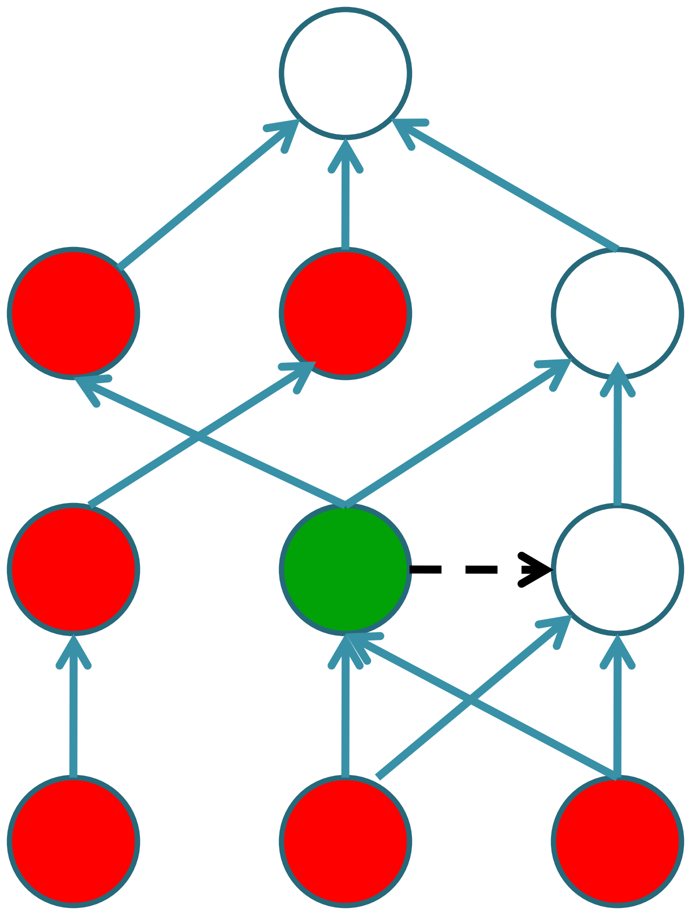
Find-Up-Edges



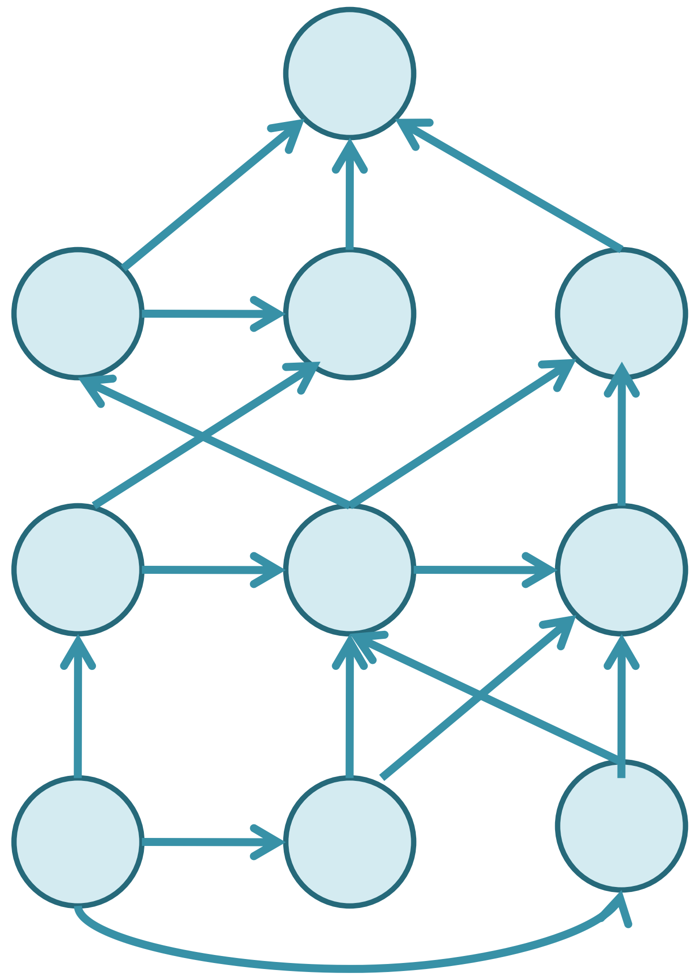
Find-Level-Edges



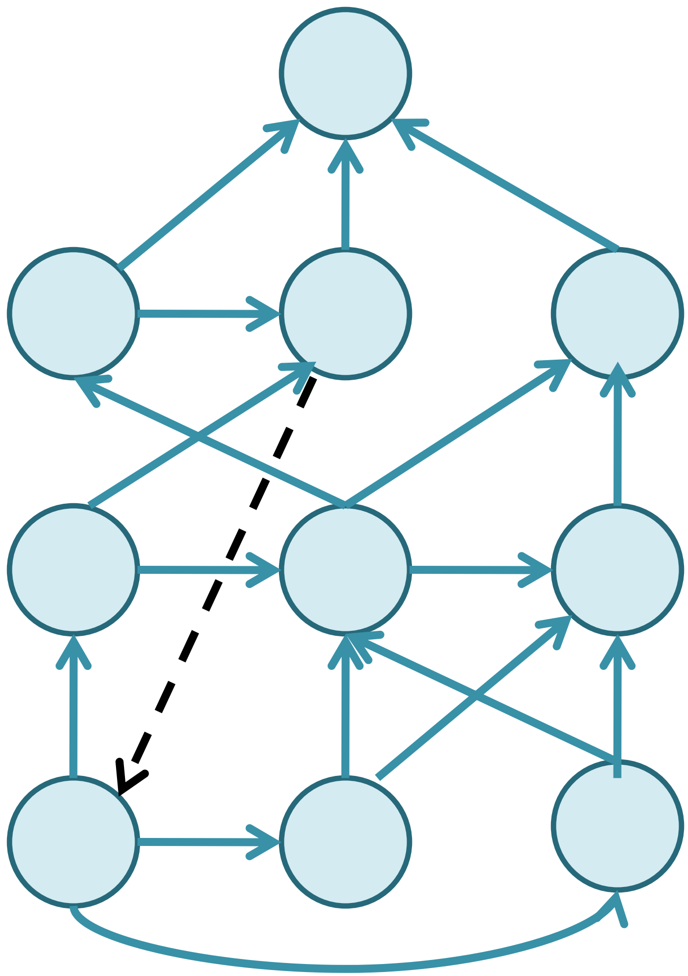
Find-Level-Edges



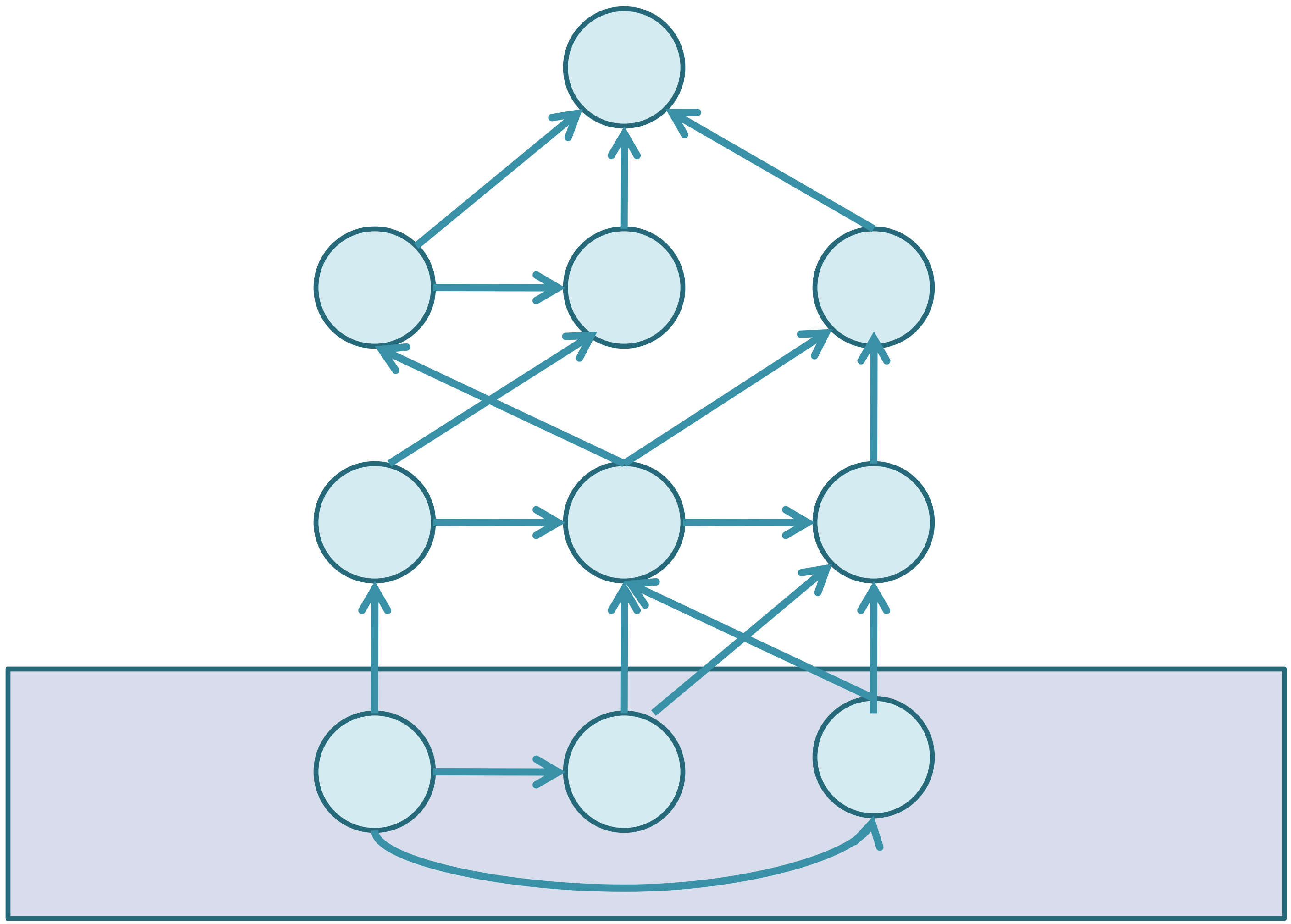
Find-Level-Edges



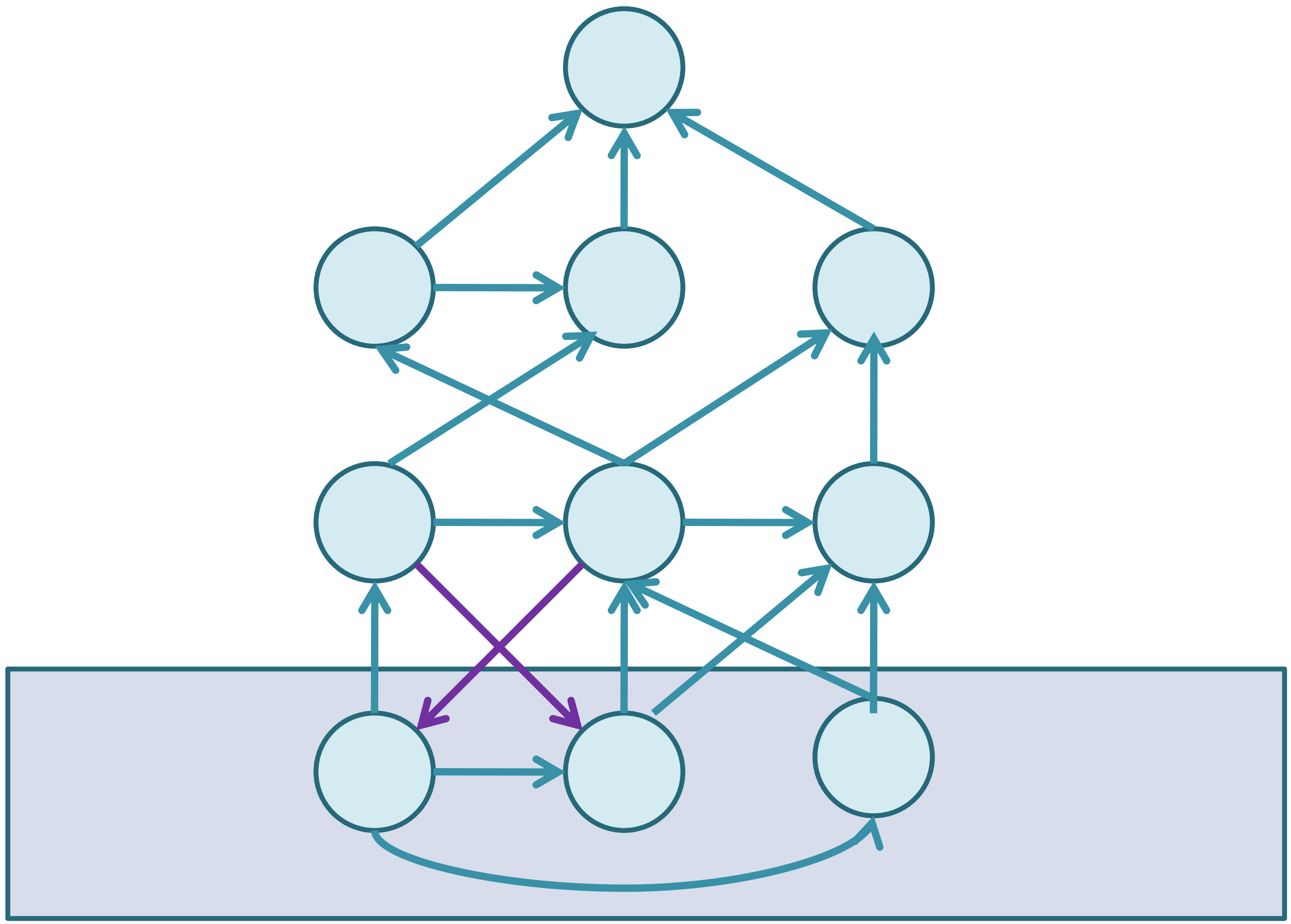
Find-Down-Edges



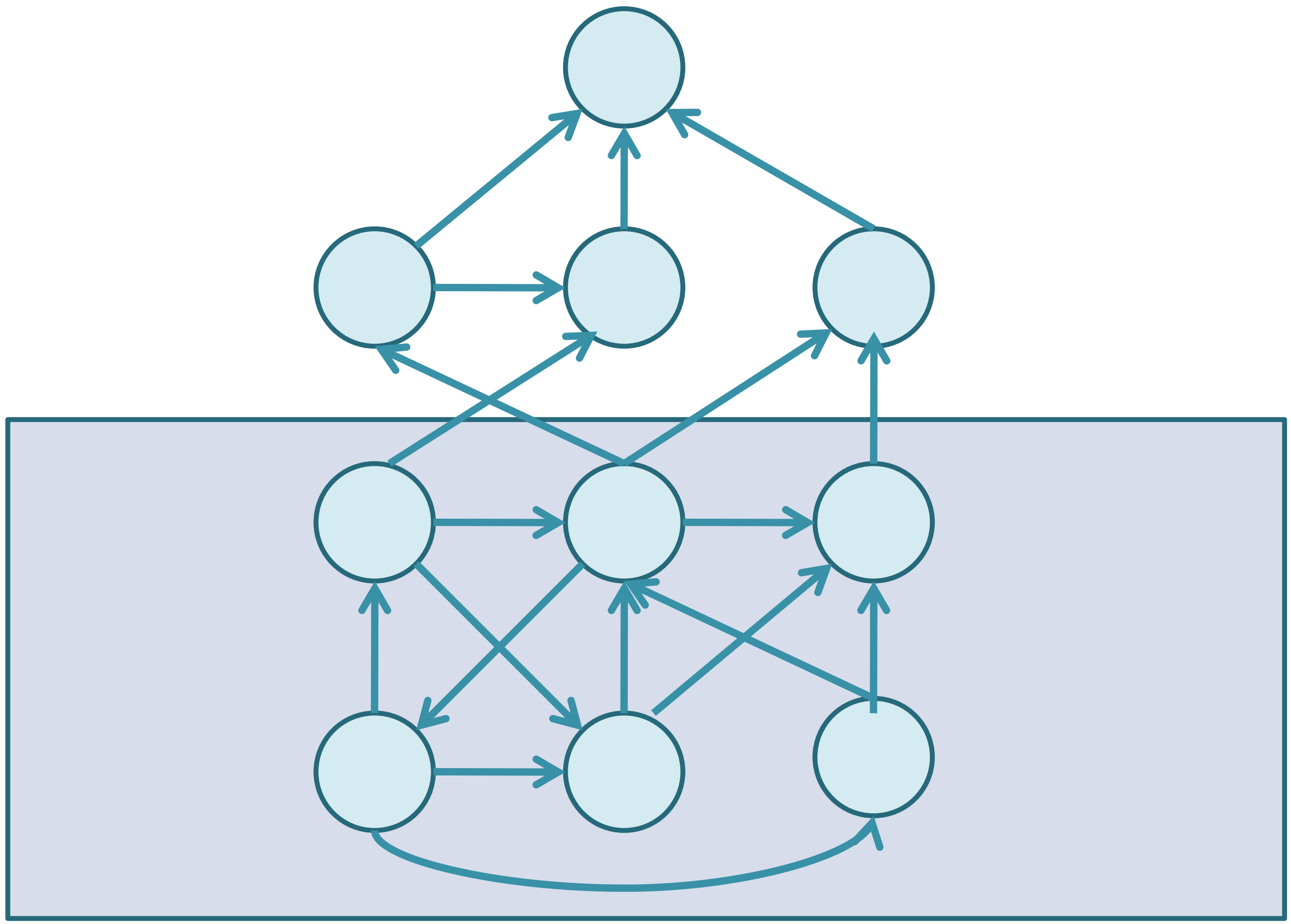
Find-Down-Edges



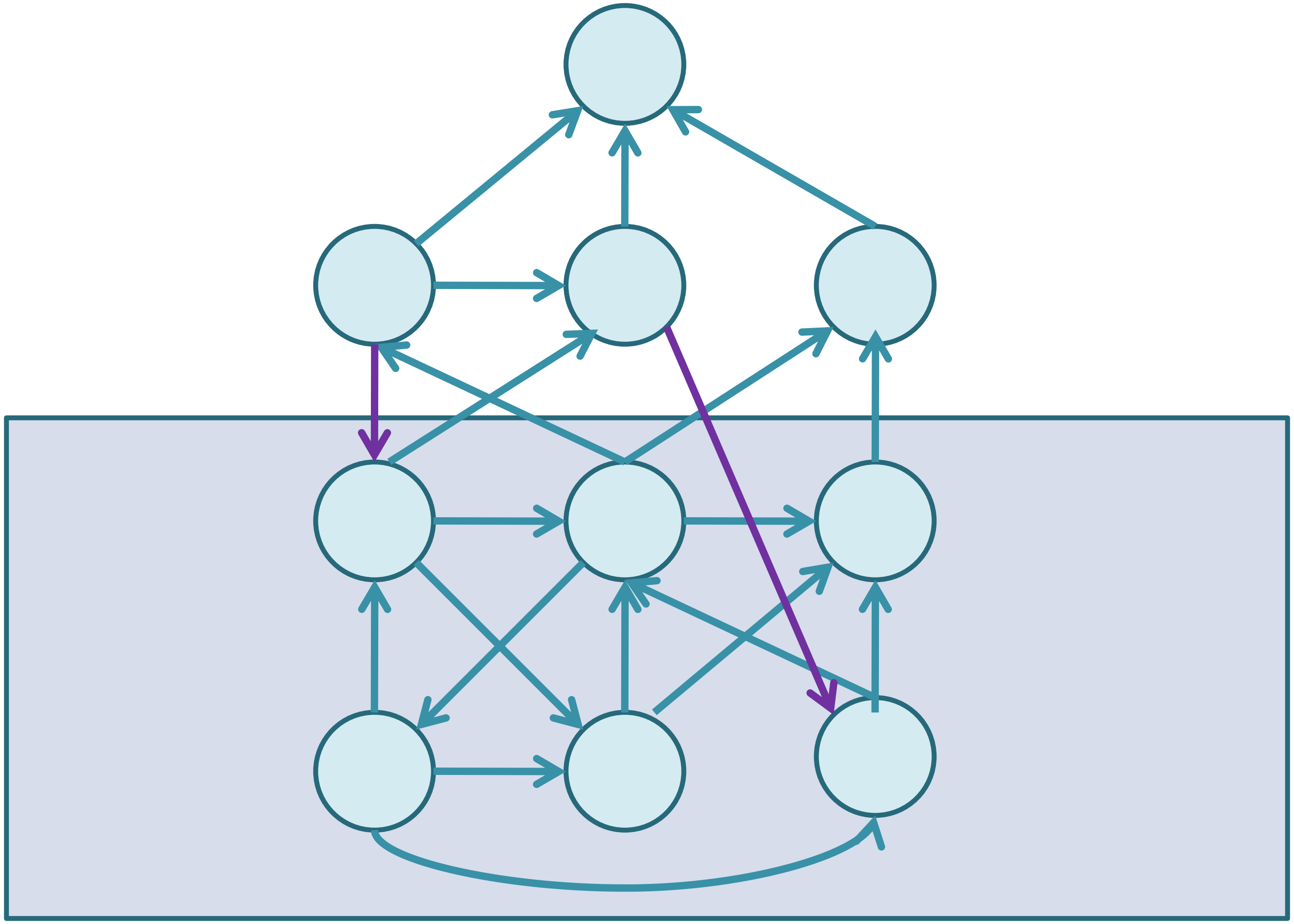
Find-Down-Edges



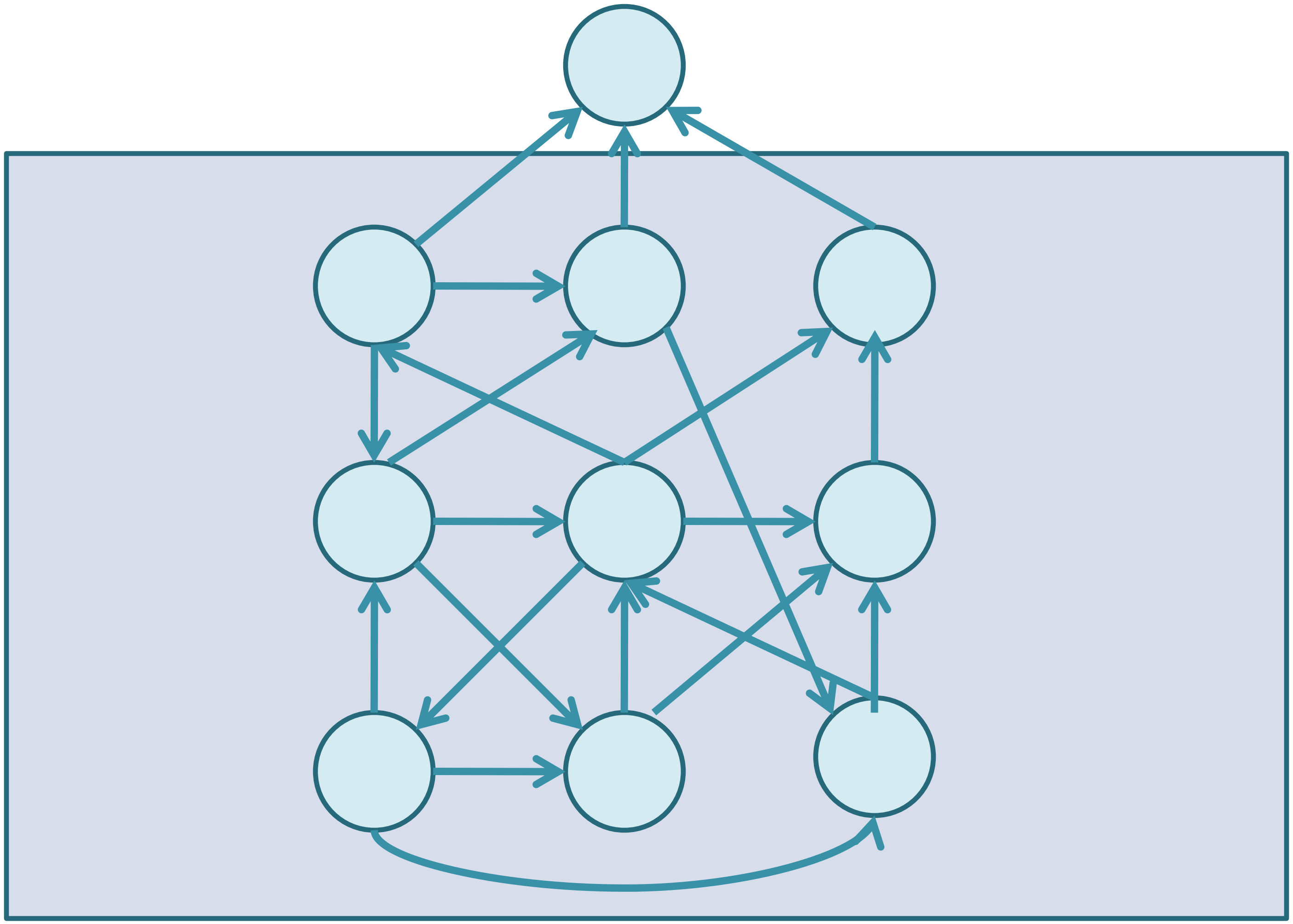
Find-Down-Edges



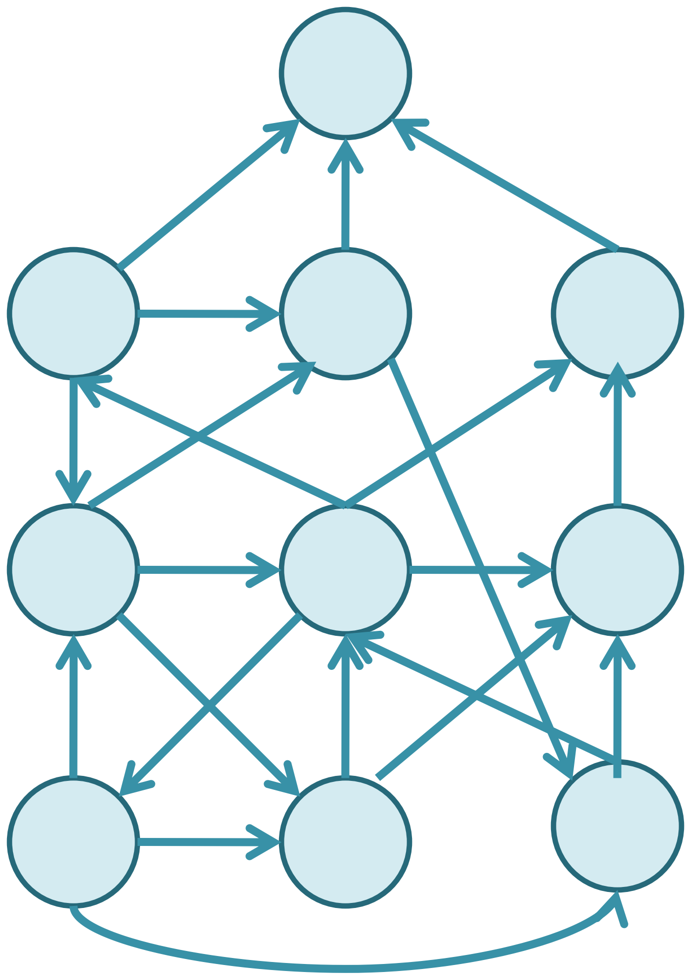
Find-Down-Edges



Find-Down-Edges



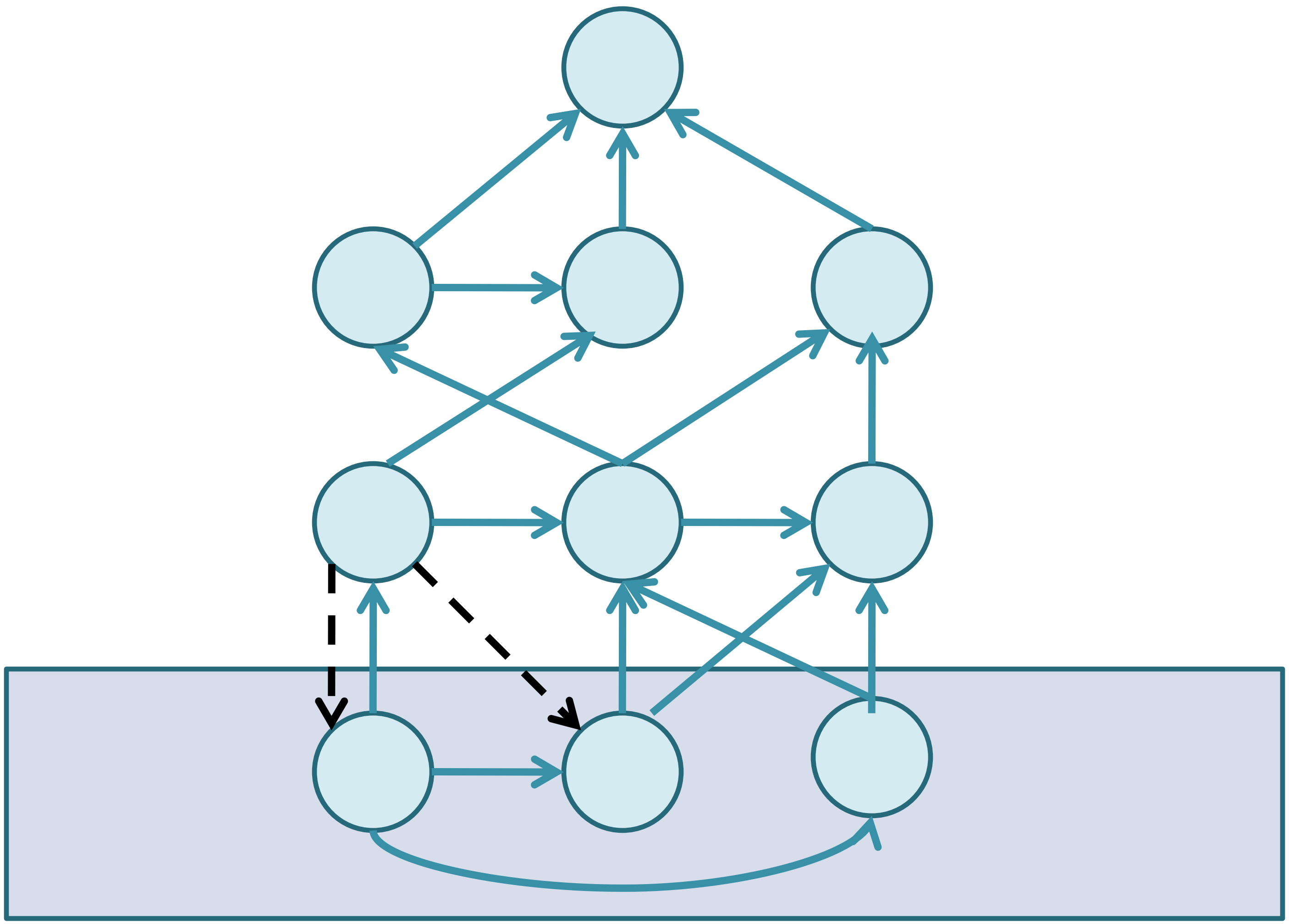
Find-Down-Edges



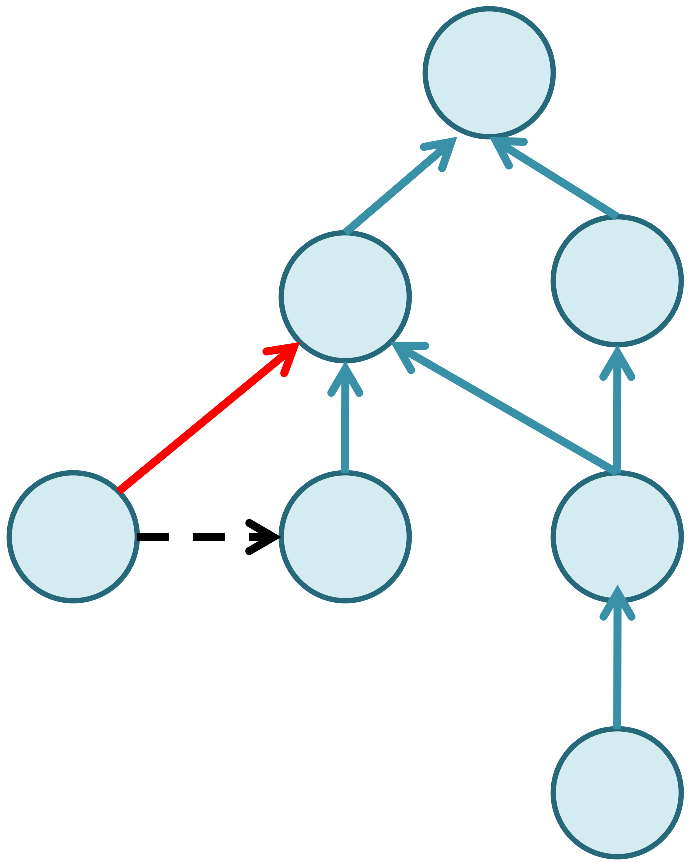
Find-Down-Edges

- 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 $v_1 \ldots v_k$ be the sorted v_is
  - Let $pi_1 \ldots pi_k$ 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.

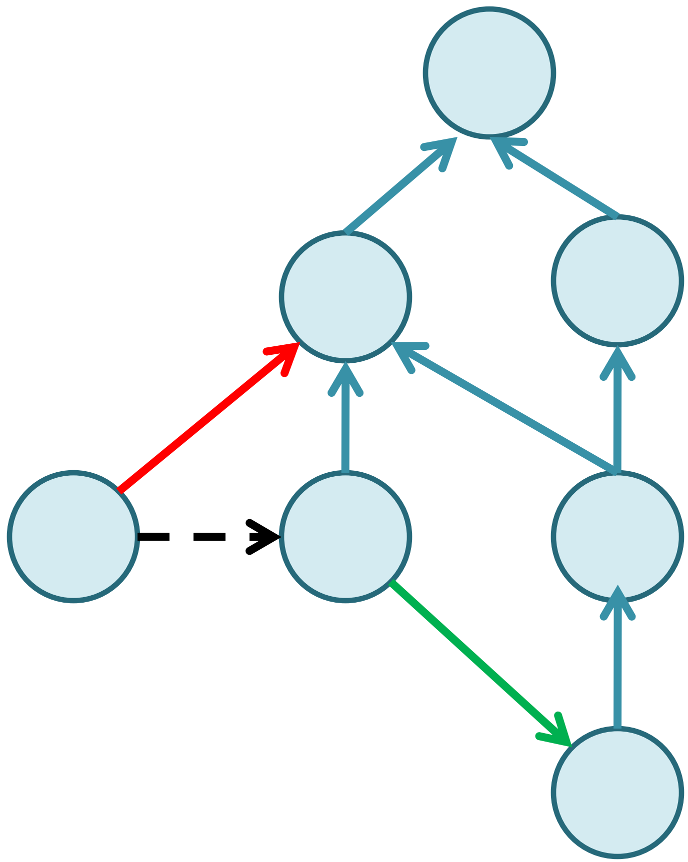
For Example



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 the smallest 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 o(log n), even if we know the structure of the network
  - we show this by a reduction from Set Cover

An Approximation Algorithm

- Say the optimal solution has m nodes
- Suppose we wanted to fire the output with probability (p – ε)
- 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

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

Papers Covered in this Thesis

Learning Evolutionary Trees

Learning Graphs (for DNA sequencing)

Learning Circuits (Gene Regulatory Networks)

Actively Learning Social Networks

Passively Inferring Social Networks

Learning Finite State Automata

What if We Cannot Manipulate the Network?

2009 Cases of Swine Flu

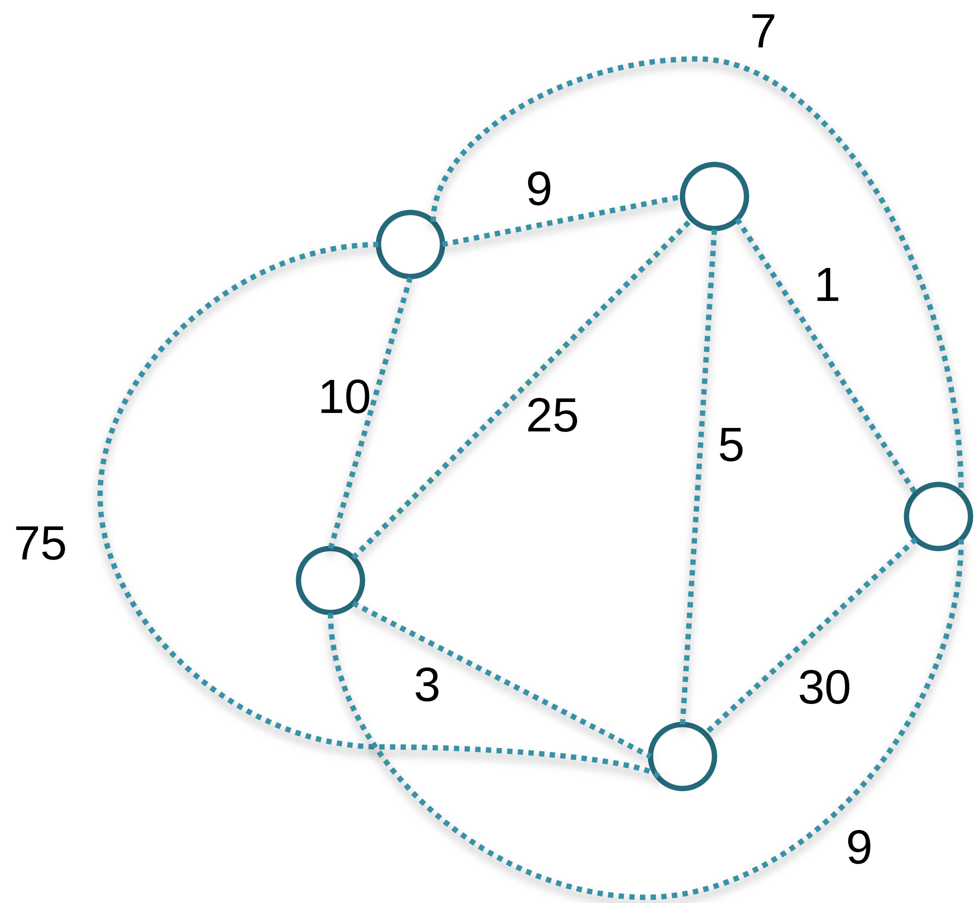
The Constraints

- 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.

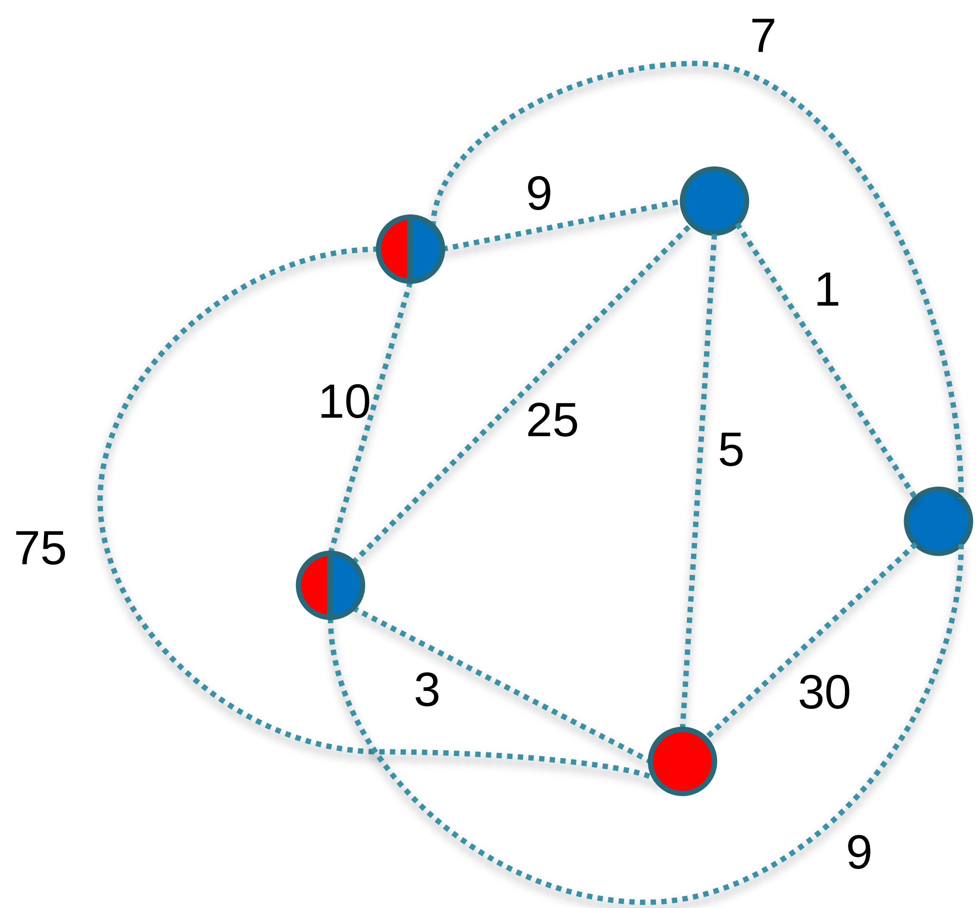
Finding the Cheapest Network

- 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

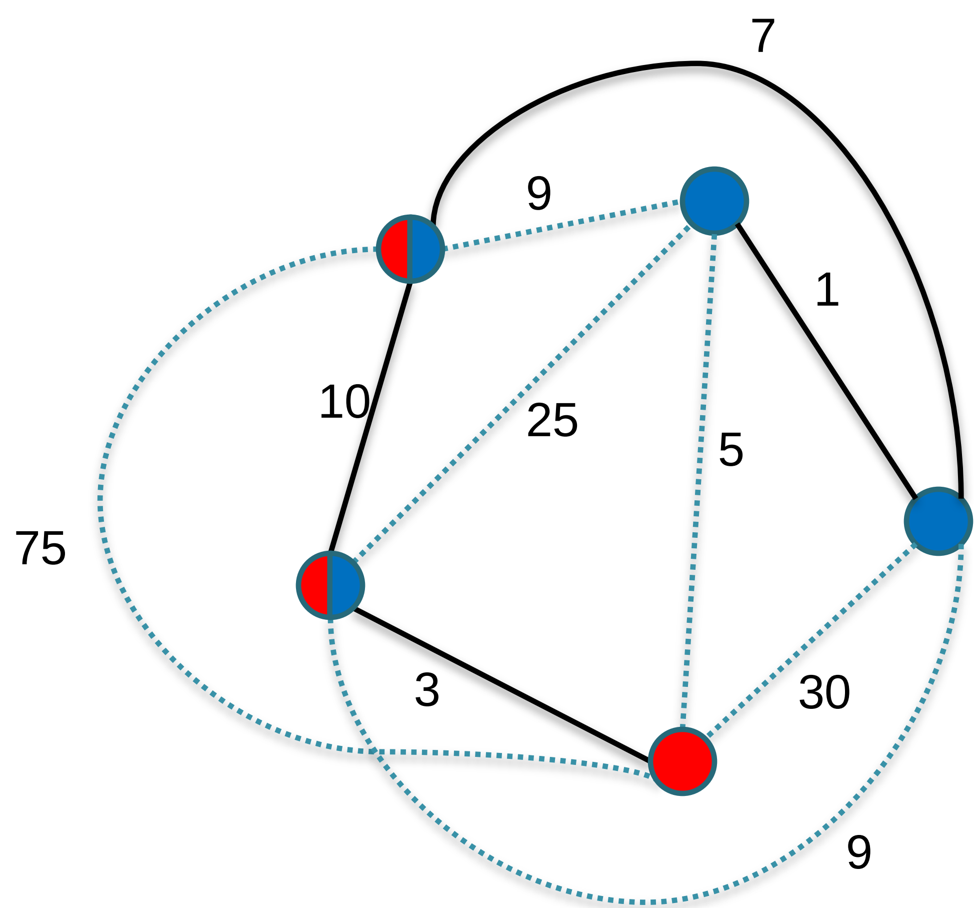
Finding the Cheapest Network Consistent with the Constraints



Finding the Cheapest Network Consistent with the Constraints



Finding the Cheapest Network Consistent with the Constraints



The Network Inference Problem

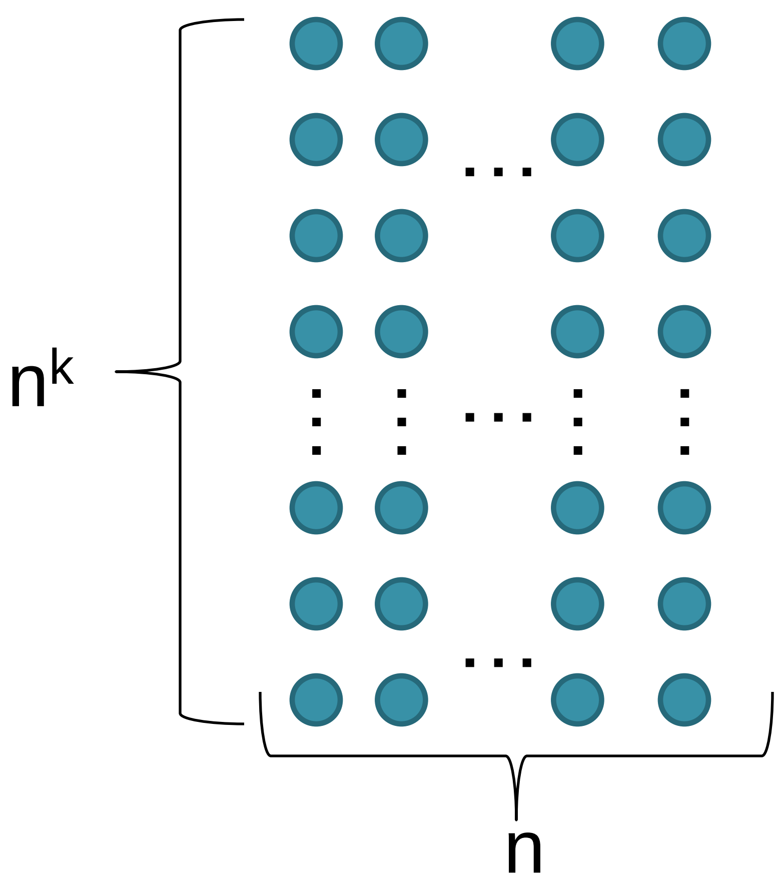
- The Network Inference Problem.
  - Given:
    - a set of vertices $V = \{v_1,\ldots,v_n\}$
    - costs c_e for each edge $e=\{v_i,v_j\}$
    - a constraint set $S = \{S_1,\ldots,S_r\}$, with $S_i \subseteq V$
  - Find: a set E of edges of lowest cost such that each S_i induces a connected subgraph 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,…,v_n}
  - C = {C_1, C_2,…,C_j}, with $C_i \subseteq U$
  - The Hitting Set problem is to minimize |H|, where $H \subseteq U \text{ s.t.} \forall C_i \; H \cap C_i \neq \phi$

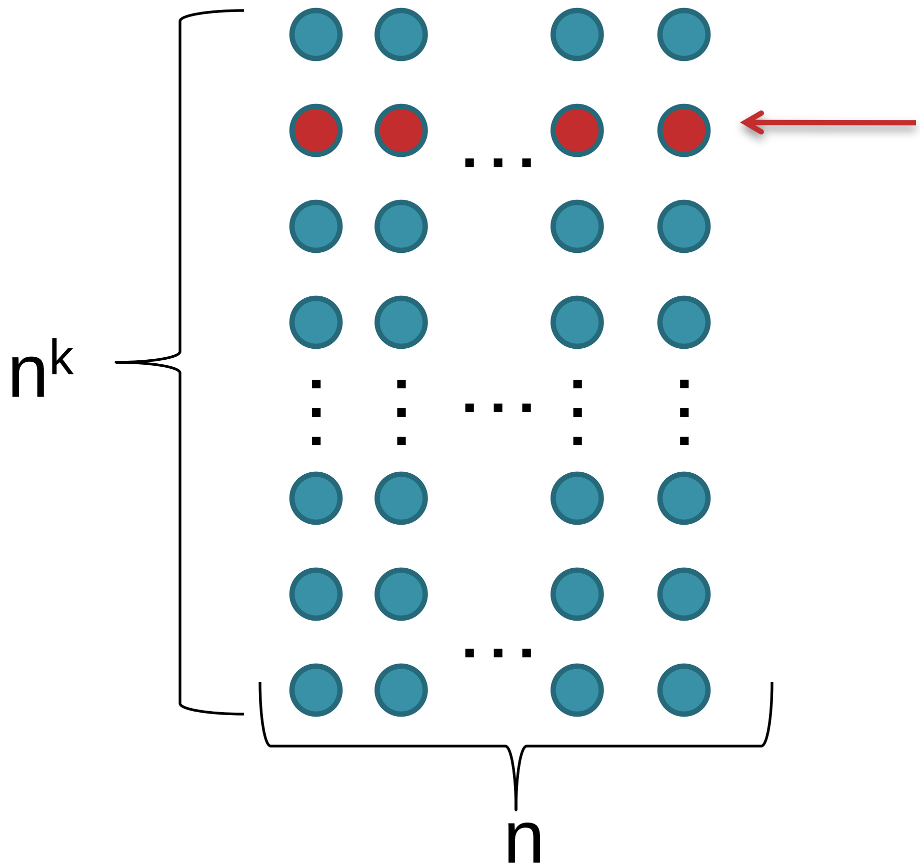
An Offline L.B. continued

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



An Offline L.B. continued

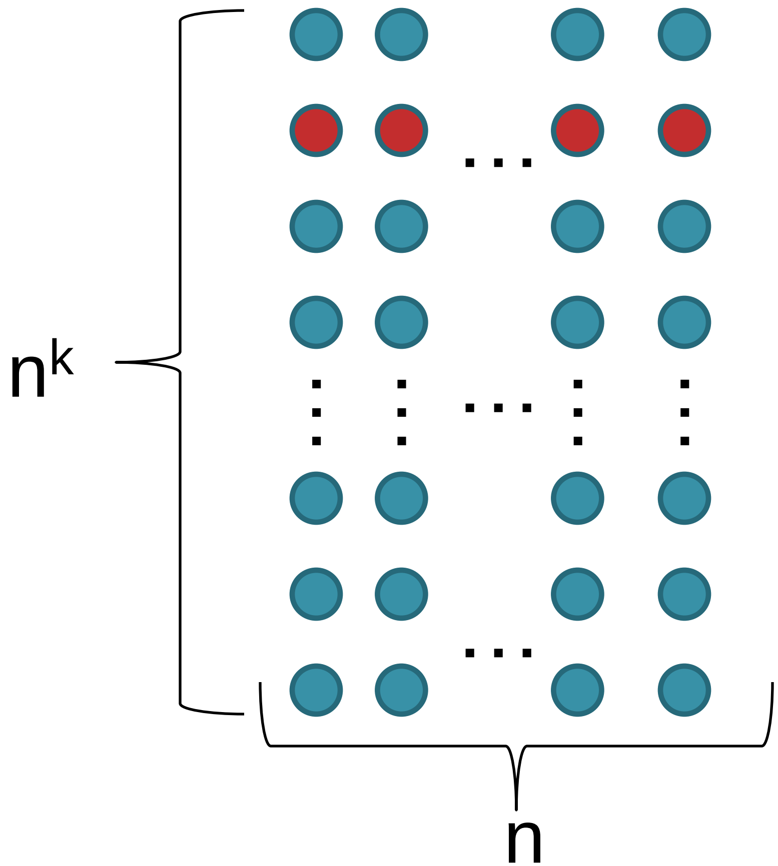
- Reduction from Hitting Set
- For a constant k, We make a N.I. instance



Each row corresponds to the elements in the Hitting Set instance

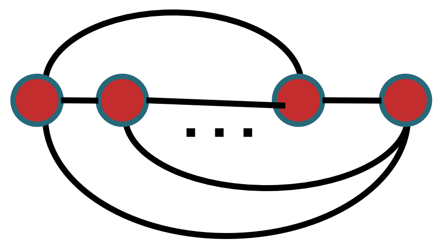
An Offline L.B. continued

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



An Offline L.B. continued

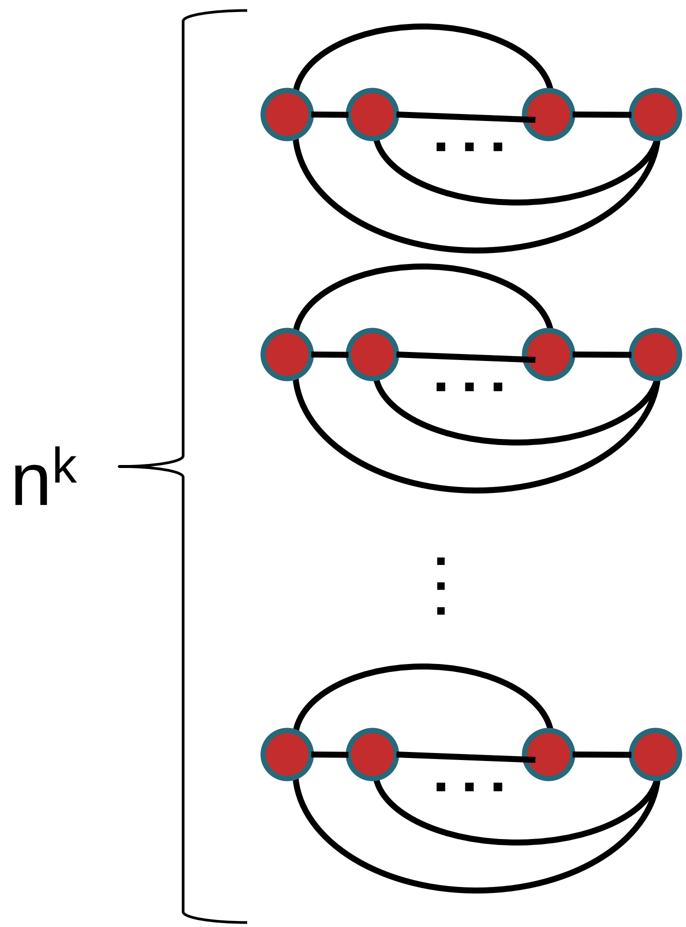
- Constraints: first, for each row, give all pairwise constraints:



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

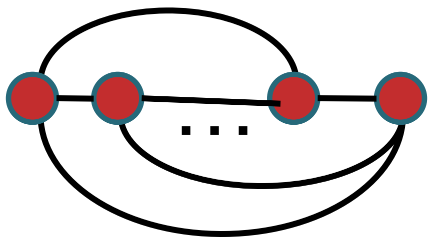
An Offline L.B. continued

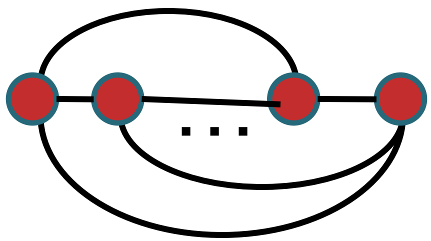
- Now we have n^k rows of cliques



An Offline L.B. continued

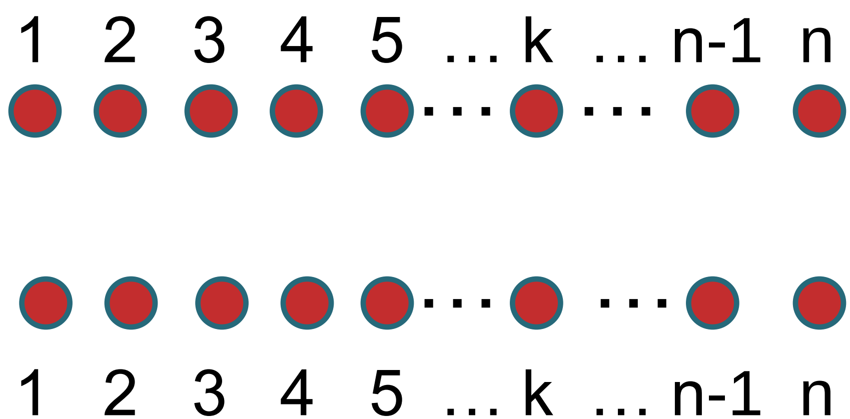
- For each pair of rows:





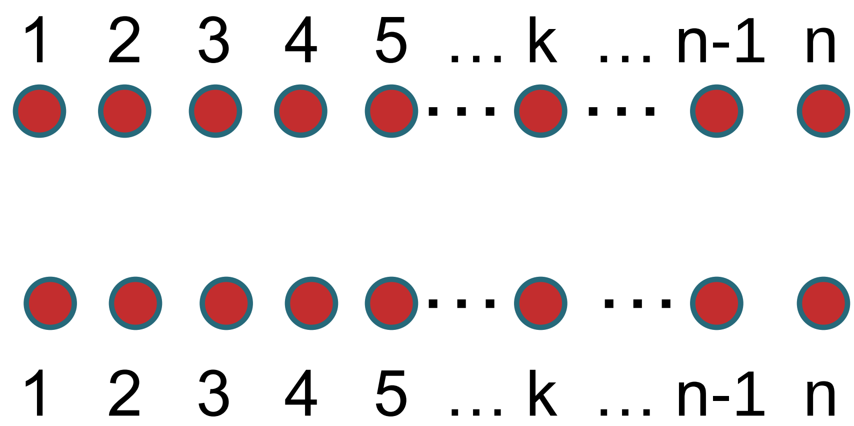
An Offline L.B. continued

- For each pair of rows:



An Offline L.B. continued

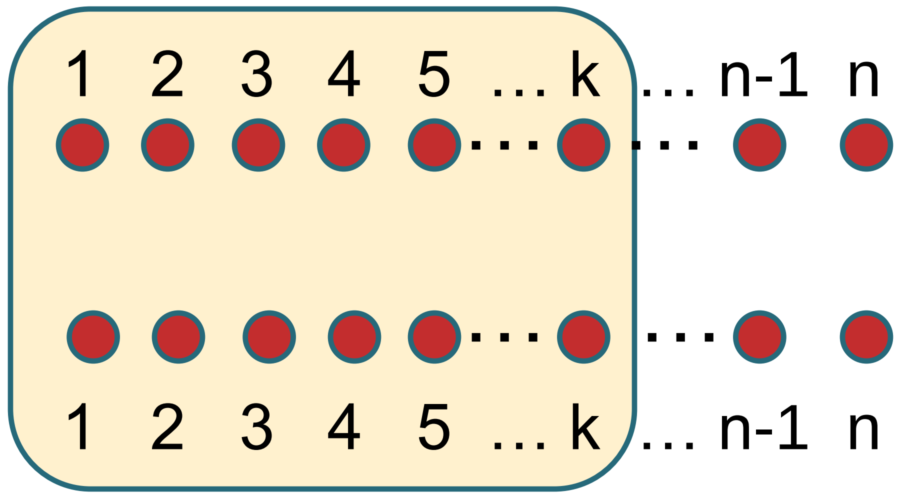
- For each pair of rows:



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

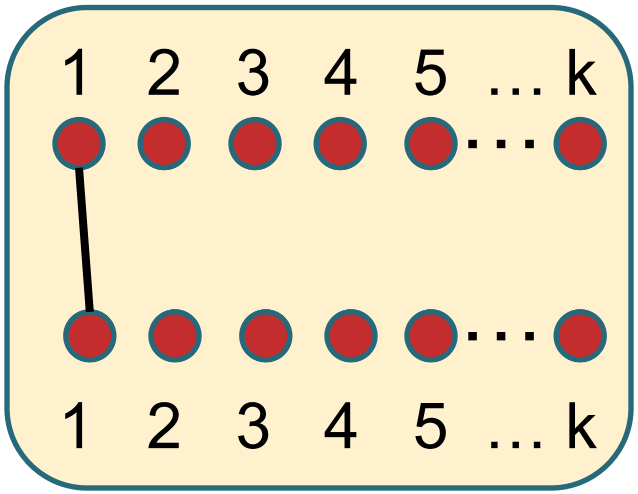
An Offline L.B. continued

- For each pair of rows:

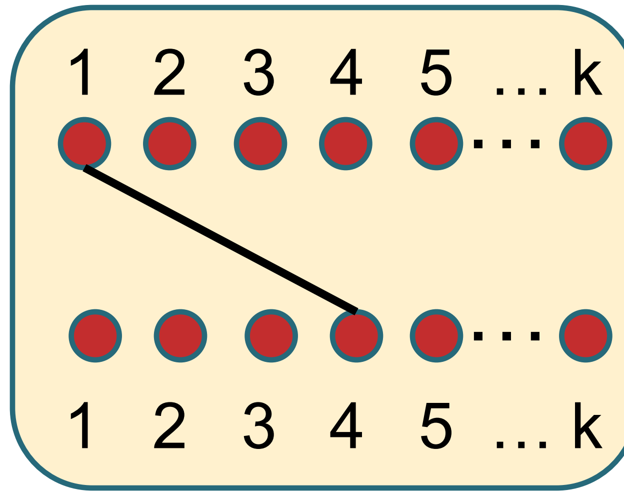


- w.l.o.g. for the Hitting Set constraint
  - $C_i = \{v_1, v_2, \ldots, v_k\}$
  - we will add the constraint:

An Offline L.B. continued

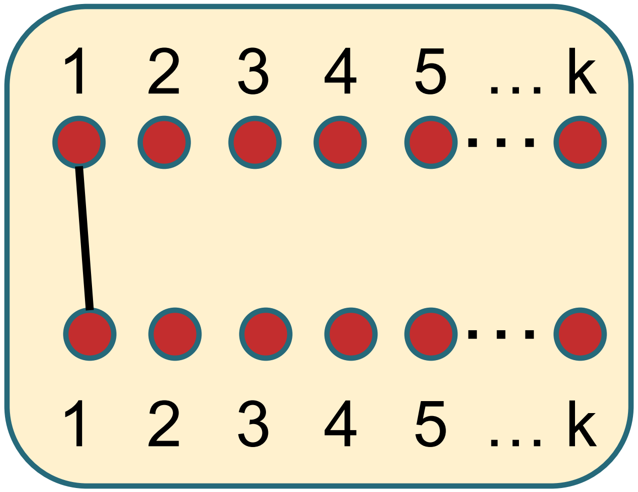


corresponds to adding v_1 to H

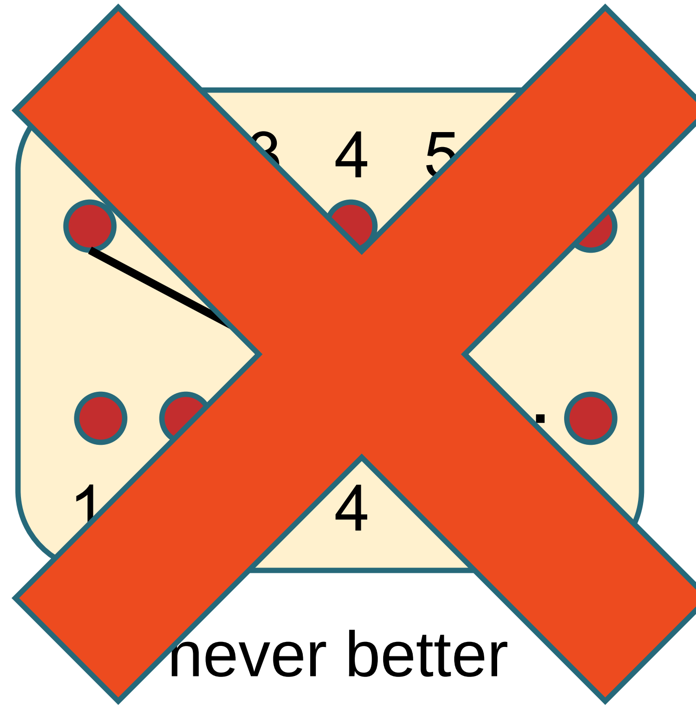


never better

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

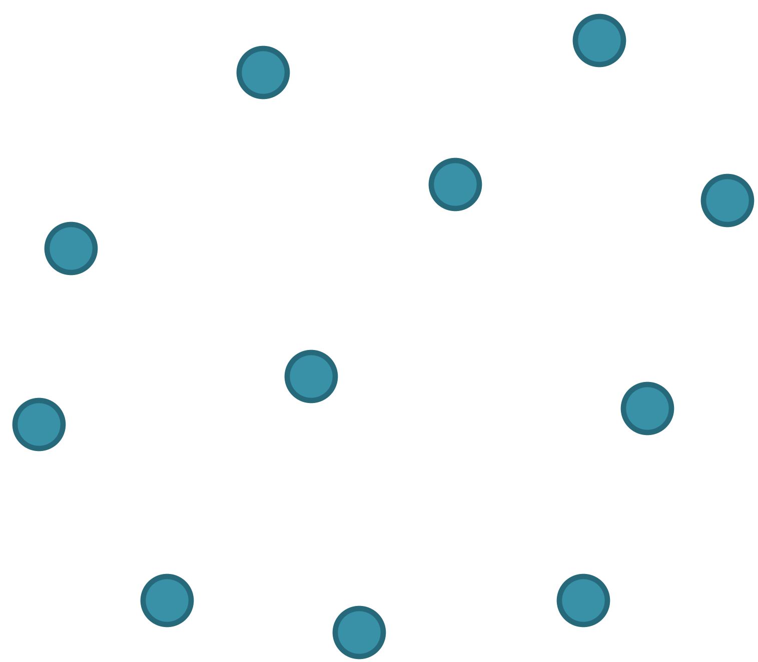
- Theorem: There is a $O(\log(n)+\log(r))$ approximation algorithm to OPT
- Proof:
  - Let **C** sum over all constraints S_i, the number of components S_i induces in G minus 1.
  - Now consider the greedy algorithm: while C > 0, add to E the edge that has the lowest ratio of c_e to ΔC.
  - This greedy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

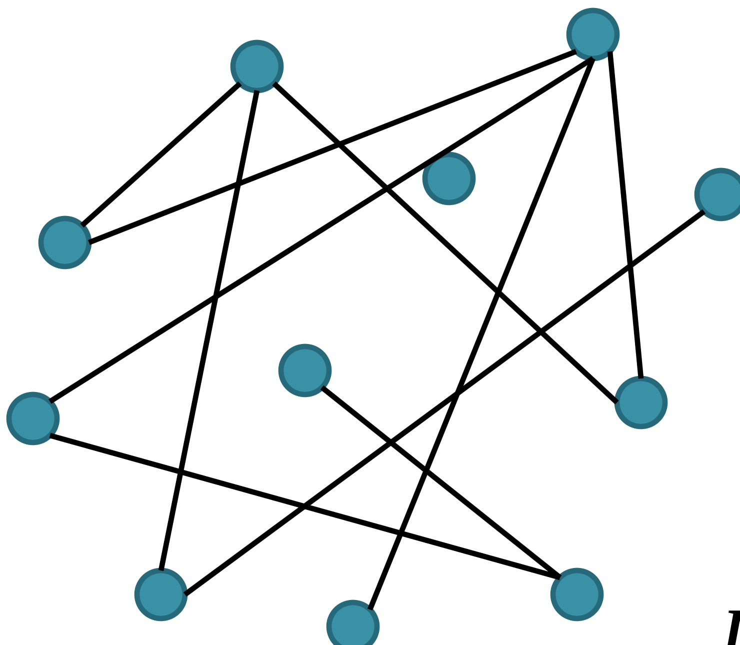
Online Algorithm Against Oblivious Adversary

$O(n^{2/3}\log^{2/3}n)$-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 $\mathrm{OPT} = \Omega(n)$, we are done.

Other Online Results

- The competitive ratio for uniform cost stars and paths is $\theta(\log n)$.
  - 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 $\Omega(n)$-competitive lower bound and $O(n \log n)$-competitive algorithm.

Papers Covered in this Thesis

Learning Evolutionary Trees

Learning Graphs (for DNA sequencing)

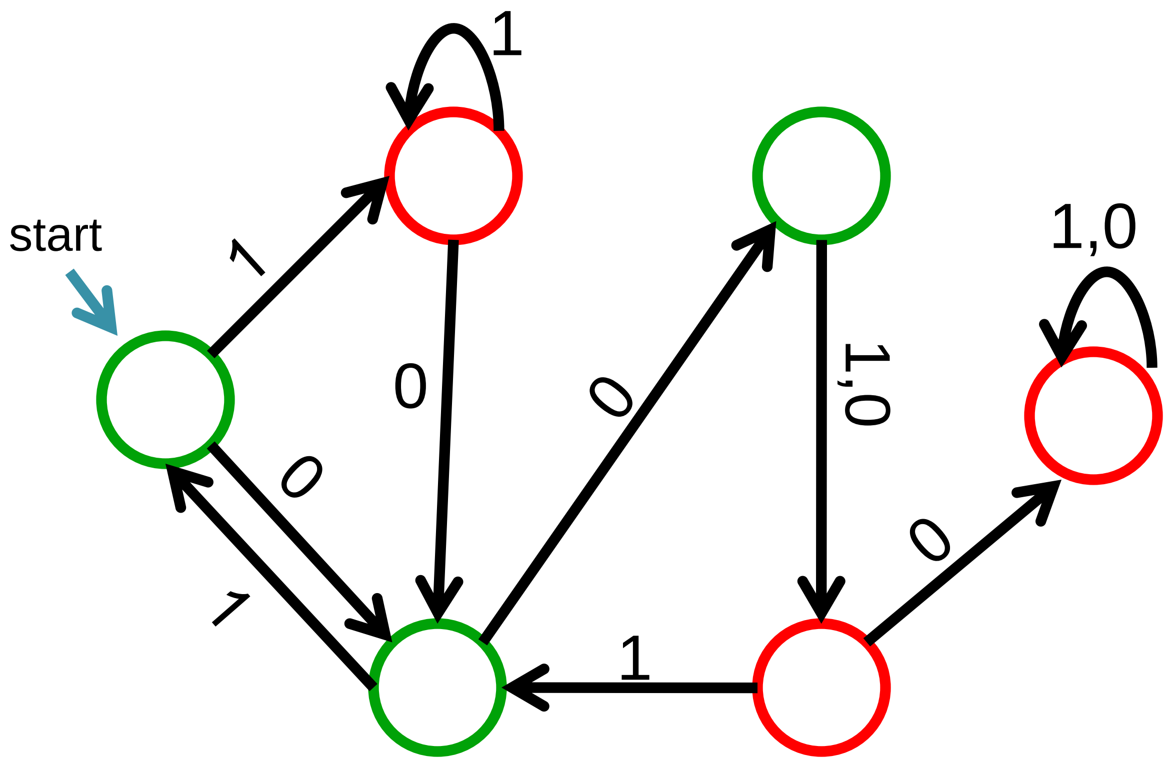
Learning Circuits (Gene Regulatory Networks)

Actively Learning Social Networks

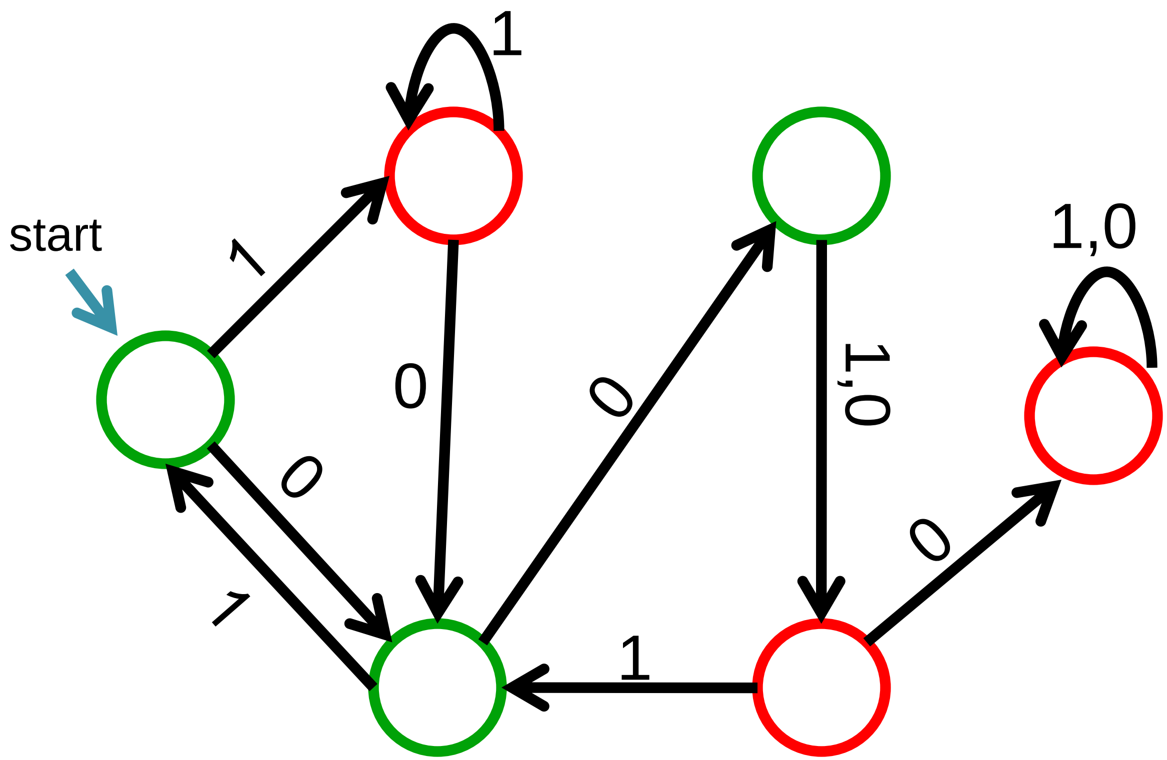
Passively Inferring Social Networks

Learning Finite State Automata

Learning FSA with Label Queries

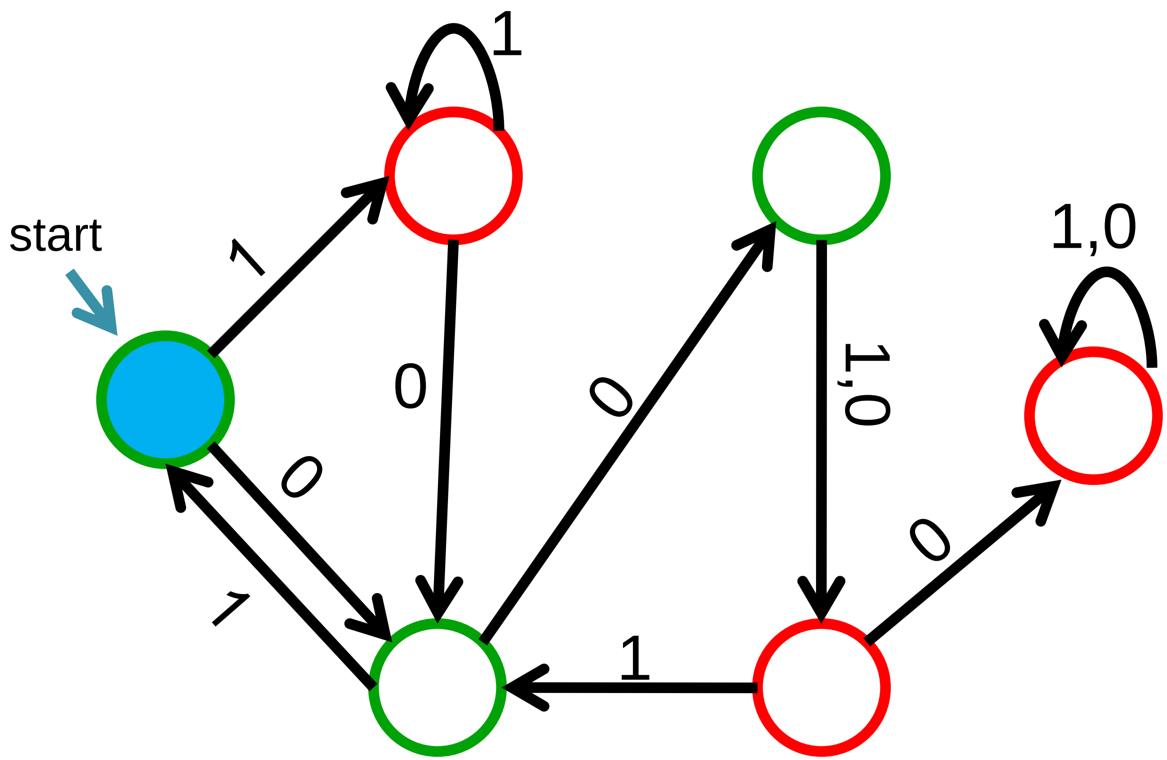


Learning FSA with Label Queries



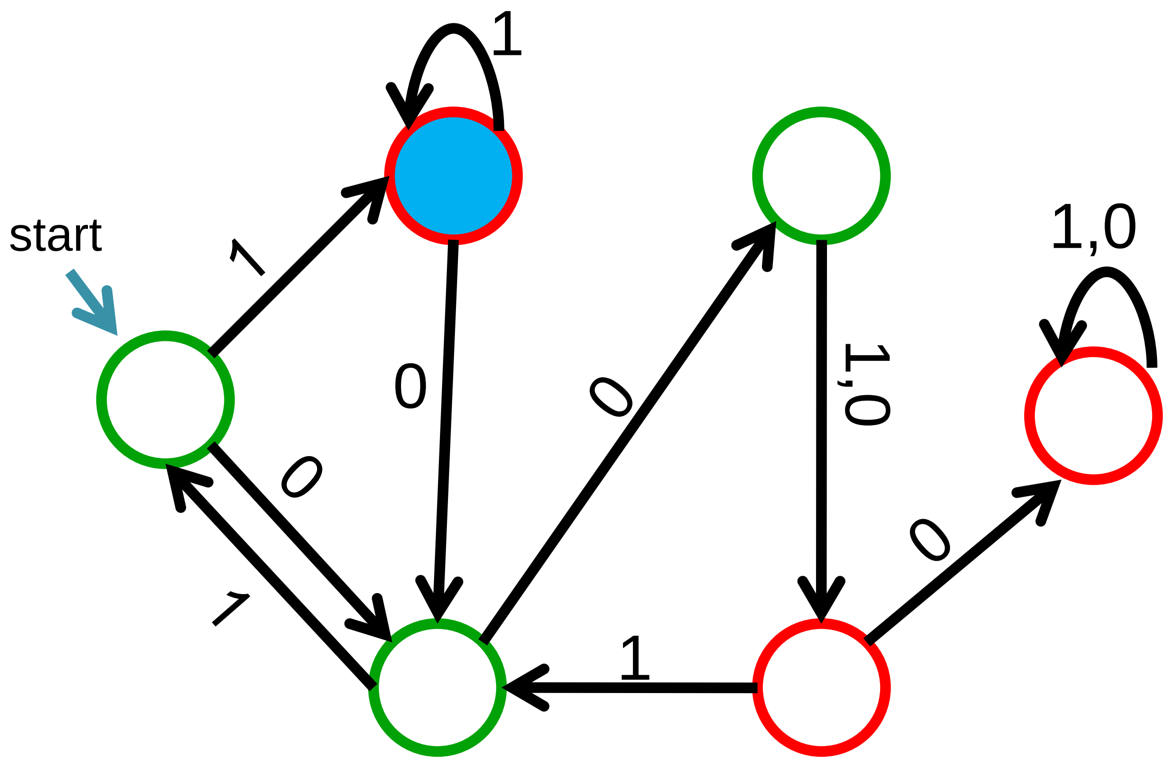
MQ("10011") = ?

Learning FSA with Label Queries



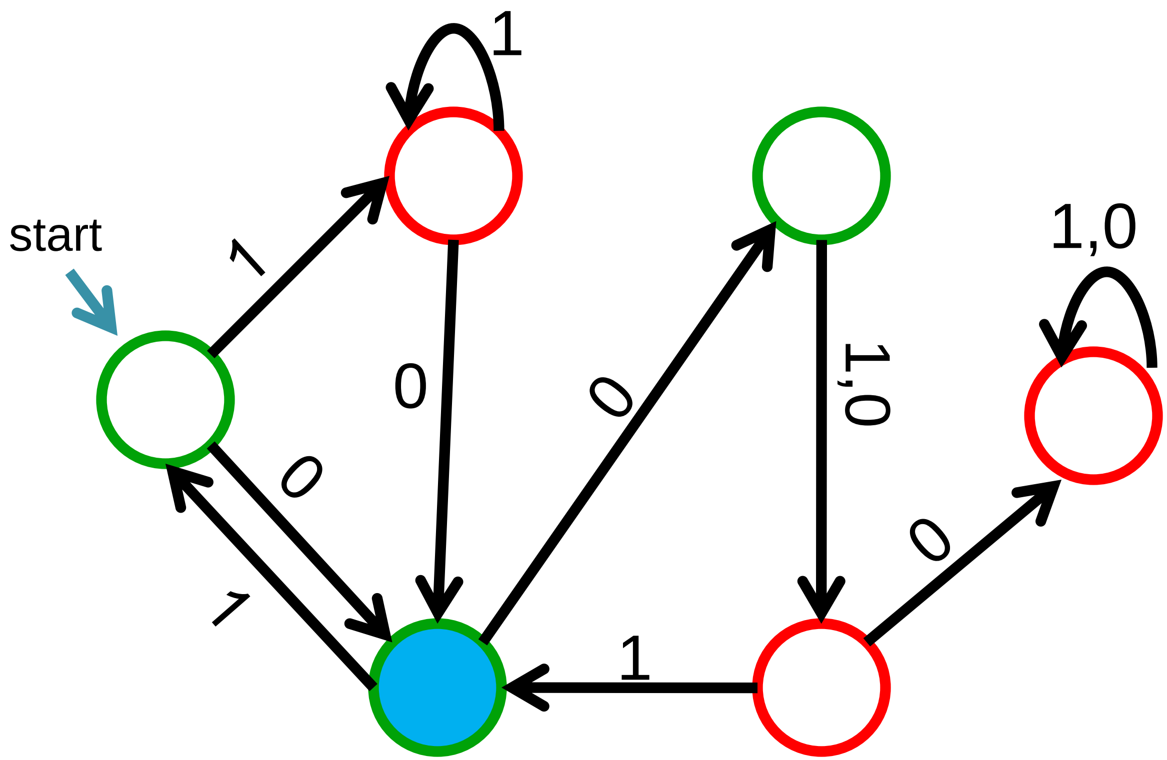
MQ("10011") = ?

Learning FSA with Label Queries



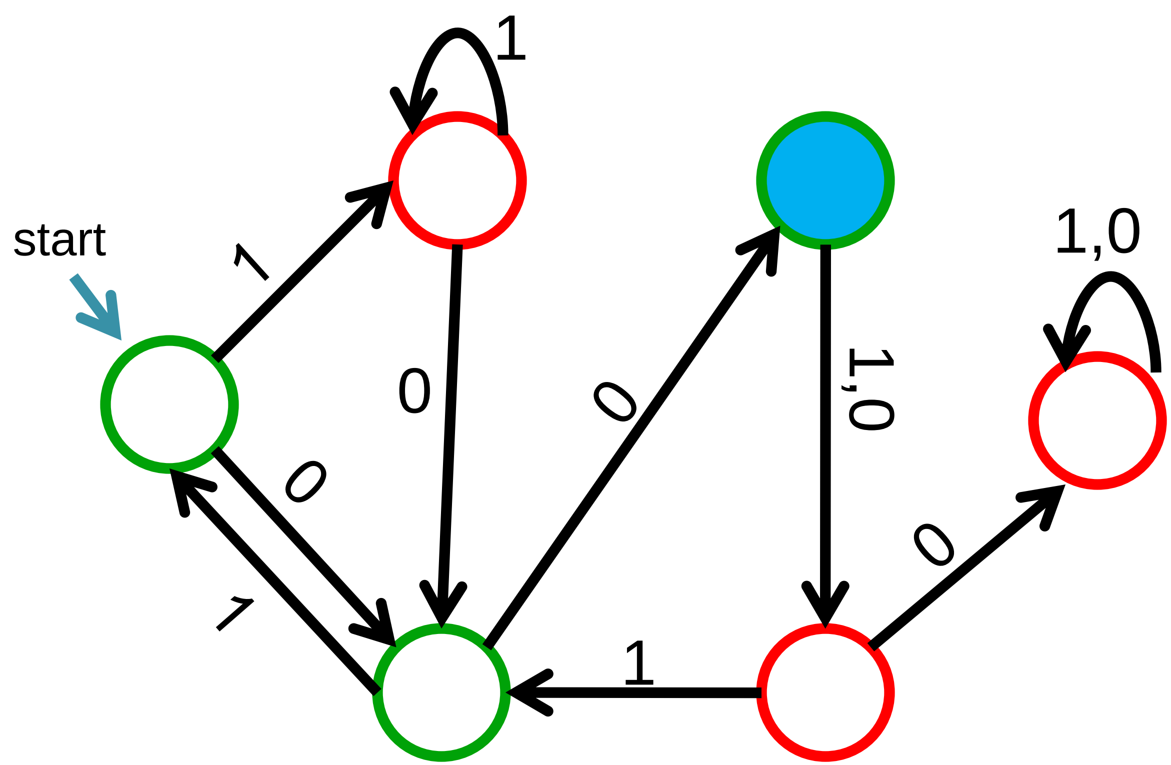
MQ(“10011”) = ?

Learning FSA with Label Queries



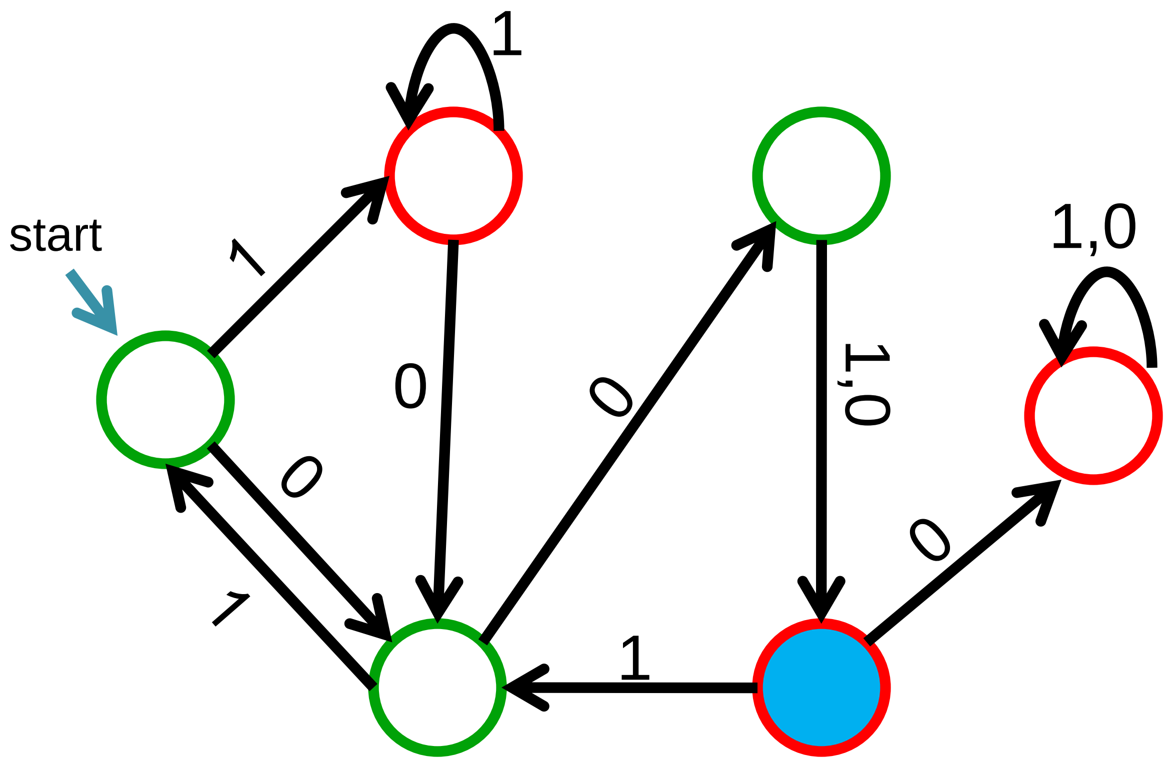
MQ("10011") = ?

Learning FSA with Label Queries



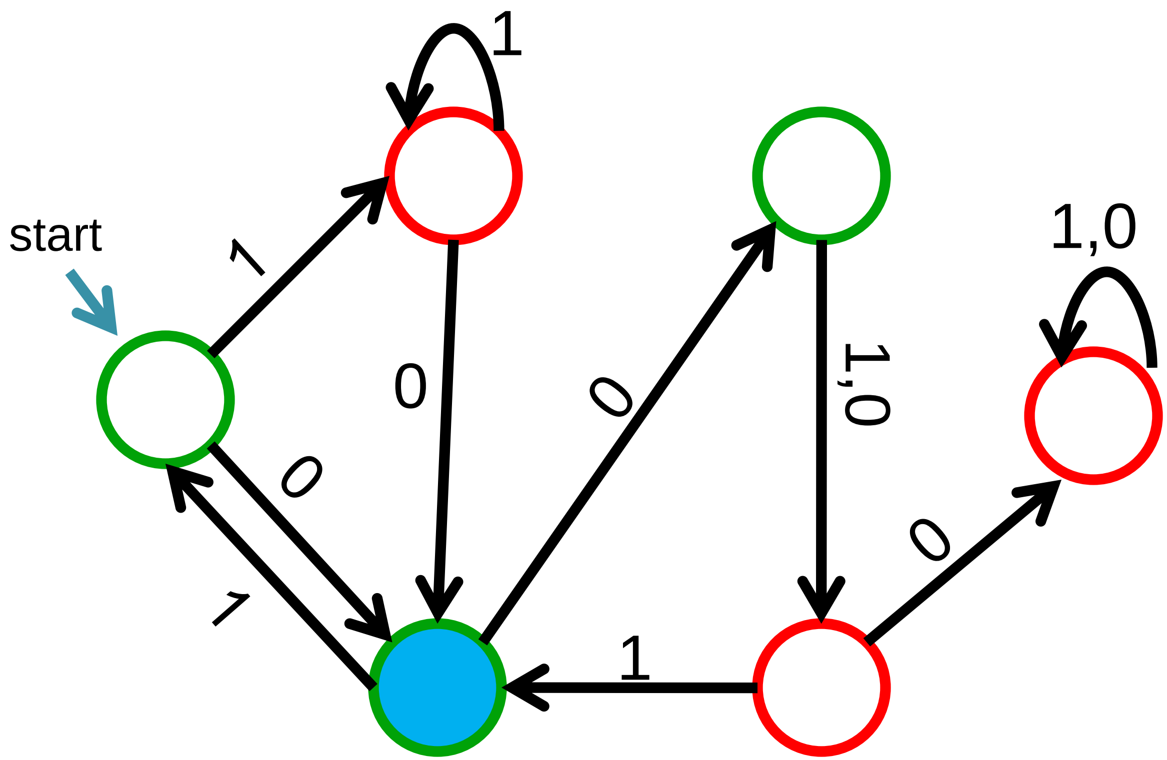
MQ(“10011”) = ?

Learning FSA with Label Queries



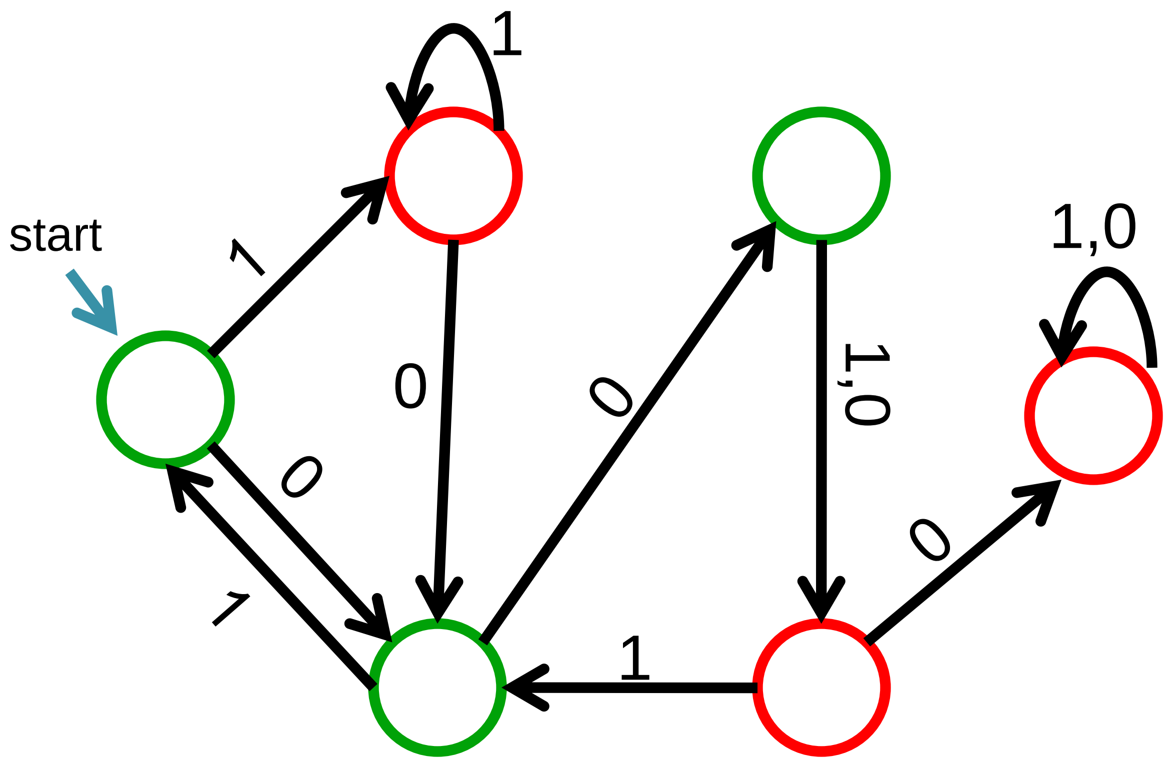
MQ(“10011”) = ?

Learning FSA with Label Queries

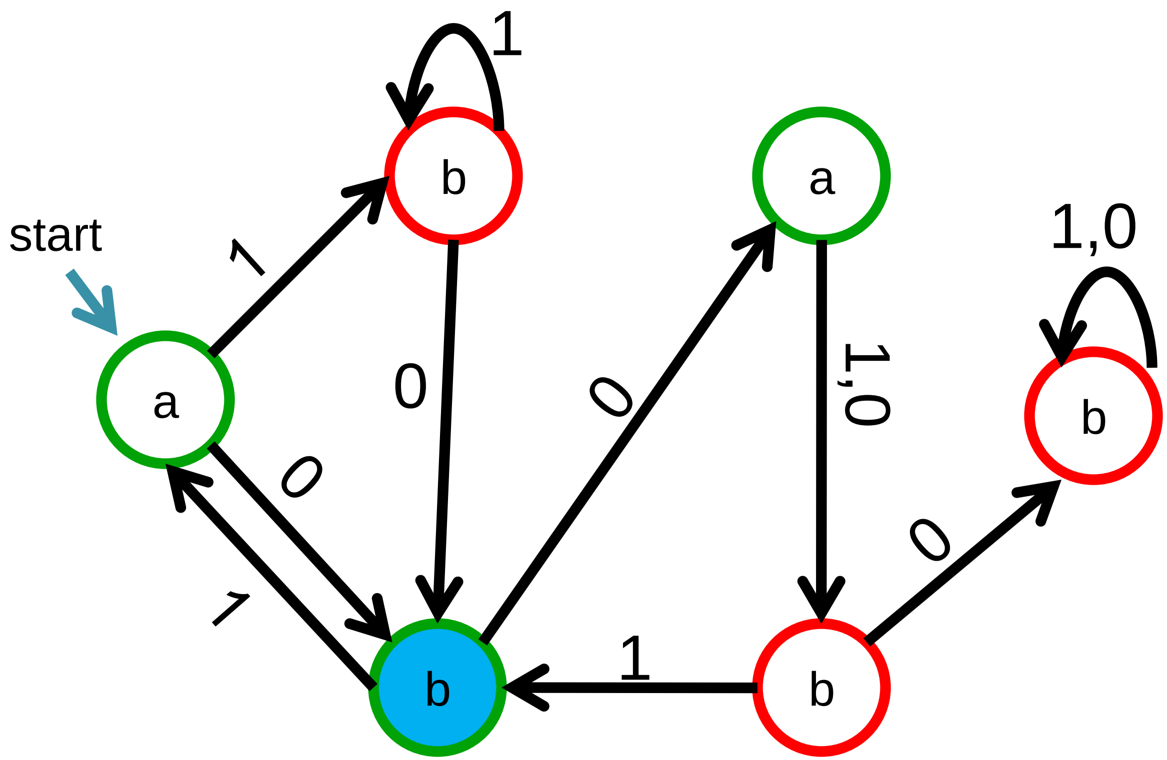


MQ("10011") = Accept

Learning FSA with Label Queries



Learning FSA with Label Queries



MQ(“10011”) = (Accept,b)

Summary

- We explored learning Interaction Networks in many contexts
- Applications include evolutionary tree reconstruction, learning DNA structure, gene regulatory networks, social networks, viral spread of diseases, and language learning.
- Similarity in techniques – and opportunity to apply results to new domains.
- Clearly, many more problems can be solved from this perspective.

Thank You!

Questions?