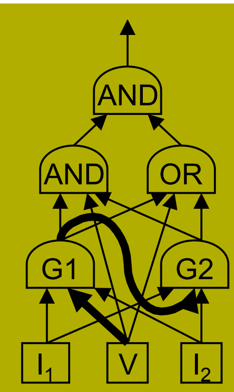
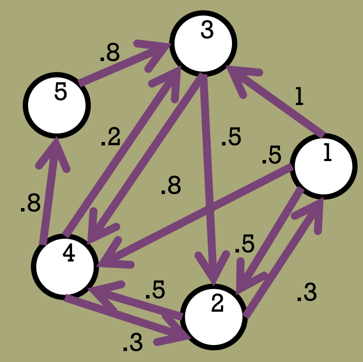
Learning Analog Circuits, Graphical Models, and Social Networks by Injecting Values



1/27/2010



Talk @
IBM Research
Theory Seminar

Lev Reyzin
Yahoo! Research
New York

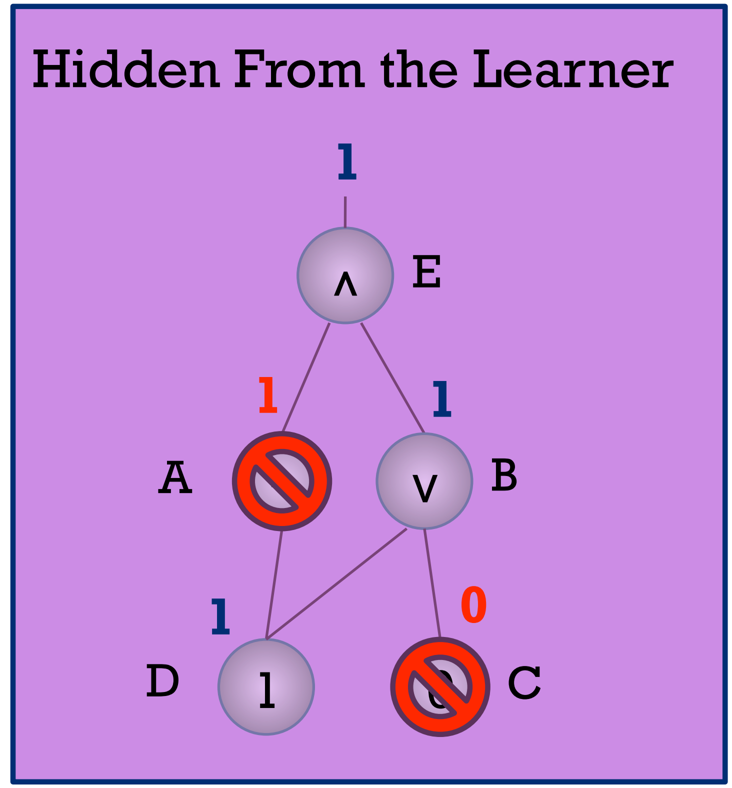
+

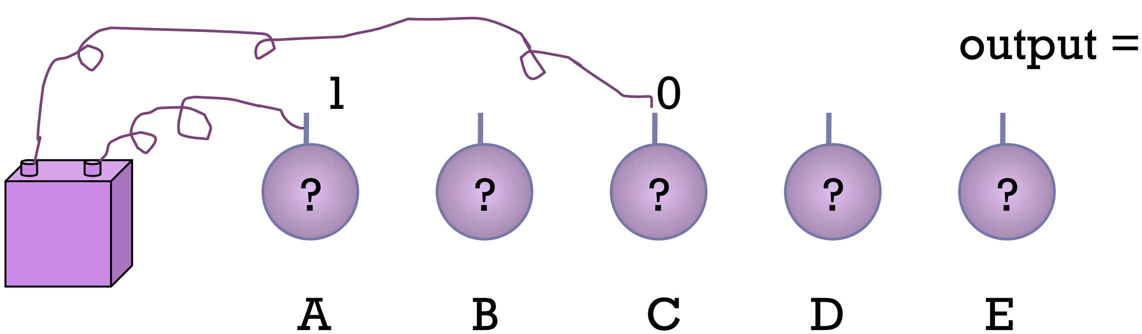
Part I
Analog Circuits

work done with Dana Angluin, James Aspnes, and Jiang Chen

+ The Value Injection Query Model

- Introduced by [AACW ’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:

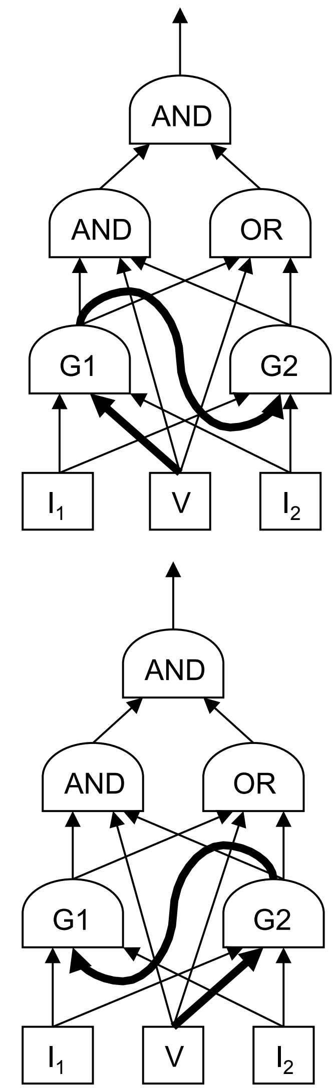




+ The Learning Problem

- **Behavioral equivalence:** Two circuits C and C' are behaviorally equivalent if for any experiment s, C(s)=C'(s).

- **The Problem:** Given query access to a hidden circuit C*, find a circuit C behaviorally equivalent to C* by making value-injection queries.



[ACCW '06]

+ Motivation for The Model

- To model gene regulatory networks as Boolean networks
- to represent gene expressions and disruptions

Previous gene regulatory network model	Fully controllable.	All gates are observable.
Existing circuit learning models	Only inputs can be manipulated.	Only the output is observable.
Value Injection Query model [AACW '06]	Fully controllable.	Only the output is observable.

IN BETWEEN

+

[AACW '06] Results for Boolean Circuits

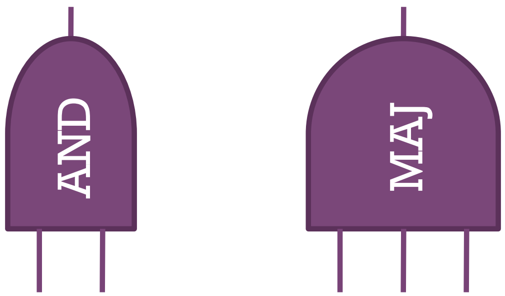
Depth	**Fan-in**	**Gates**	**Learnability**
Unbounded	Unbounded	AND/OR	$2^{\Omega(N)}$ queries
Unbounded	2	AND/OR	NP-hard
Constant	Unbounded	AND/OR/Θ_2	NP-hard
Log	Constant	Arbitrary	Poly-time **(NC1)**
Constant	Unbounded	AND/OR/NOT	Poly-time **(AC0)**

Looking at Large Alphabet Circuits

- Gene regulatory networks have more states than just expressed and disrupted.
- A larger alphabet than {0,1} is needed to more fully represent many other types of networks.
- Looking at what happens for large alphabet size is a natural, interesting theoretical question.
- Helps us get at analog circuits.

Large-Alphabet Circuits

Gates in Boolean Circuits



Input 1	Input 2	Output
1	1	O_1
1	0	O_2
0	1	O_3
0	0	O_4

Gates in Large-Alphabet circuits

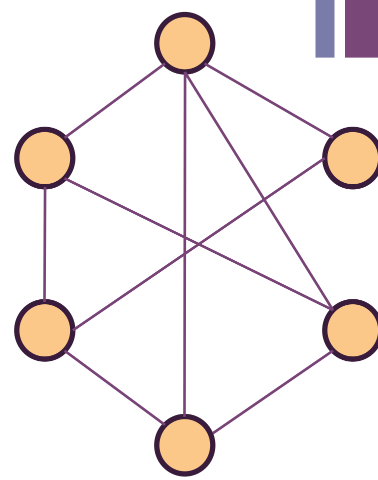
Input 1	Input 2	Output
A	A	O_1
A	B	O_2
A	C	O_3
B	A	O_4
B	B	O_5
B	C	O_6
C	A	O_7
C	B	O_8
C	C	O_9

+ What Happens For Large-Alphabet Circuits? (Our Results)

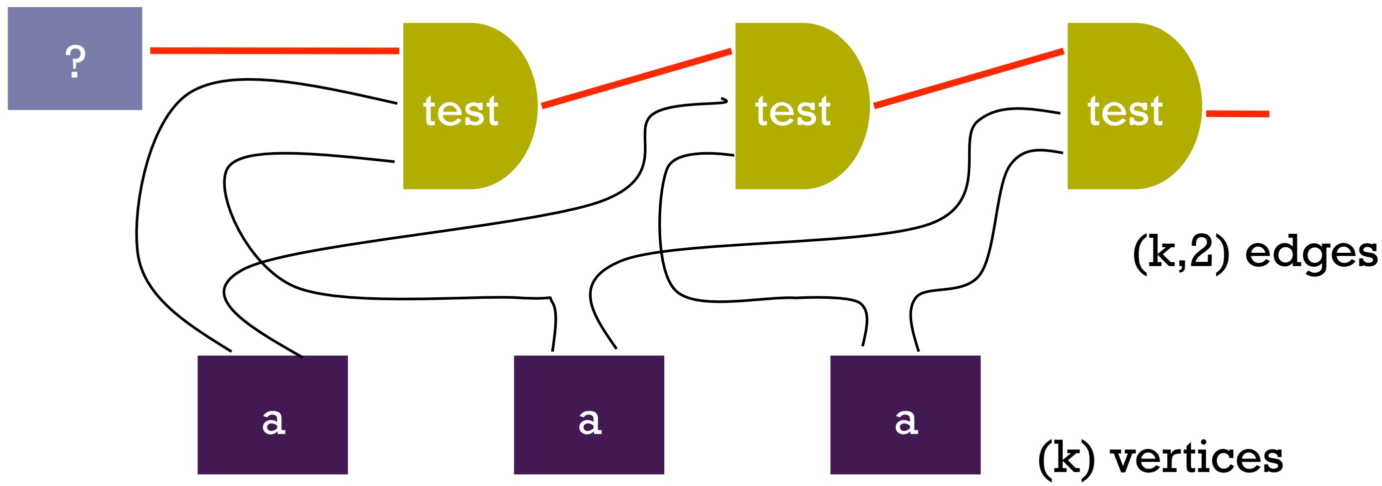
- There is evidence that learning log depth, constant fan-in large-alphabet circuits may be computationally intractable
- Circuits of **bounded shortcut width** (and **transitively reduced circuits**) can be learned in time polynomial in the number of wires and the alphabet size.
- We can approximately learn bounded shortcut-width analog circuits that satisfy a Lipshitz condition.

Hardness of Learning Large Alphabet Circuits

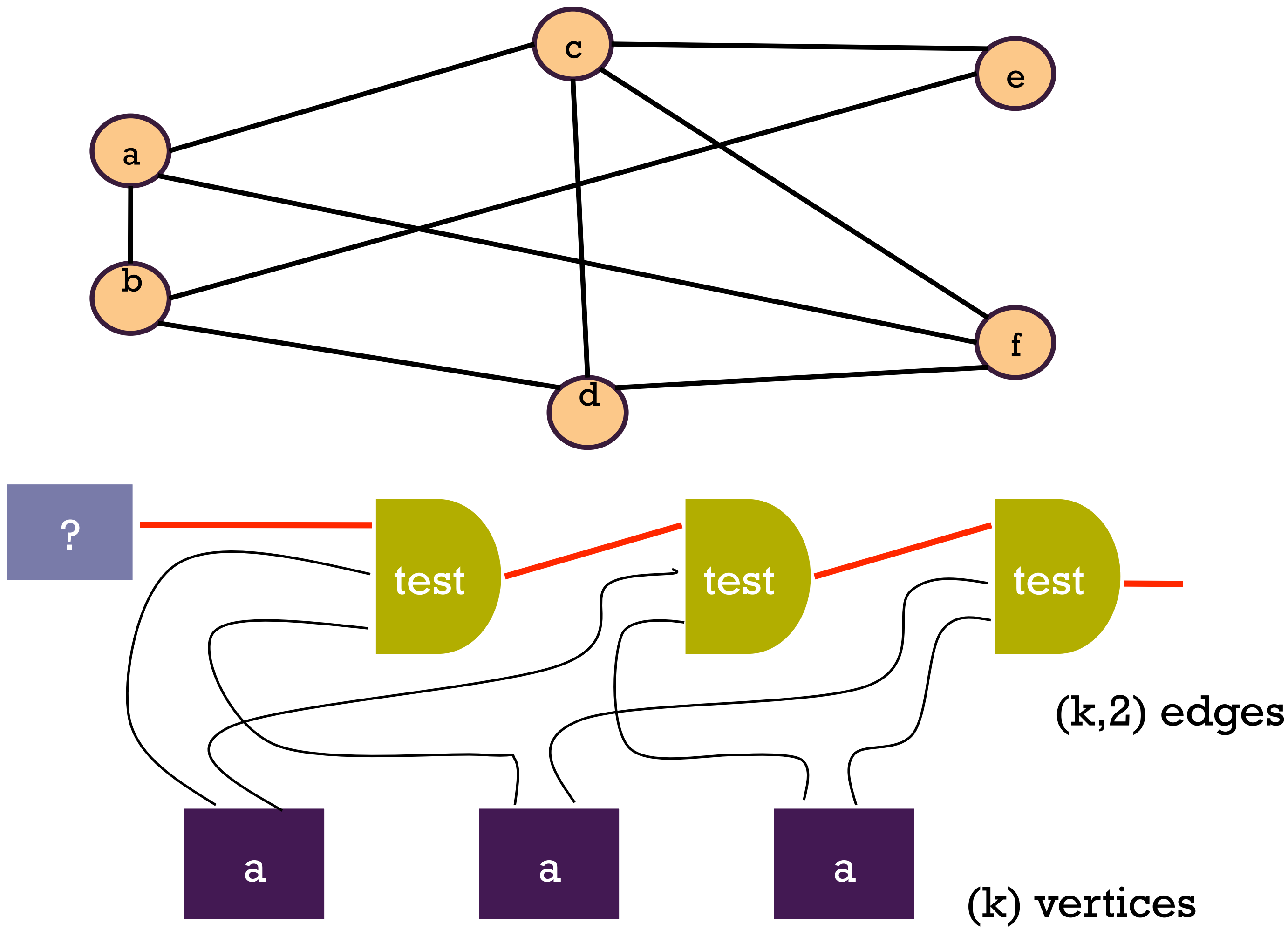
- Consider the problem on input (G,k) of telling whether the graph G on n vertices has a clique of size k



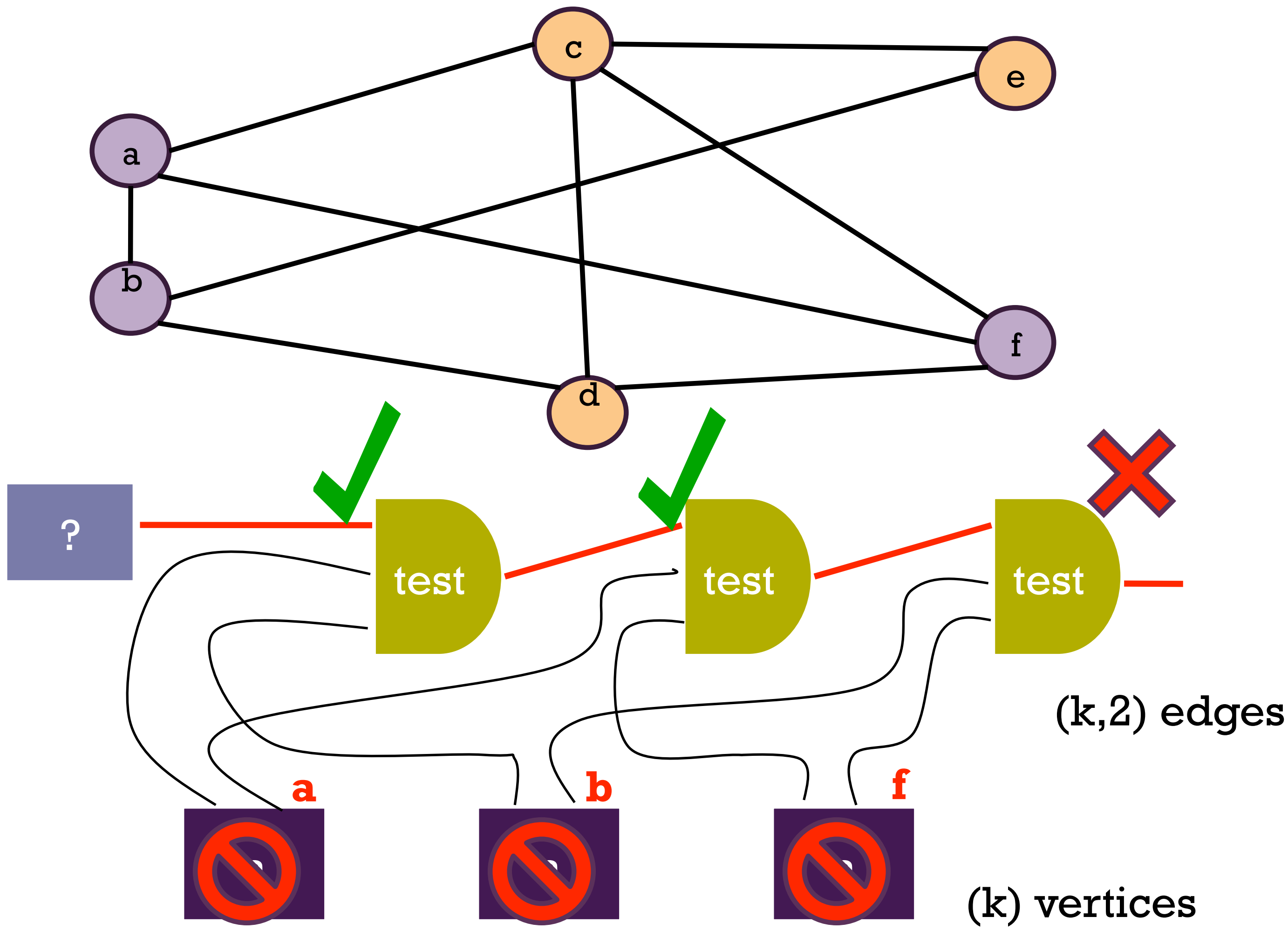
- We give a reduction that turns a large-alphabet circuit learning algorithm into a clique tester



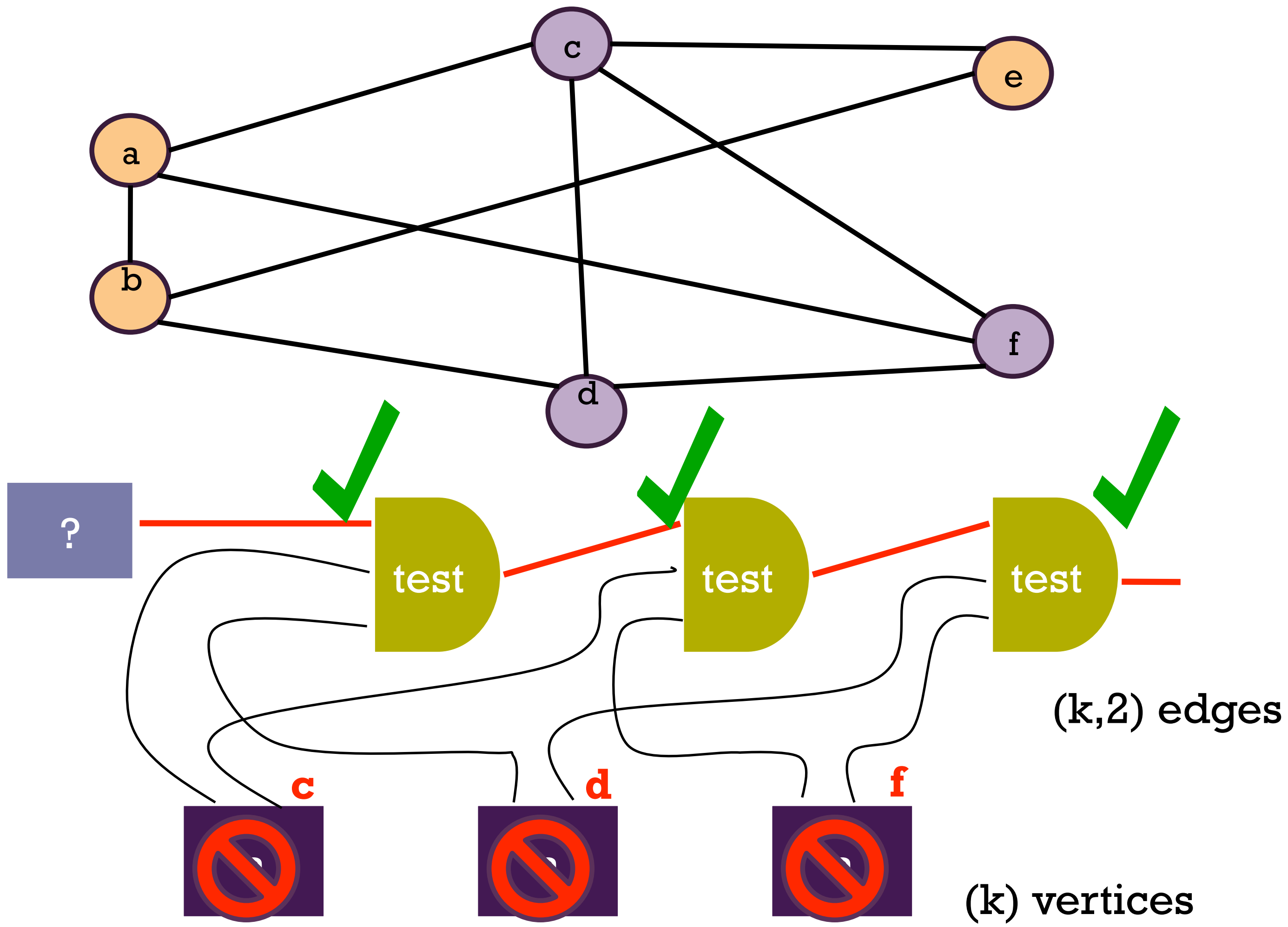
+ Reducing the Clique Problem to Circuit Learning



+ Reducing the Clique Problem to Circuit Learning



+ Reducing the Clique Problem to Circuit Learning



+ # Hardness of Learning Circuits of Unrestricted Topology

- The clique problem is complete for the **parameterized complexity class** W[1]
  - There is no known algorithm for the clique problem that runs in time $f(k)n^c$ (and we believe one doesn't exist)

- **Theorem** **An algorithm for learning circuits polynomial in the number of wires and alphabet size would imply fixed parameter tractability for all problems in W[1]**

+ To Compare with the Boolean Case

Boolean Circuits [AACW '06]:

Depth	Fan-in	Gates	Learnability
Log	Constant	Arbitrary	Poly-time

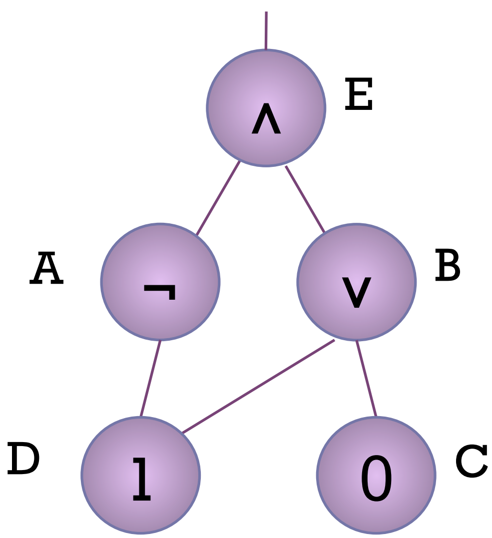
Large Alphabet Circuits:

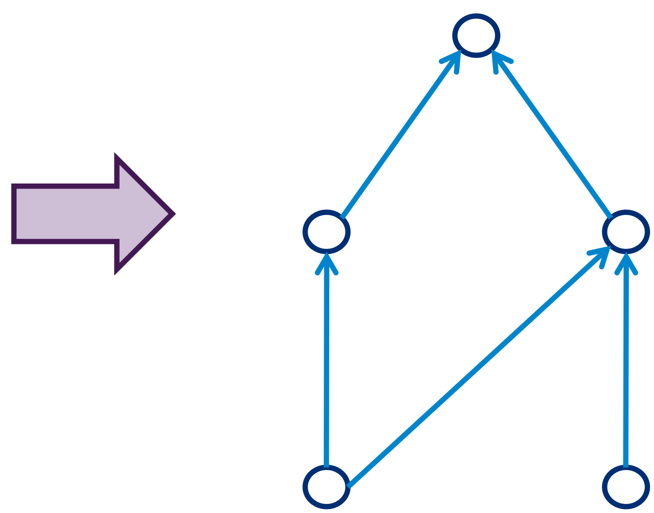
Depth	Fan-in	Gates	Learnability
Log	Constant	Arbitrary	W[1] Hard

This motivates looking at classes of large-alphabet circuits with restricted topology

+ A Circuit's Underlying Graph

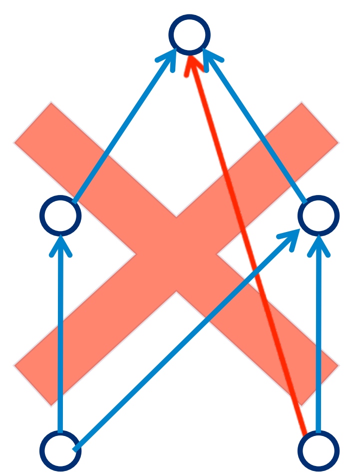
We only consider circuits whose simple, connected, directed graphs are acyclic.





+ Transitively Reduced Circuits

A circuit is transitively reduced if its underlying directed graph has no shortcuts. If (u,v) is an edge and there is a path of length ≥ 2 from u to v, then (u,v) is a **shortcut edge**



Distinguishing Tables

- For each wire w, we keep a distinguishing table. A 1 entry in $T_w(\sigma, \tau)$ means alphabet values σ and τ are **distinguishable**. For each 1 entry we keep a corresponding **distinguishing path** and a "processed bit."

Gate functions

A

In	Out
a	a
b	a
c	b

B

Out
a

Distinguishing Tables

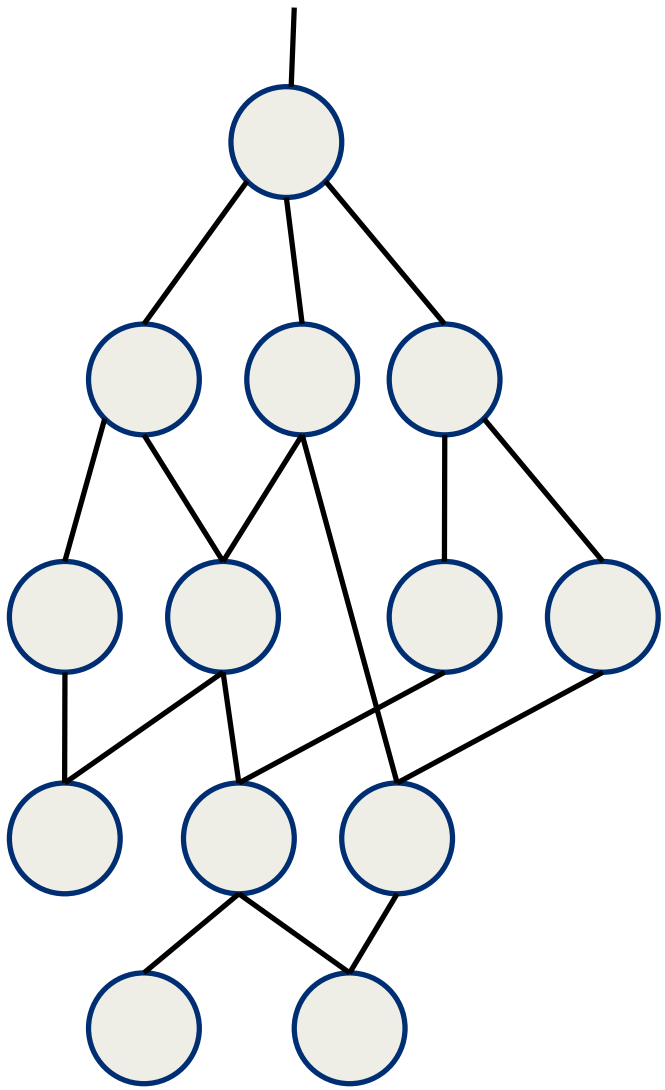
A

	a	**b**	**c**
a	-	1	1
b	-	-	1
c	-	-	-

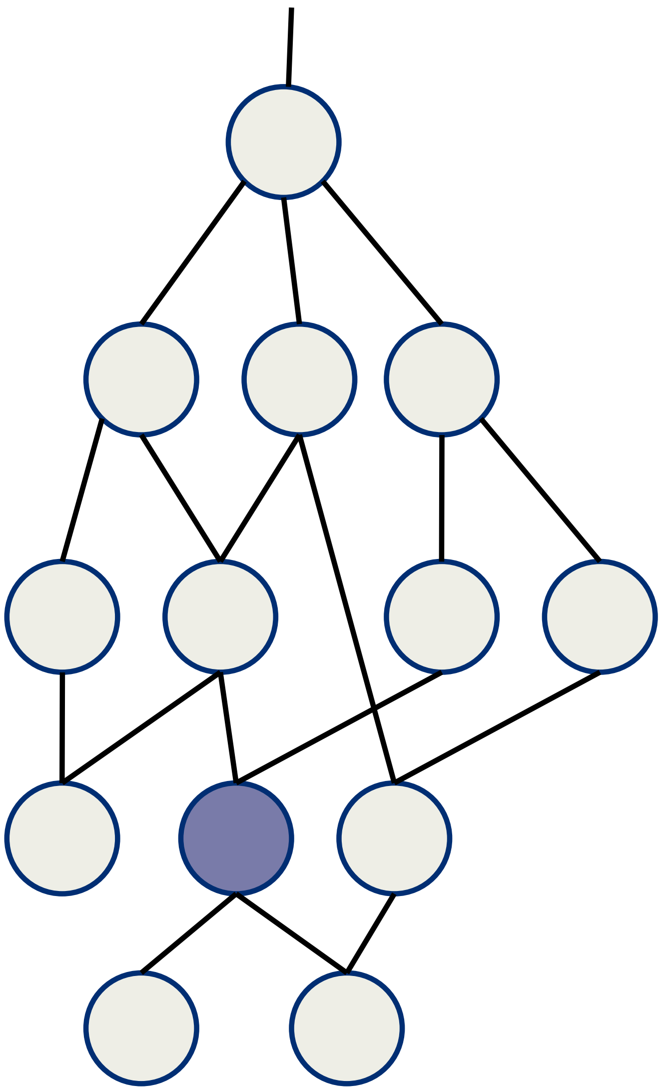
B

	a	**b**	**c**
a	-	0	1
b	-	-	1
c	-	-	-

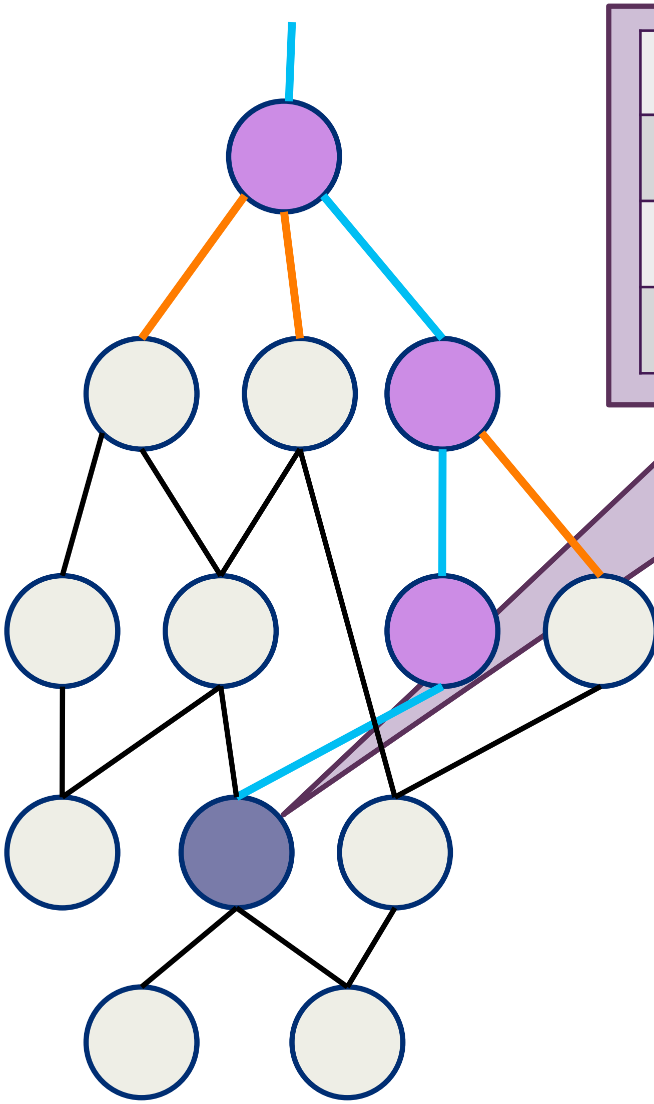
+ # Distinguishing Paths



Distinguishing Paths



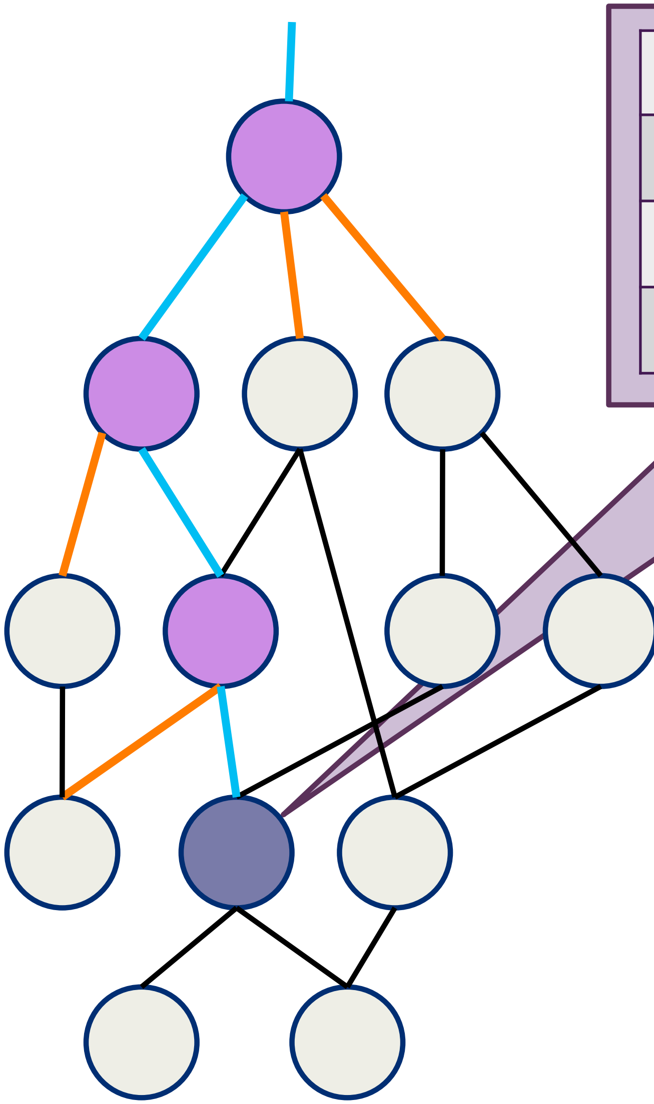
Distinguishing Paths



	a	b	c
a	-	1(u)	**1(u)**
b	-	-	1(u)
c	-	-	-

Notice that for transitively reduced circuits, no wires along a distinguishing path are side wires.

+ Distinguishing Paths



	a	b	c
a	-	**l(u)**	l(u)
b	-	-	l(u)
c	-	-	-

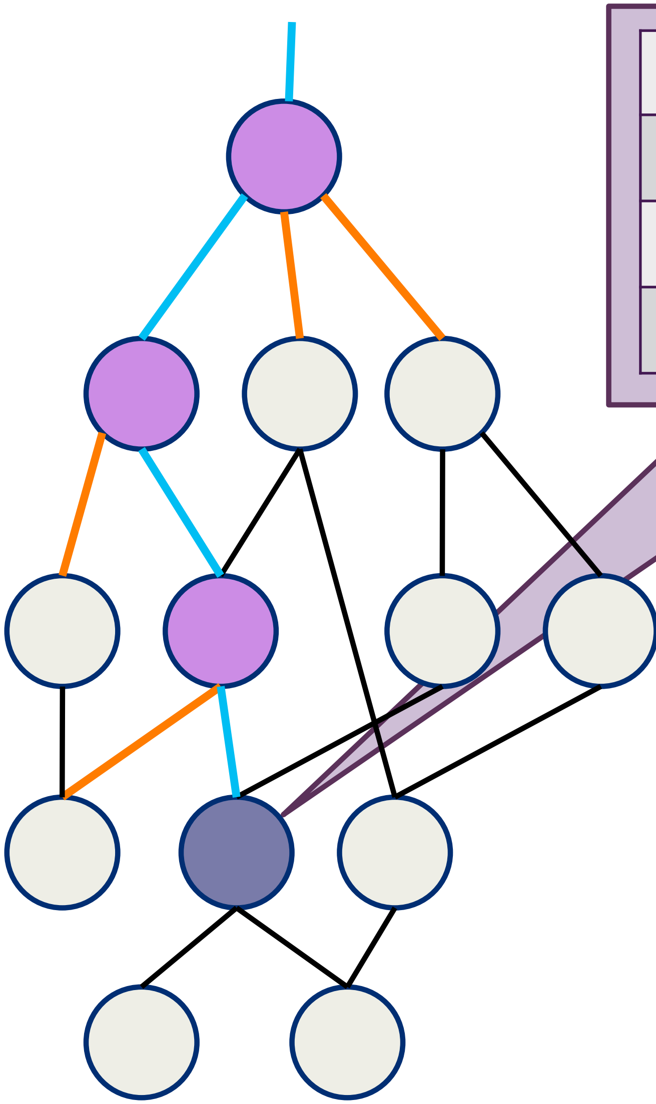
Notice that for transitively reduced circuits, no wires along a distinguishing path are side wires.

+

The Distinguishing Paths Algorithm (Outline)

- For the output wire w_n, we initialize T_{w_n} with all values initialized to 1, marked unprocessed. The rest of the tables are initialized to all 0's.
- While there are unprocessed 1 entries, pick one and run **Find Inputs** and **Extend Paths**.
- Finally, **Reconstruct the Circuit**.

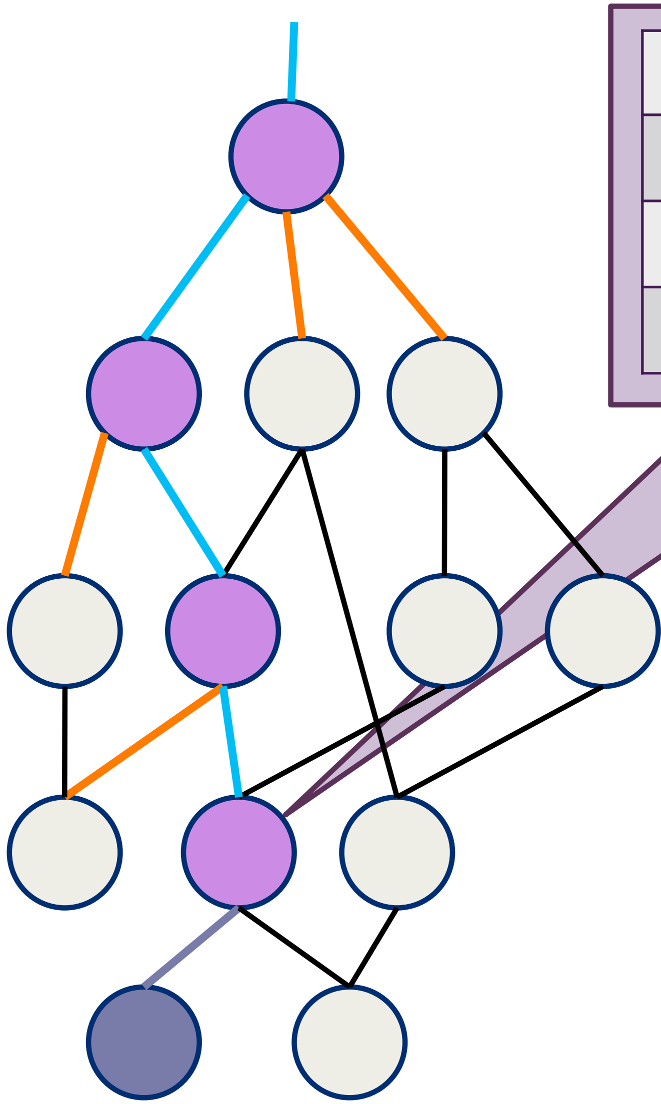
Find Inputs and Extend Paths



	a	b	c
a	-	**1(u)**	1(u)
b	-	-	1(u)
c	-	-	-

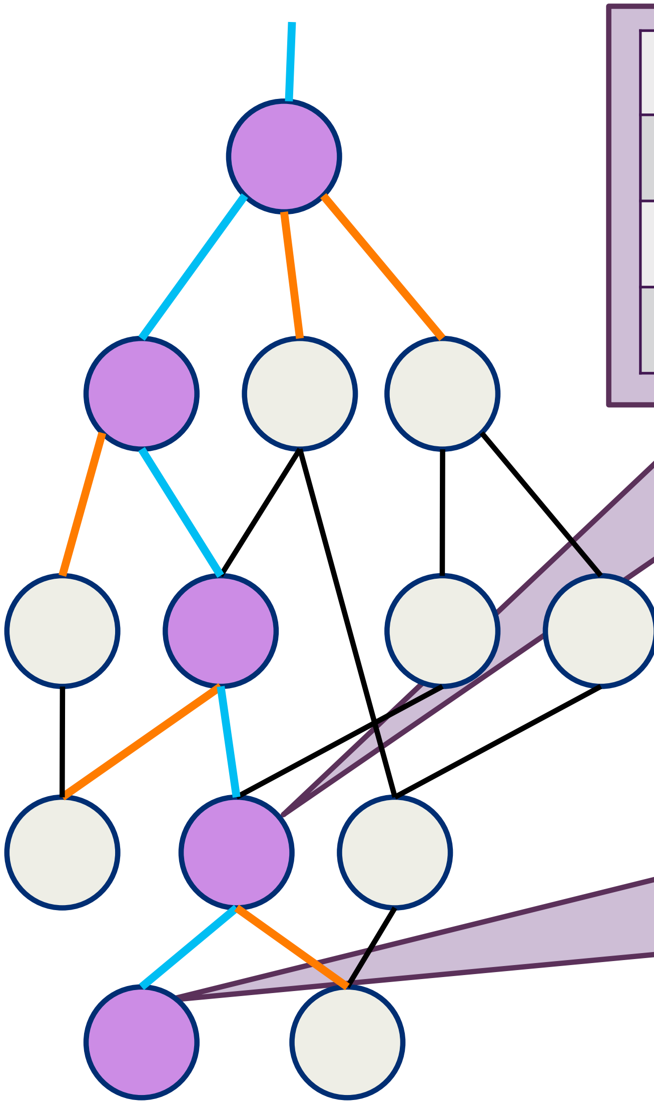
Now we want to use this distinguishing path to **Find** more **Inputs** and **Extend** the **Paths** to find new distinguishing paths.

Find Inputs and Extend Paths



	a	b	c
a	-	**l(u)**	l(u)
b	-	-	l(u)
c	-	-	-

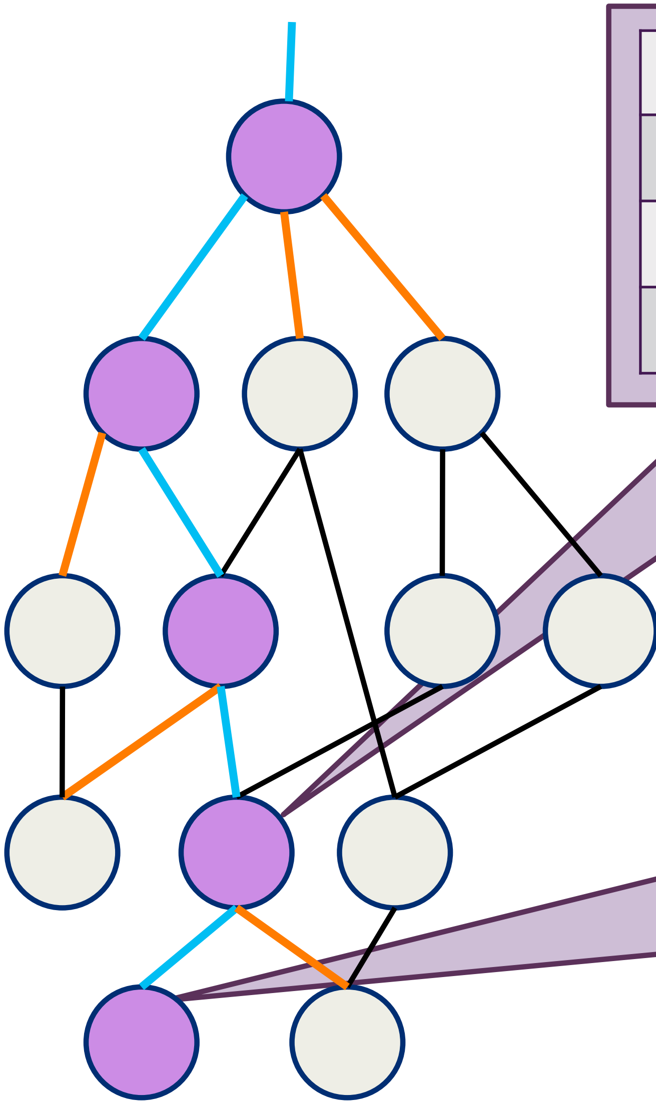
+ Find Inputs and Extend Paths



	a	b	c
a	-	**1(u)**	1(u)
b	-	-	1(u)
c	-	-	-

	a	b	c
a	-	1(u)	0
b	-	-	1(u)
c	-	-	-

+ Find Inputs and Extend Paths

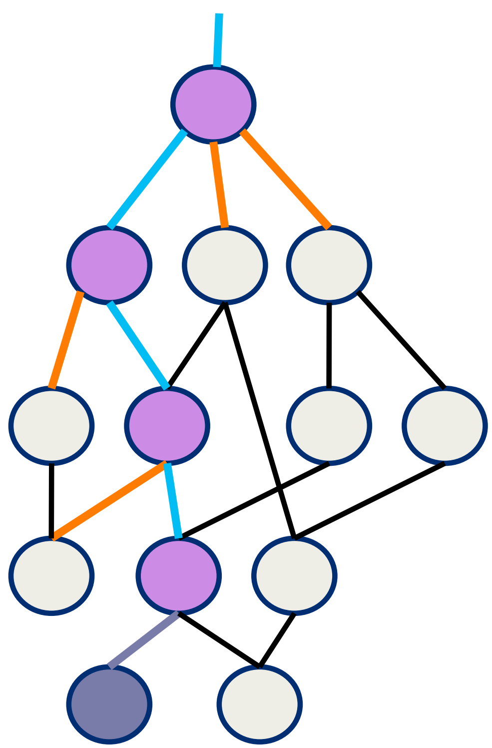


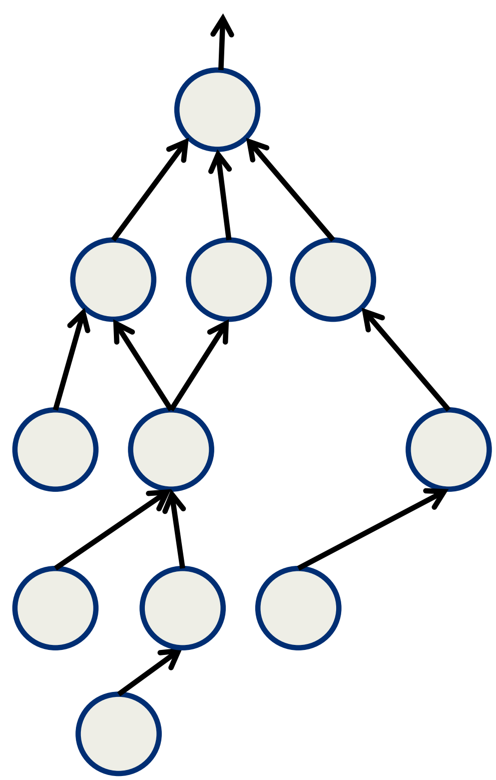
	a	b	c
a	-	1(p)	1(u)
b	-	-	1(u)
c	-	-	-

	a	b	c
a	-	1(u)	0
b	-	-	1(u)
c	-	-	-

Reconstructing Transitively Reduced Circuits

- We keep a separate directed graph G to reconstruct the graph of the circuit.





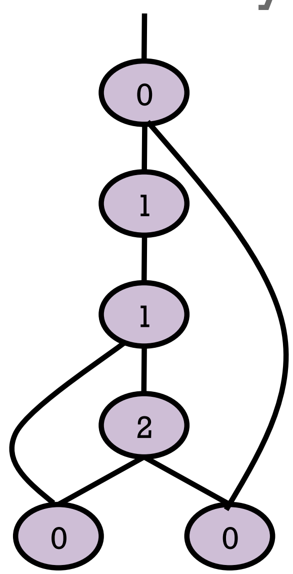
Reconstructing Transitively Reduced Circuits

- We keep a separate directed graph G to reconstruct the graph of the circuit.

- **Theorem** **The complete distinguishing tables and G are enough to construct a circuit behaviorally equivalent to the target circuit in polynomial time and $O(n^{2k+1}s^{2k+2})$ queries.**

Bounded Shortcut Width

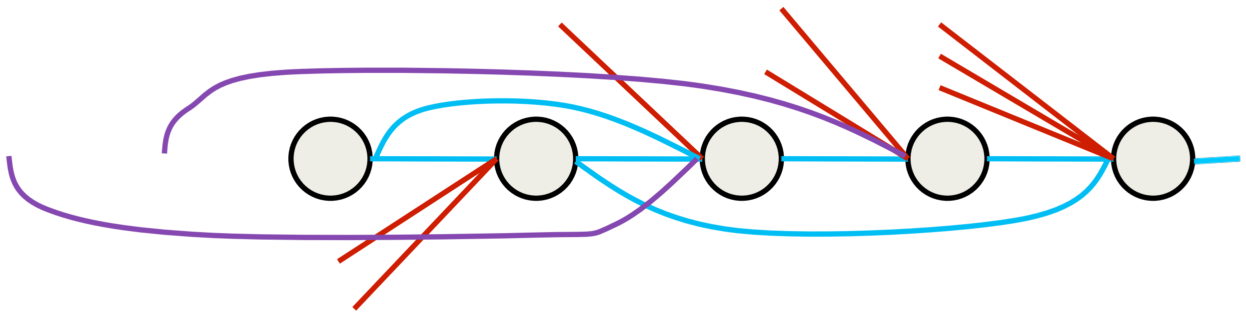
- Bounded shortcut width is a generalization of transitive reduction.
- The shortcut width of a wire w_i is the number of wires w_j such that w_j is both an ancestor of w_i and an input of a descendent of w_i.
  - Transitively reduced circuits have shortcut width 0.



The bounded shortcut width of a circuit is the maximum shortcut width of any output-connected wire in the circuit.

+ Distinguishing Paths with Shortcuts

- We generalize the definition of a distinguishing path to a distinguishing path with shortcuts.
  - These are made of path wires, side wires, and cut wires.



- We also generalize the notion of distinguishing tables to include cut wires.

+

Learning Circuits of Bounded Shortcut Width

- When all 1 entries in the generalized distinguishing tables are processed, the tables and graph G we can create a set of sufficient experiments for **CircuitBuilder** of [AACW '06].

- **Theorem** **The Shortcuts Algorithm 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.**

Learning Analog Circuits

- An analog circuit is a circuit for which $\Sigma = [0,1]$.
- ε-equivalence: If $d(C(e),C'(e)) \leq \varepsilon$ for every experiment e, then C and C' are ε-equivalent.
- We can discretize analog circuits that satisfy a Lipshitz condition and use our large-alphabet learning algorithms on them.
- **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.**

+

Part II
Graphical Models

work done with Dana Angluin, James Aspnes, David Eisenstat, and Jiang Chen

+

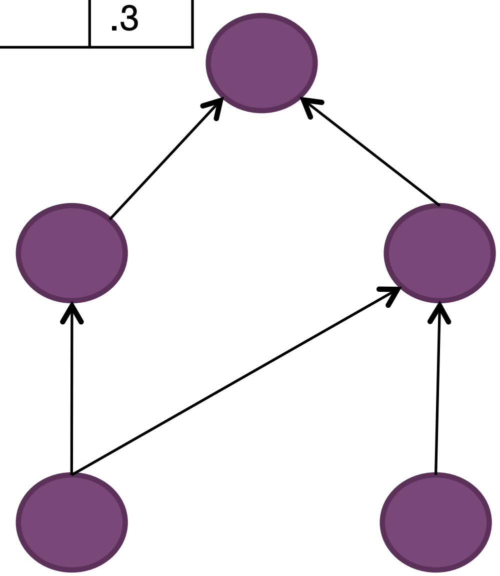
Part II
Graphical Models

aka Bayesian Networks and Probabilistic Circuits

work done with Dana Angluin, James Aspnes, David Eisenstat, and Jiang Chen

+ (Acyclic) Probabilistic Circuits

Input 1	Input 2	P(1)
0	0	.15
0	1	.5
1	0	1
1	1	.3



Input 1	Input 2	P(1)
0	0	.7
0	1	1
1	0	.5
1	1	0

Input	P(1)
0	.4
1	.8

P(1)
.9

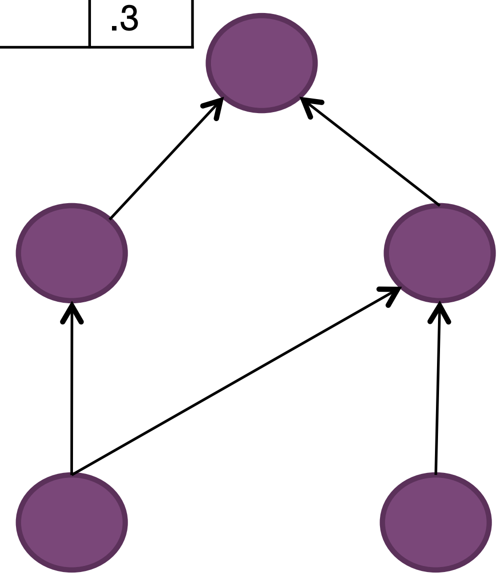
P(1)
.9

(Acyclic) Probabilistic Circuits

Input 1	Input 2	P(1)
0	0	.15
0	1	.5
1	0	1
1	1	.3

Input 1	Input 2	P(1)
0	0	.7
0	1	1
1	0	.5
1	1	0

Input	P(1)
0	.4
1	.8



P(1)
.9

P(1)
.9

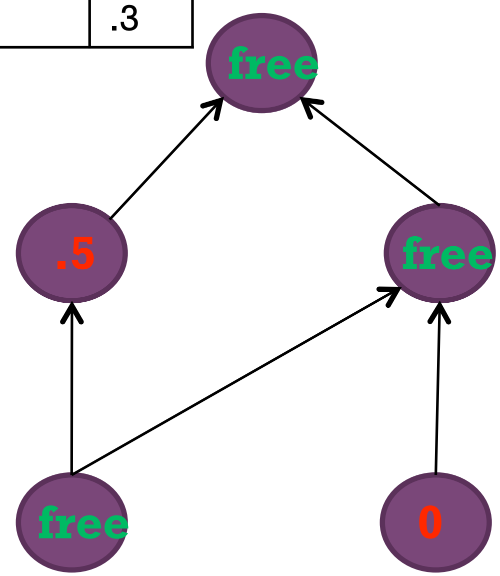
VIQs on Probabilistic Circuits

Exact VIQs

+ (Acyclic) Probabilistic Circuits

Input 1	Input 2	P(1)
0	0	.15
0	1	.5
1	0	1
1	1	.3

Inpu t	P(1)
~~0~~	~~.4~~
~~1~~	~~.8~~



Input 1	Input 2	P(1)
0	0	.7
0	1	1
1	0	.5
1	1	0

P(1)
.9

~~P(1)~~
~~0~~

VIQs on Probabilistic Circuits

Exact VIQs

The Learning Problems

- **ε-Approximate Learning**
  - **ε-behavioral equivalence:** Circuits C and C' are ε-behaviorally equivalent if for any experiment s, $d(C(s)-C'(s)) < \varepsilon$.
    - $d(C(s)-C'(s))$ is a notion of statistical distance
  - **The problem:** Given query access to a hidden circuit C^*, find a circuit C ε-behaviorally equivalent to C^* by making value-injection queries.
- **Exact Learning**
  - **behavioral equivalence:** Two circuits C and C' are behaviorally equivalent if for any experiment s, $C(s)=C'(s)$.
  - **The problem:** Given query access to a hidden circuit C^*, find a circuit C behaviorally equivalent to C^* by making exact value-injection queries.

Previous Work

Circuit	Fan-in	Topology	Gates	VIQ Learnability
Boolean	2	arbitrary	AND/OR	NP-Hard
Boolean	unbounded	constant depth	AND/OR/ Θ_2	NP-Hard
Boolean	constant	log depth	arbitrary	Poly-time
Large $\sum$	constant	log depth	arbitrary	W(1) Hard in shortcut width
Large $\sum$	constant	bounded shortcut width	arbitrary	Poly-time
Analog	constant	bounded shortcut width	arbitrary	Poly-time approximate

Main Results on Probabilistic Circuits

- The Test Path Lemma
- Boolean Probabilistic Circuits
  - Approximately Learnable
- Larger Alphabet Probabilistic Circuits
  - Not Learnable Using Test Paths
  - Learnable with Function Injection Queries

Main Results on Probabilistic Circuits

- The Test Path Lemma
- Boolean Probabilistic Circuits
  - Approximately Learnable
- Larger Alphabet Probabilistic Circuits
  - Not Learnable Using Test Paths
  - Learnable with Function Injection Queries

If nothing else, I want to show you how probabilistic circuits behave differently than you might expect

The Test Path Lemma

- A test path for a wire w is a value injection experiment in which the free gates form a directed path in the circuit graph from w to the output wire. All the other wires in the circuit are fixed, including the inputs of w.

- The test path lemma: Let C be a deterministic circuit.

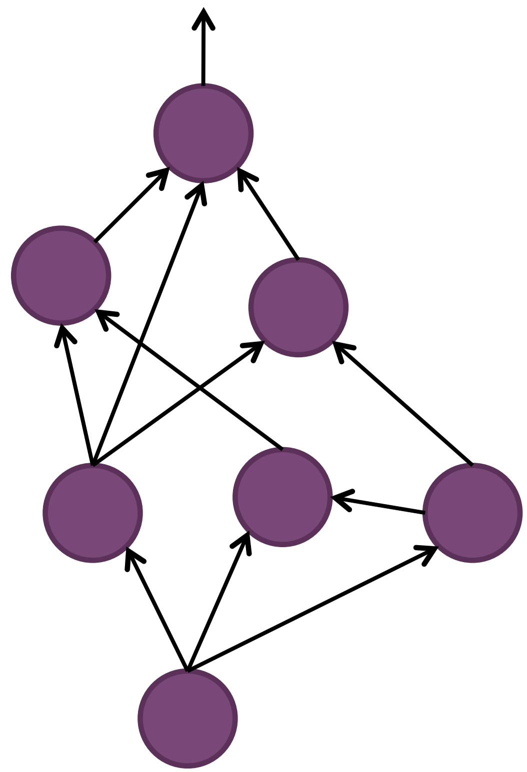
  If for some value injection experiment e, wire w and alphabet symbols σ and τ it is the case that

  $C(p|_{w=\sigma}) = C(p|_{w=\tau})$

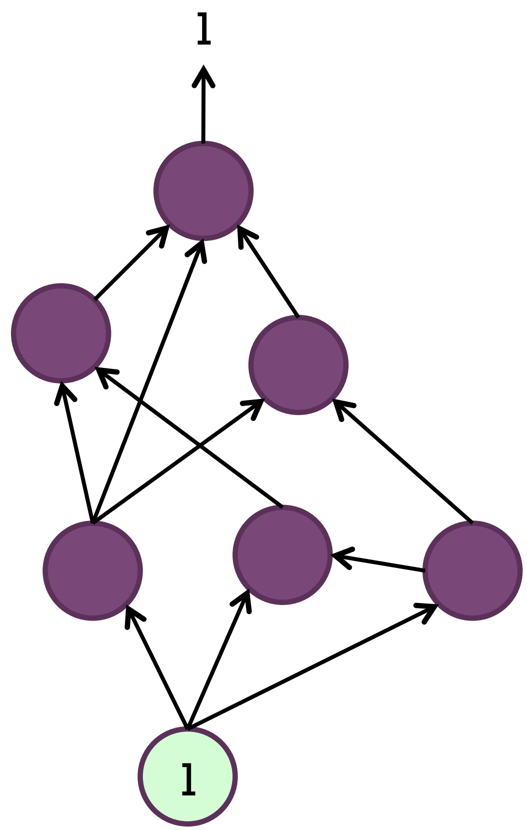
  Then for every test path p < e, then also

  $C(e|_{w=\sigma}) = C(e|_{w=\tau})$.

+ Test Path Lemma Illustrated

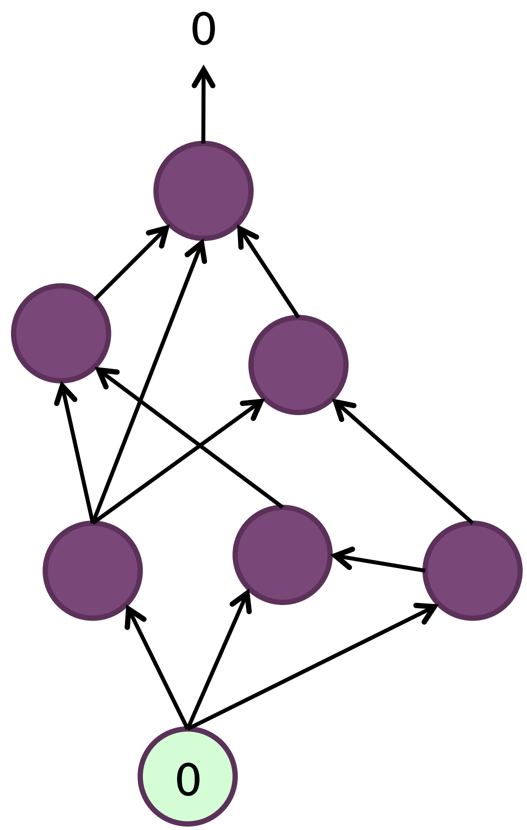


Test Path Lemma Illustrated

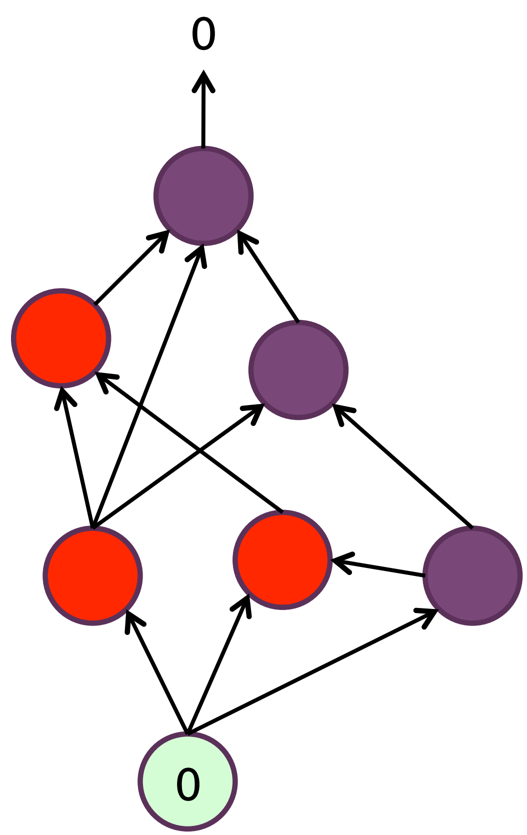


+

Test Path Lemma Illustrated

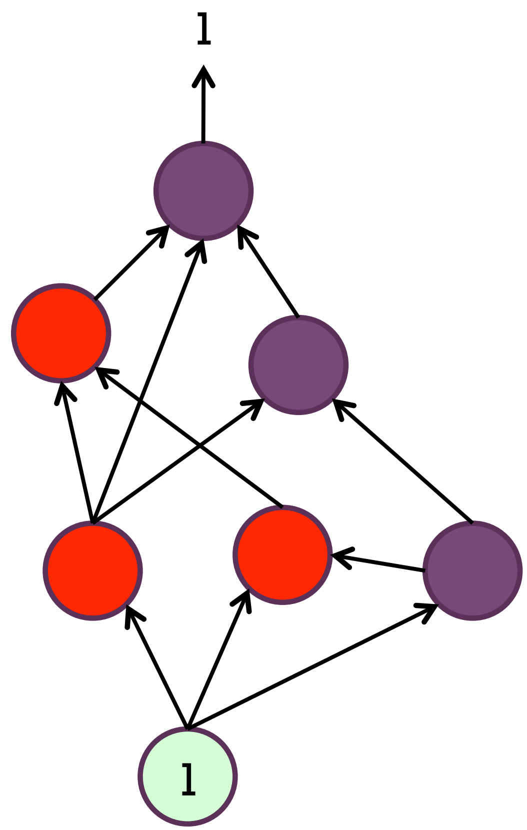


Test Path Lemma Illustrated



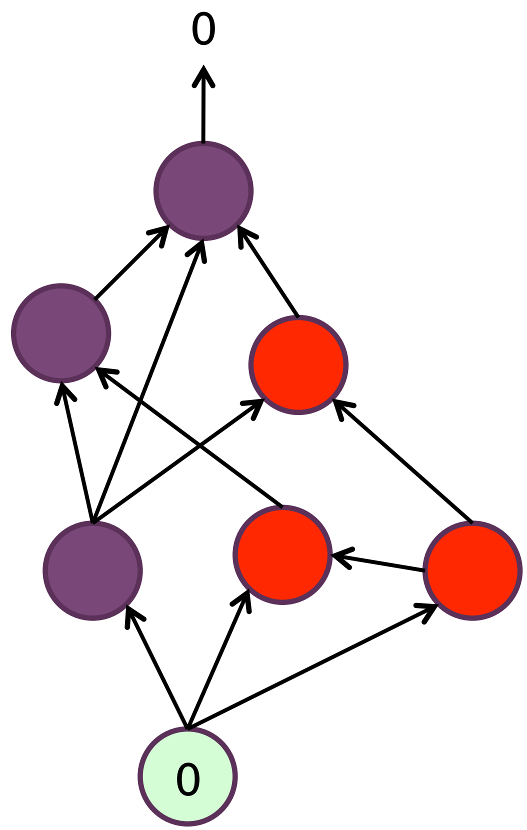
+

Test Path Lemma Illustrated

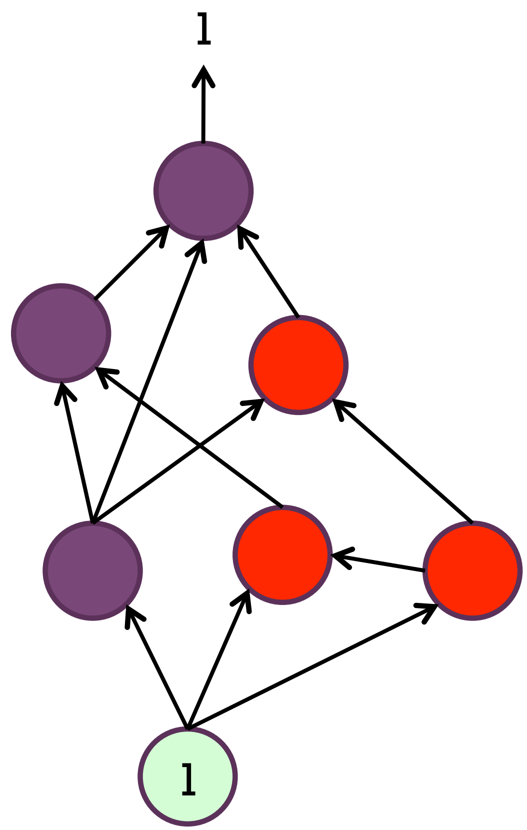


+

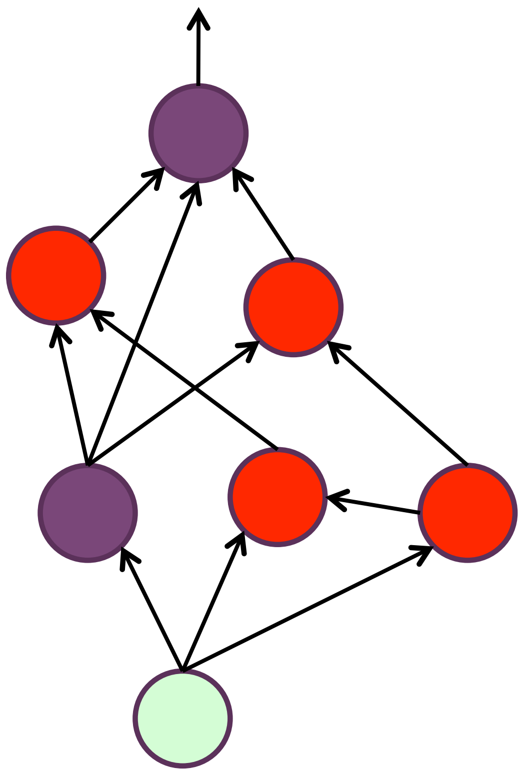
Test Path Lemma Illustrated



Test Path Lemma Illustrated

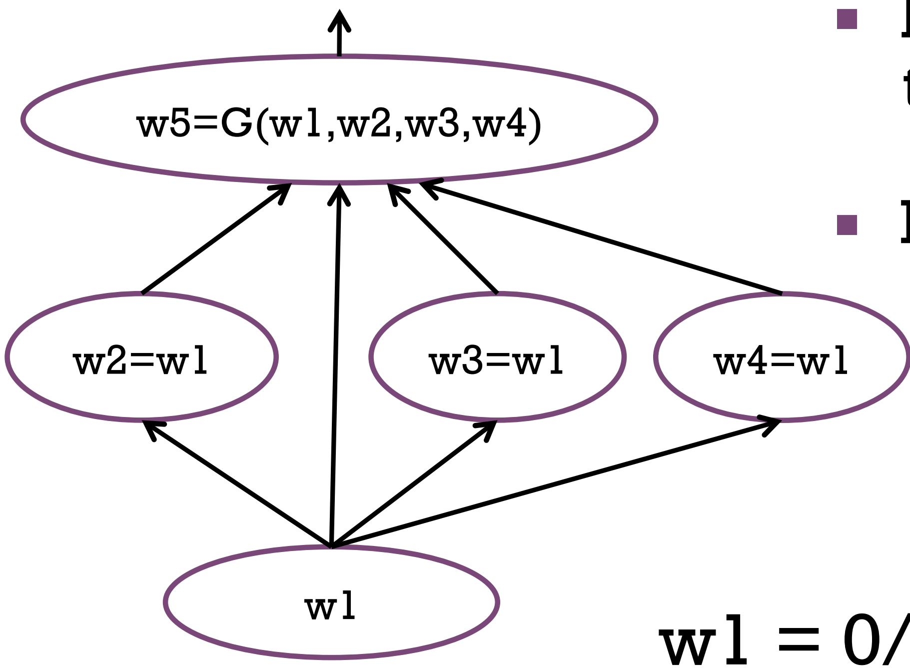


Test Path Lemma Illustrated



+ Attenuation of Signal in Test Paths

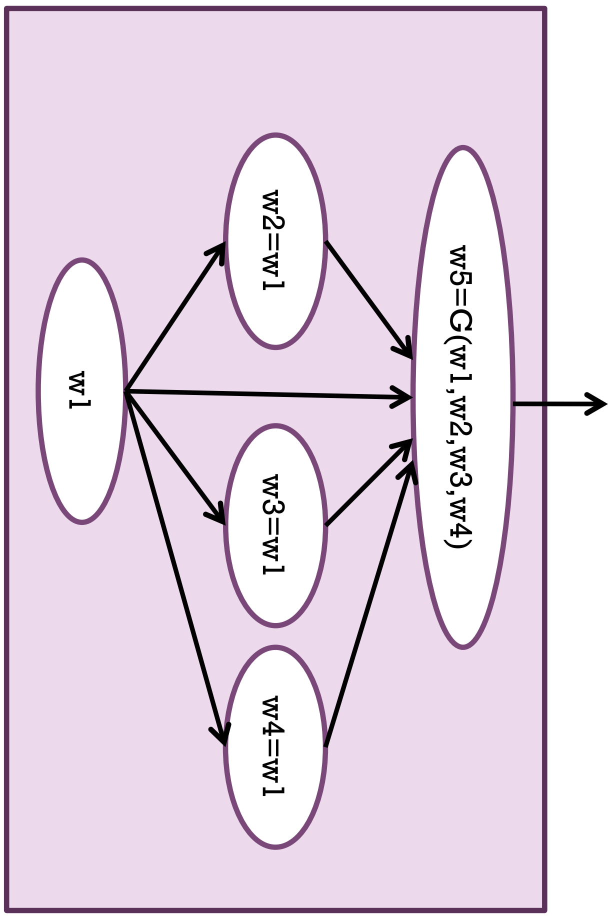
Let $G(w1,w2,w3,w4) = ((1-w1)+2w2+2w3+2w4)/7$



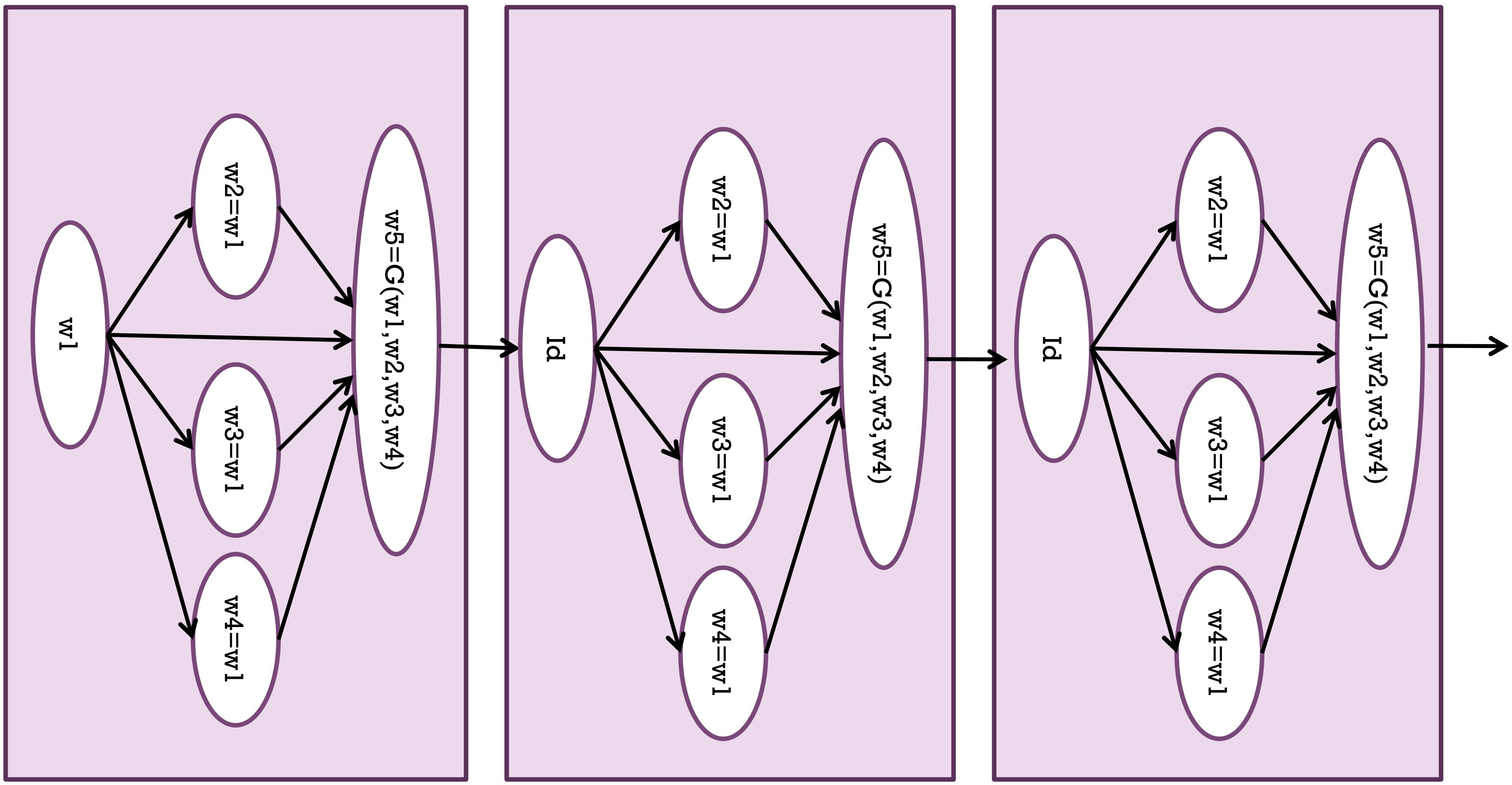
- If e sets all wires to be free, then
  $d(D1(e|_{w=0}),D1(e|_{w=1})) = 5/7.$
- But for any test path p for w1
  $d(D1(p|_{w=0}),D1(p|_{w=1})) = 1/7.$

w1 = 0/1

Exponential Attenuation



+ Exponential Attenuation



+ Boolean Probabilistic Circuits

But we still have (attenuated) test paths

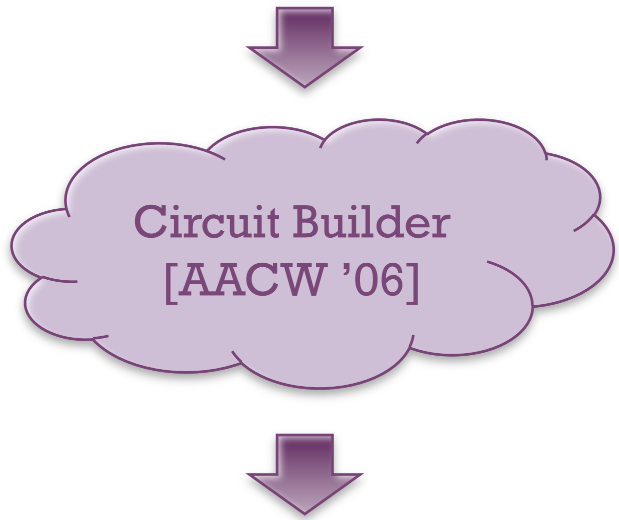
Some magic happens

There is a nonadaptive learning algorithm that with probability at least $(1 - \delta)$ ε-approximately learns any Boolean probabilistic circuit w/ n wires, constant fan-in and depth c log n using value injection queries in time bounded by a **polynomial in n, 1/ ε and log(1/ δ).**

+

Boolean Probabilistic Circuits

But we still have (attenuated) test paths



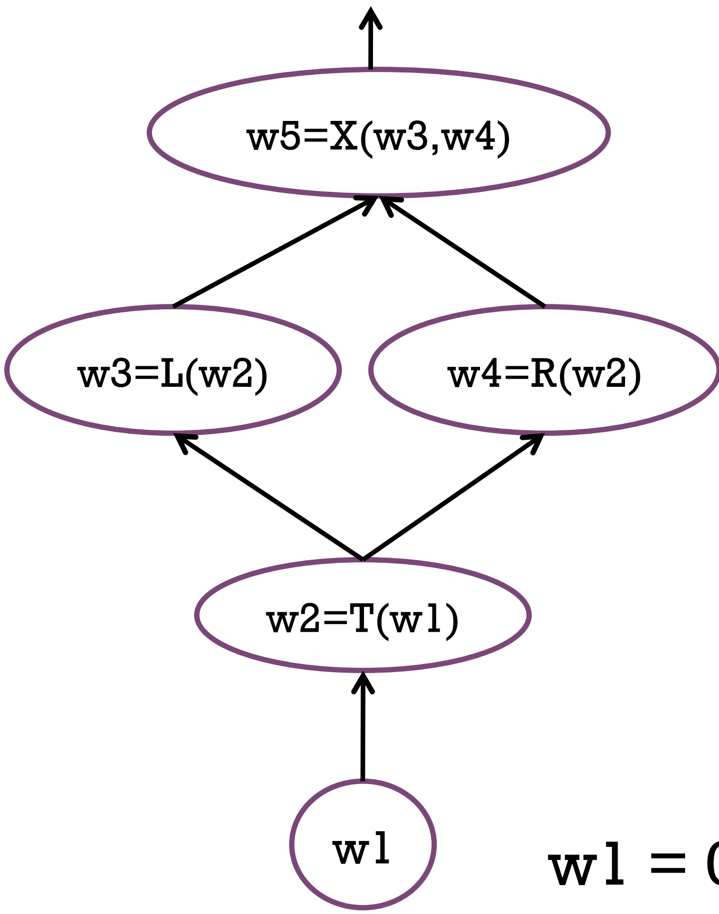
There is a nonadaptive learning algorithm that with probability at least $(1 - \delta)$ ε-approximately learns any Boolean probabilistic circuit w/ n wires, constant fan-in and depth c log n using value injection queries in time bounded by a **polynomial in n, $1/\varepsilon$ and $\log(1/\delta)$.**

Larger Alphabet Probabilistic Circuits

- Lets consider probabilistic circuits that have gates that operate on more than two alphabet symbols.

- What happens to the test path lemma in the large alphabet, probabilistic case?

Test Paths Fail (Completely) for |∑|>2



T(00)=T(11) =	U({00,11})
T(01)=T(10) =	U({01,10})
L(00)=L(01)=	00
L(10)=L(11)=	01
R(00)=R(10)=	00
R(01)=R(11)=	01
X(ab,cd)=	0(b⊗d)

w1 = 00 / 01

Function Injection Queries

- An **alphabet transformation** is a function f that maps symbols to distribution over symbols.

- A **function injection experiment** is a mapping that for each wire either leaves it free, assigns it an alphabet symbol, or assigns a transformation f.

- A **function injection query** (**FIQ**) takes a function injection experiment and returns the symbol assigned to the output wire.

+ Learning Large Alphabet Circuits

- A 2-partition experiment is a function injection experiment in which every alphabet transformation is a 2-partition.
- By using 2-partition experiments, we can "smash" the large alphabet circuits back to the Boolean case.
  - We get same positive learnability results for probabilistic large alphabet circuits using FIQs as we have for probabilistic Boolean circuits using VIQs.

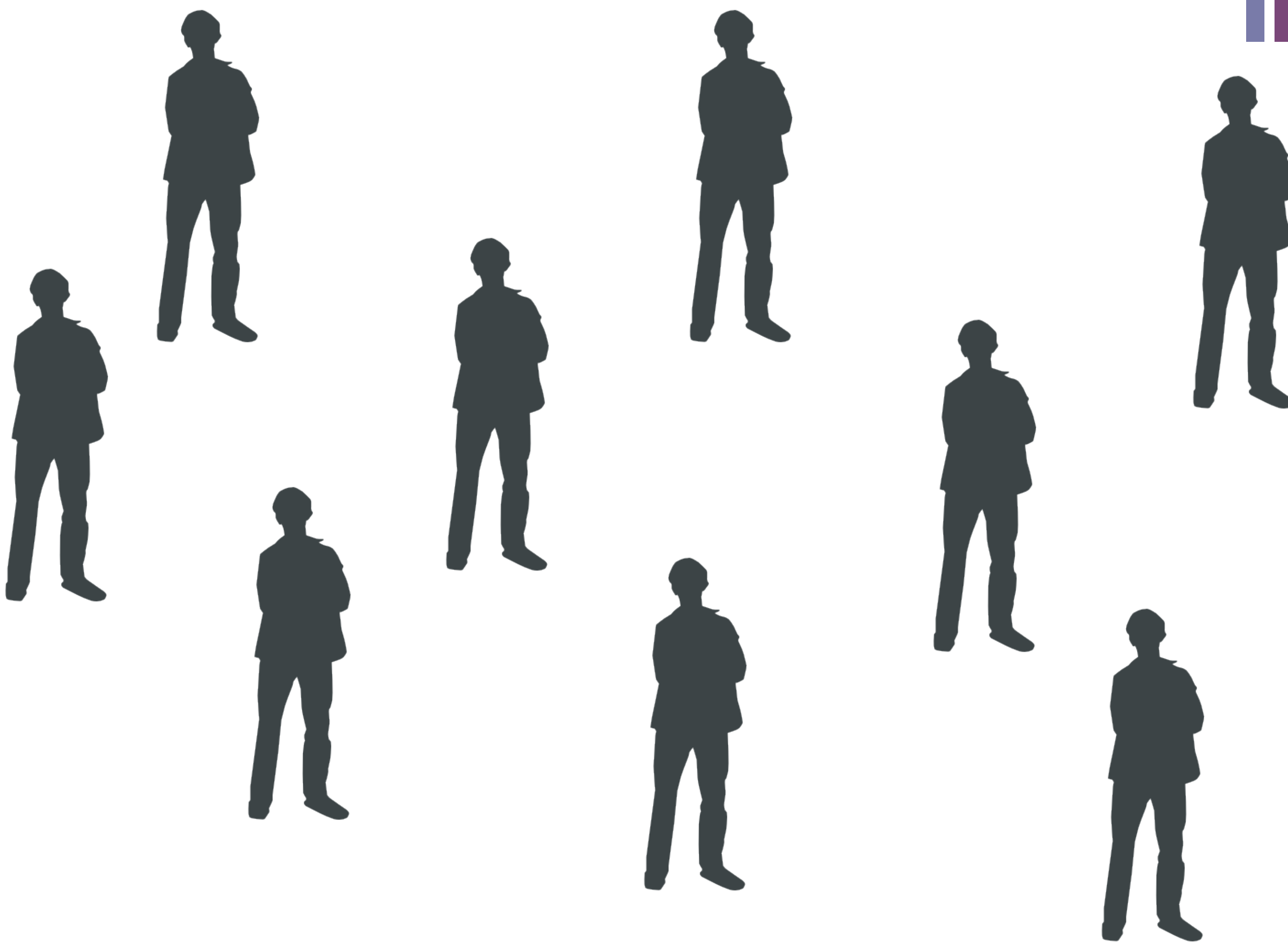
+ Results Table

Circuit	Fan-in	Topology	Gates	VIQ Learnability
Boolean	2	arbitrary	AND/OR	NP-Hard
Boolean	unbounded	const depth	AND/OR/Θ_2	NP-Hard
Boolean	constant	log depth	arbitrary	Poly-time
Large ∑	constant	log depth	arbitrary	W(1) Hard in shortcut width
Large ∑	constant	Bounded sc width	arbitrary	Poly-time
Analog	constant	bounded sc width	arbitrary	Poly-time approximate
Probabilistic Boolean	constant	log depth	arbitrary	Poly-time approximate
Probabilistic Large ∑	constant	log depth	arbitrary	Poly-time w/ FIQs
Probabilistic cyclic!	Unbounded	arbitrary	independent cascade	Poly-time w/ exact VIQs

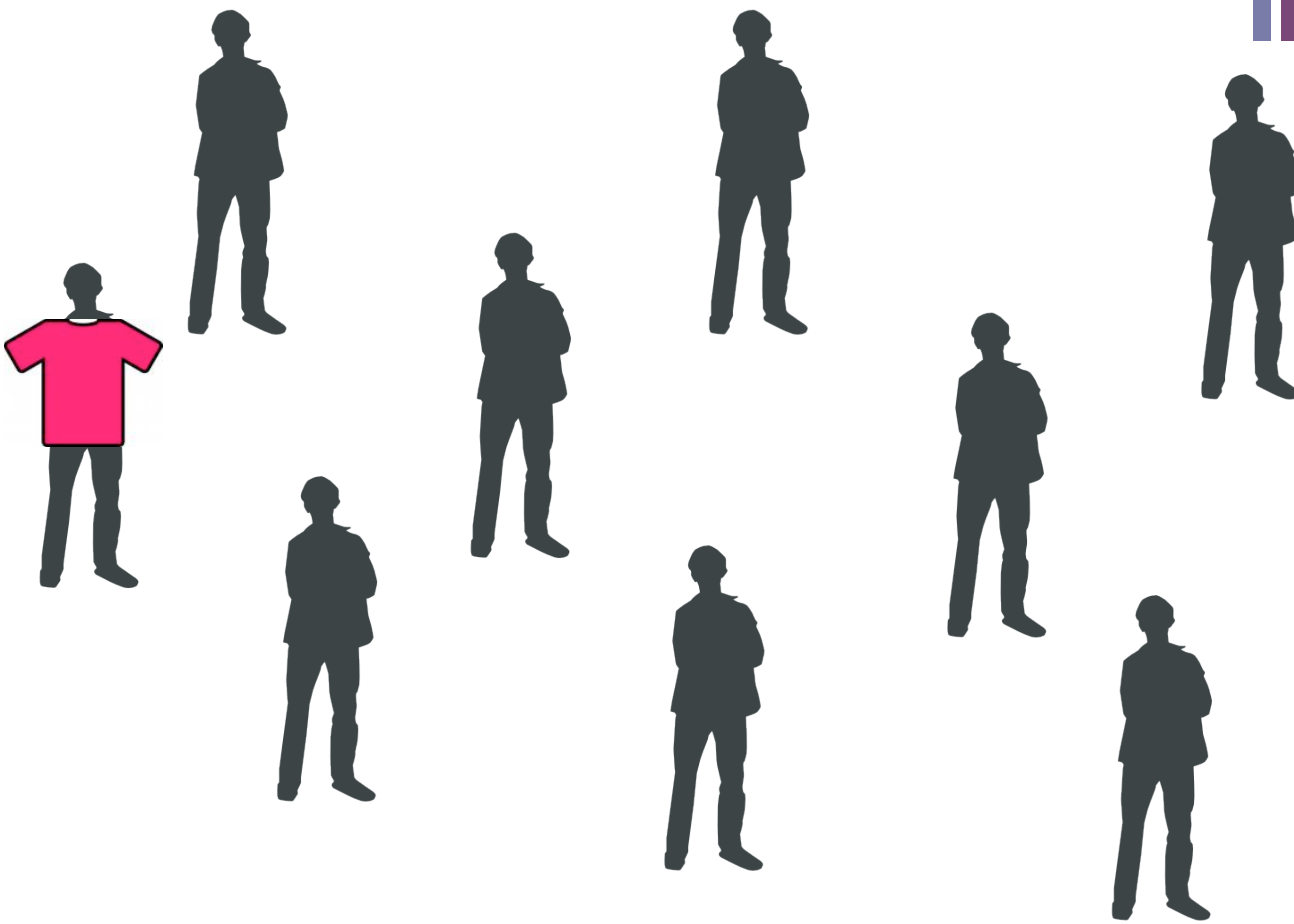
Part III
Social Networks

work done with Dana Angluin and James Aspnes

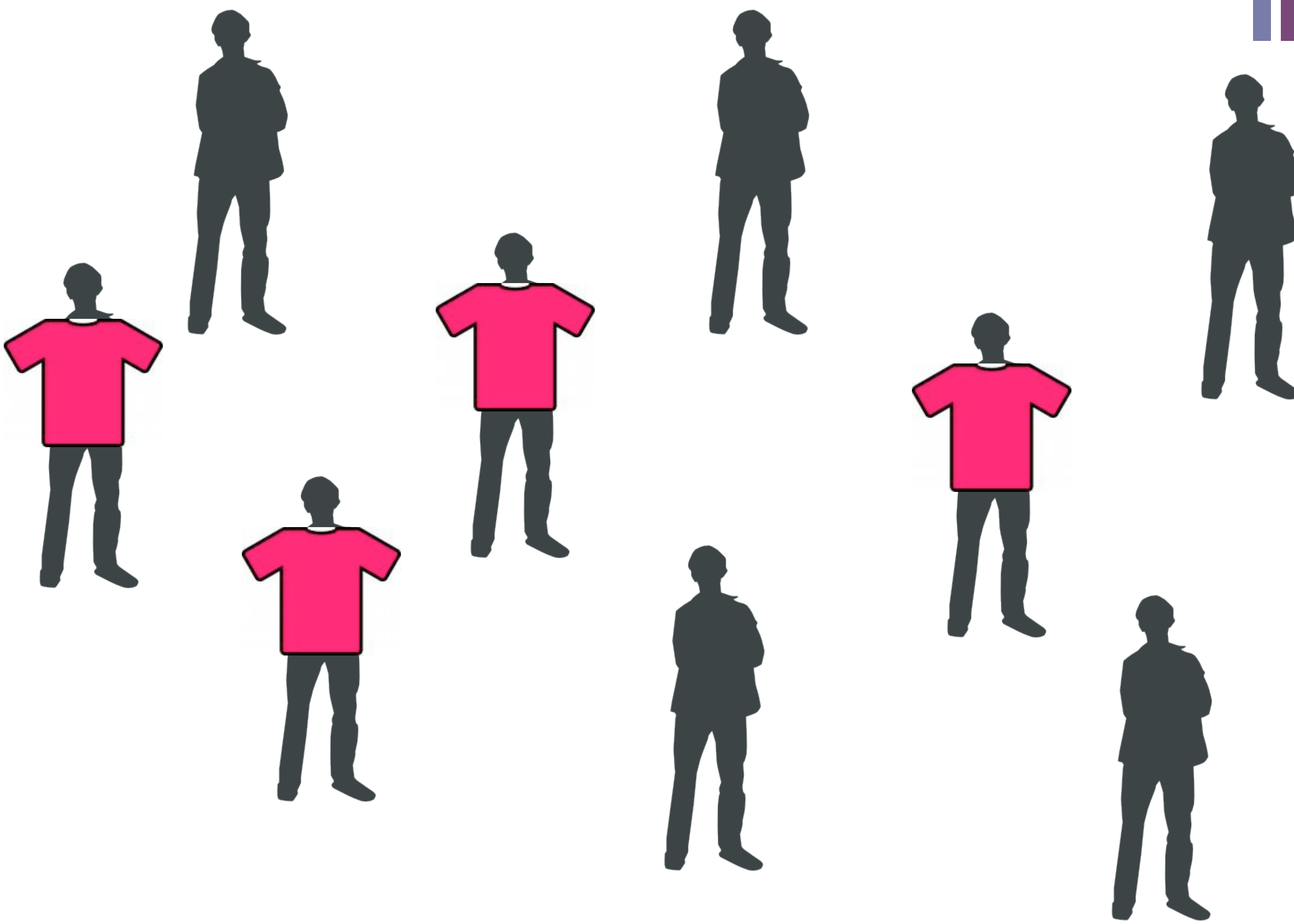
+ Trends Spreading through a Social Network



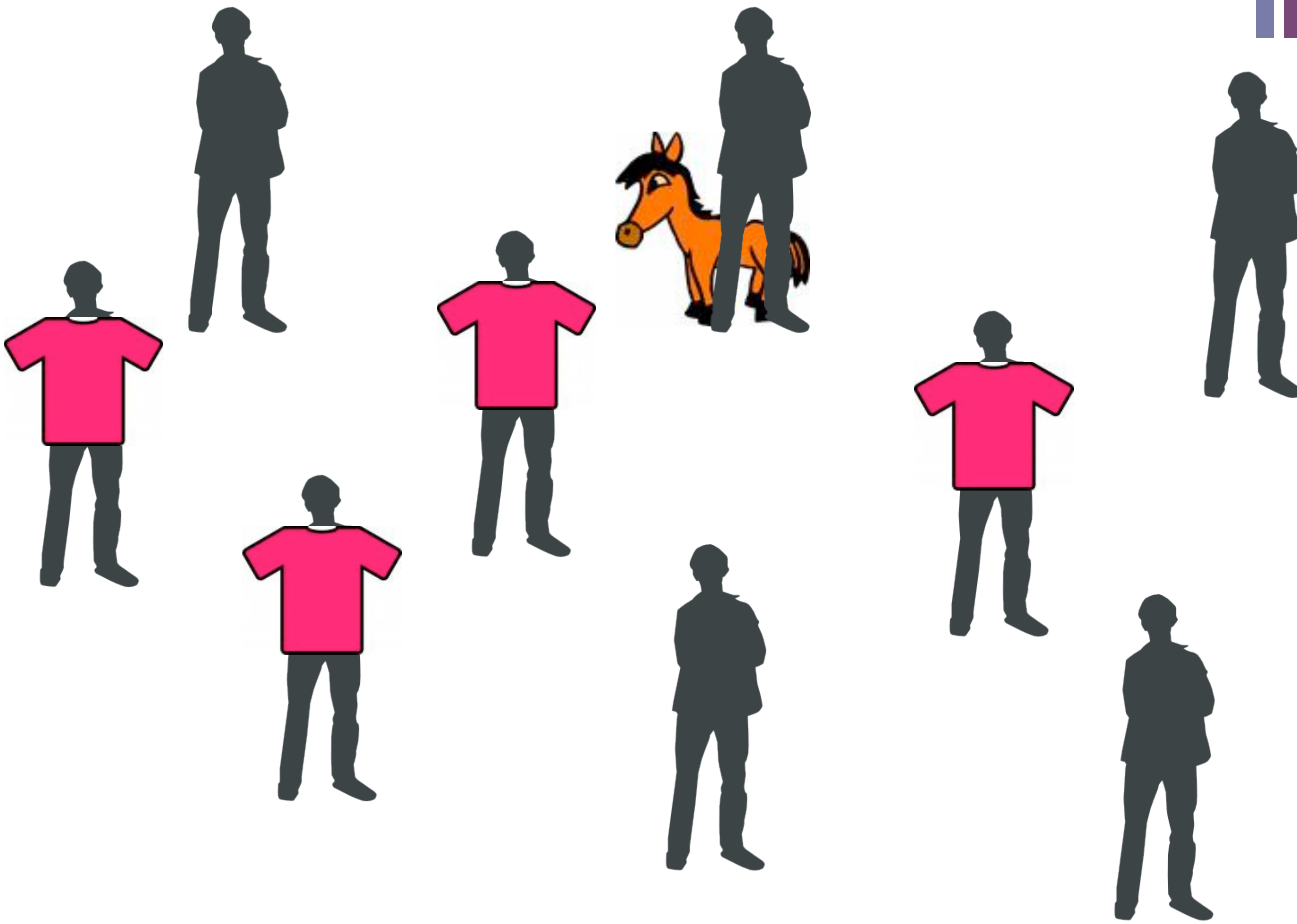
Trends Spreading through a Social Network



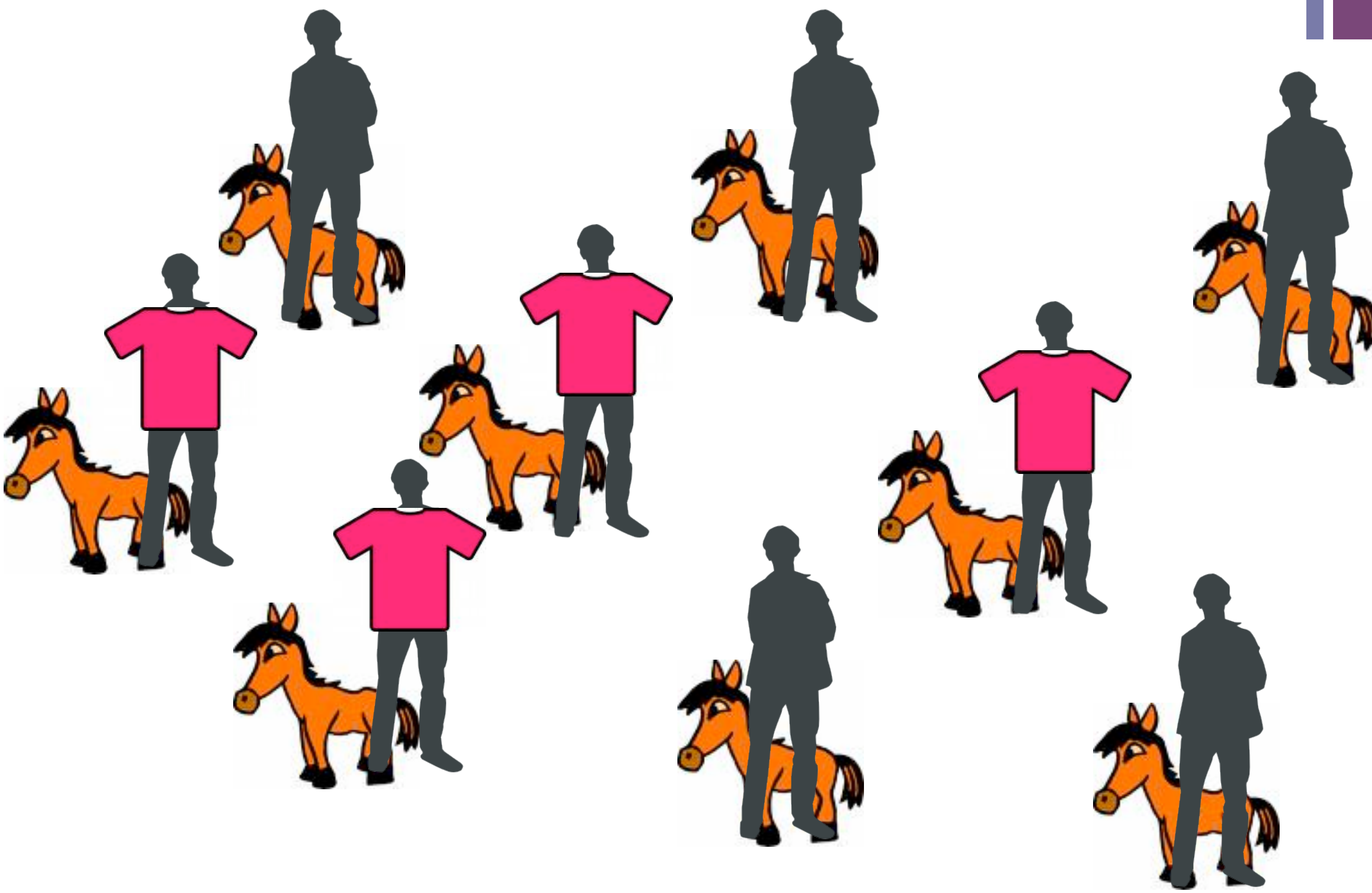
Trends Spreading through a Social Network



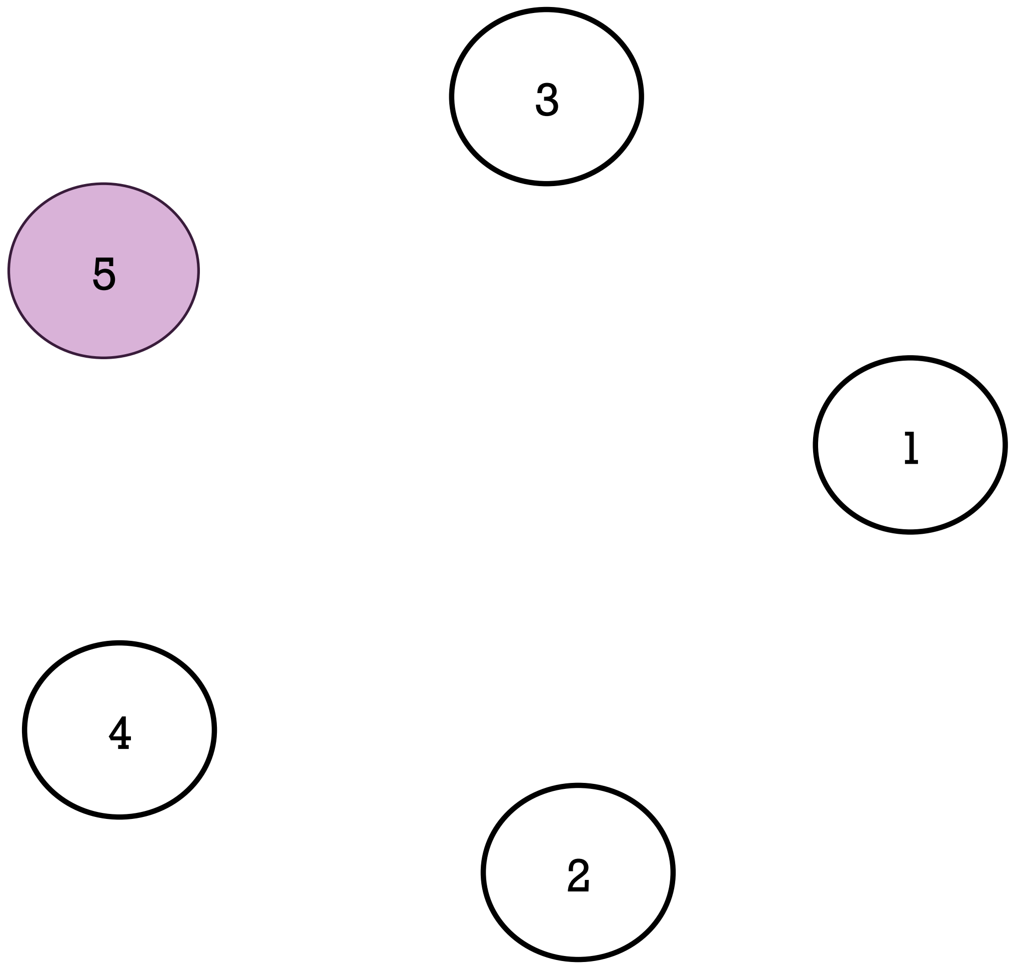
Trends Spreading through a Social Network



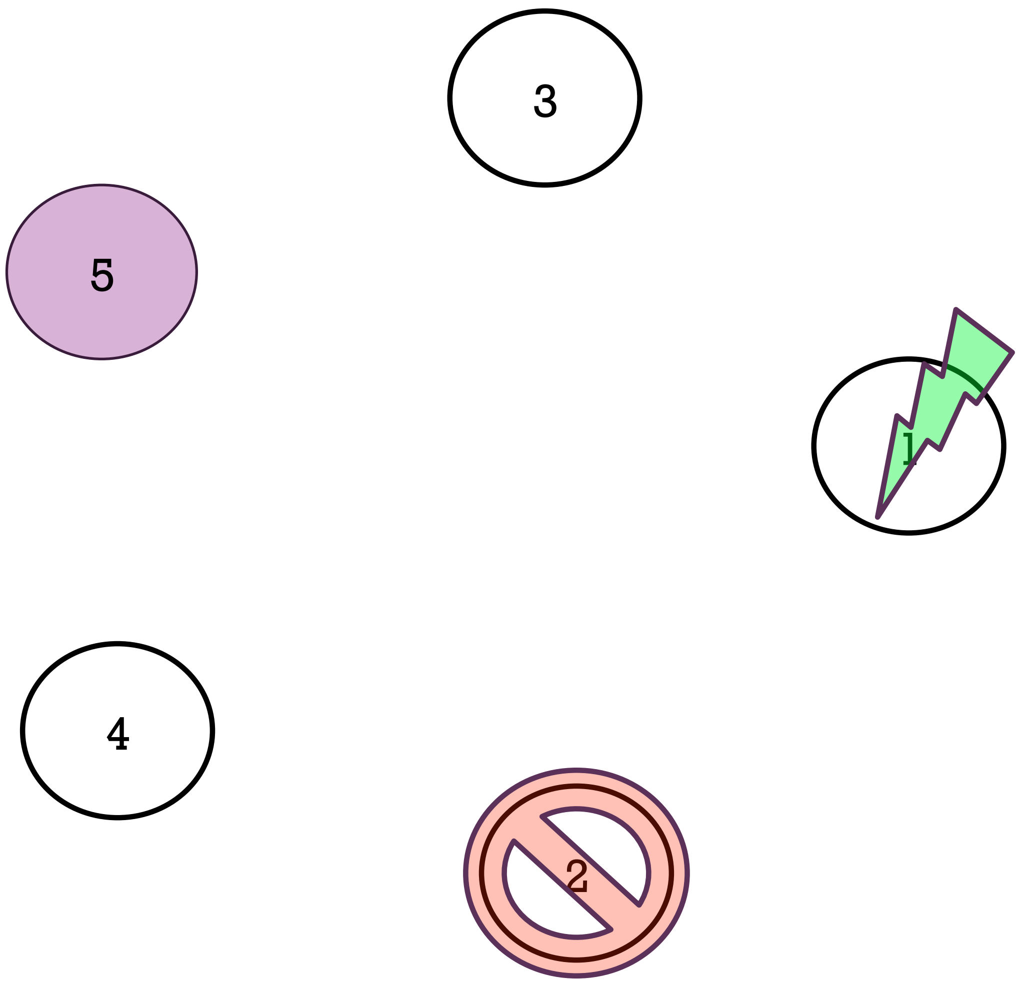
+ # Trends Spreading through a Social Network



What the Learner Sees

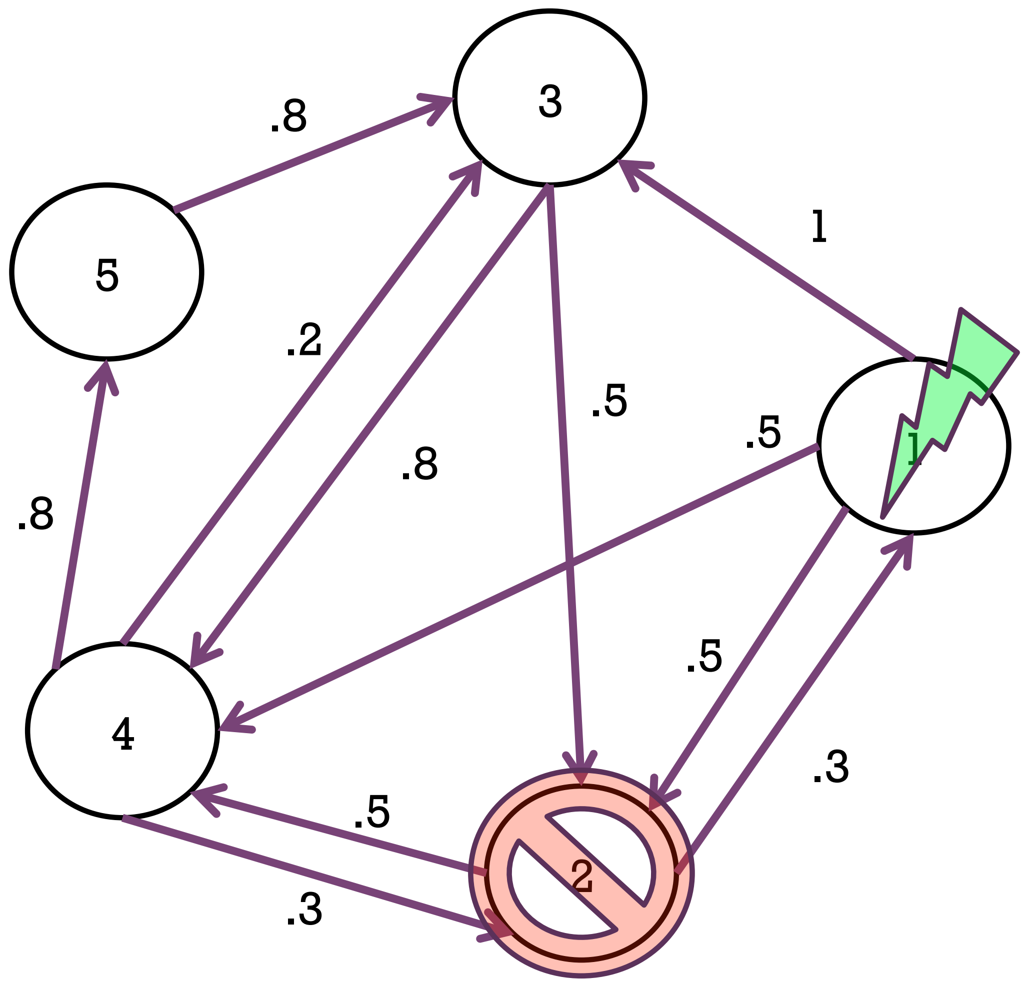


Activations and Suppressions

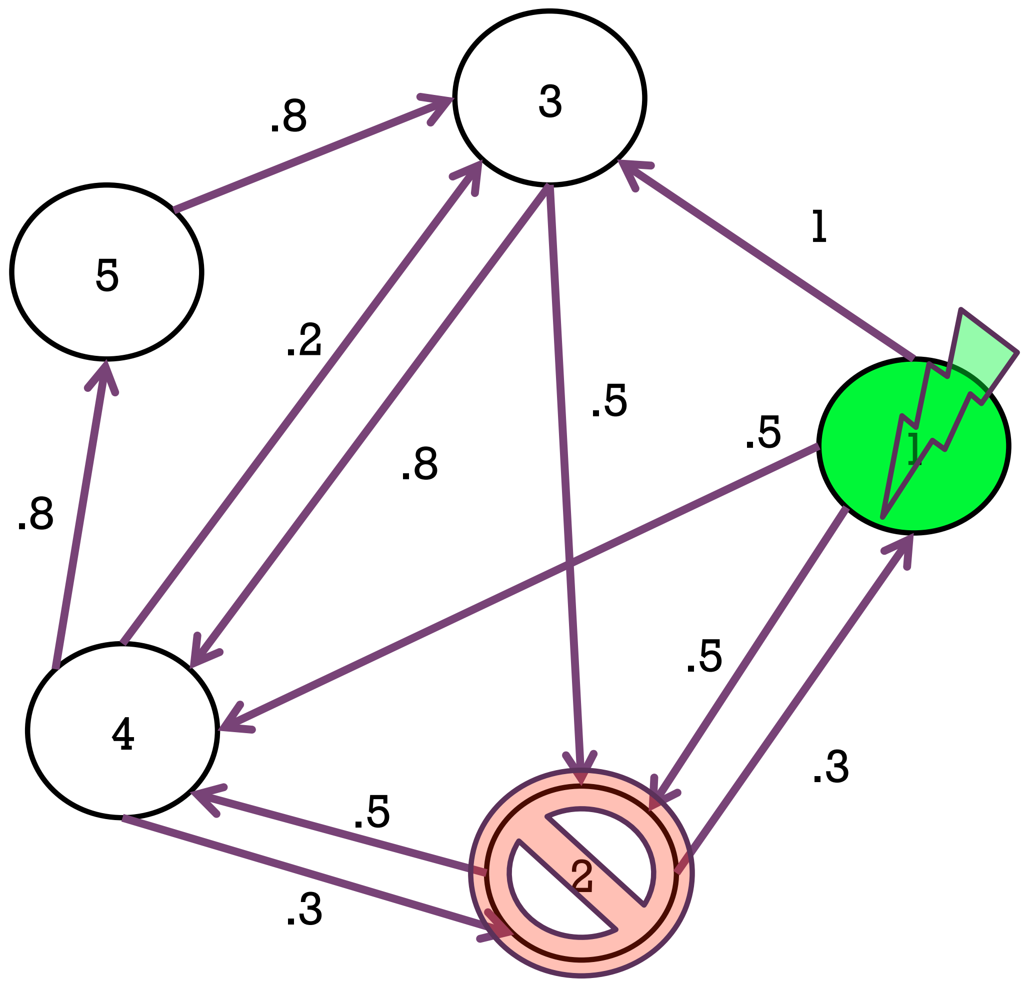


+

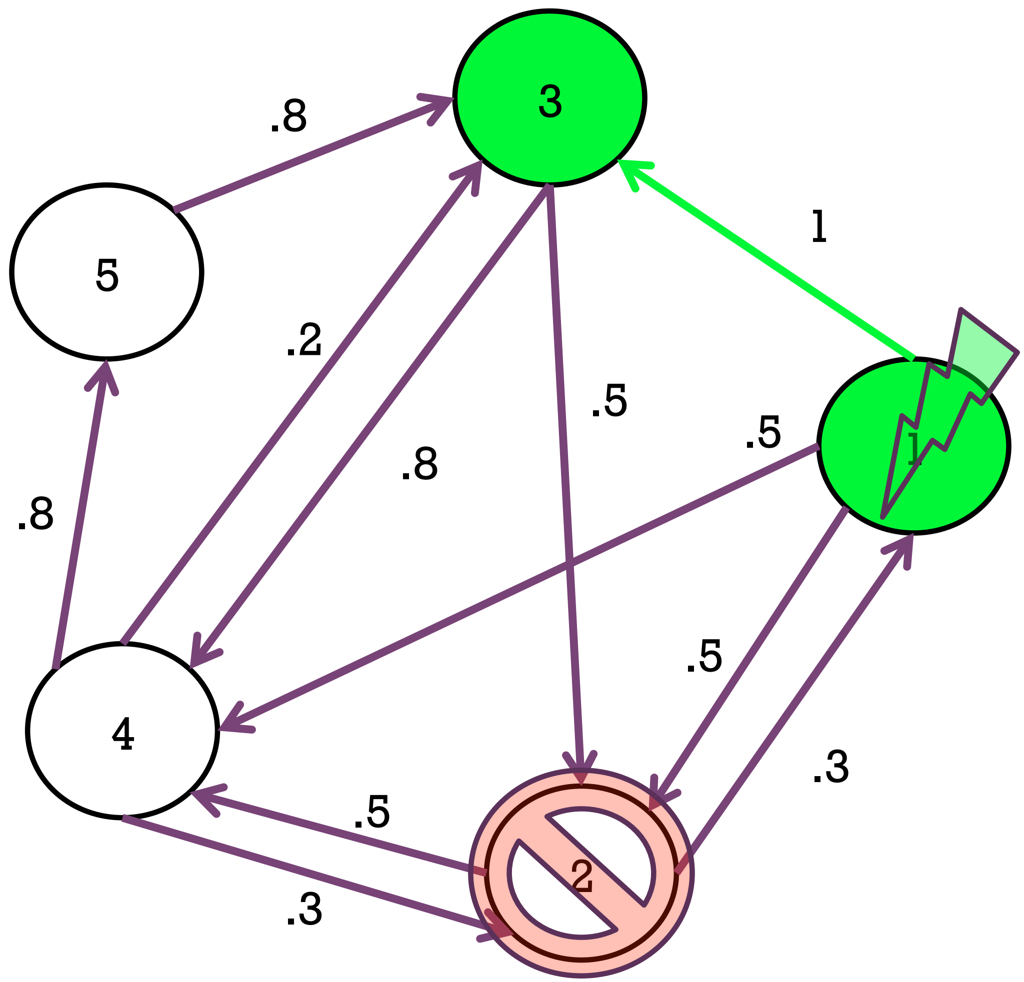
Activations and Suppressions



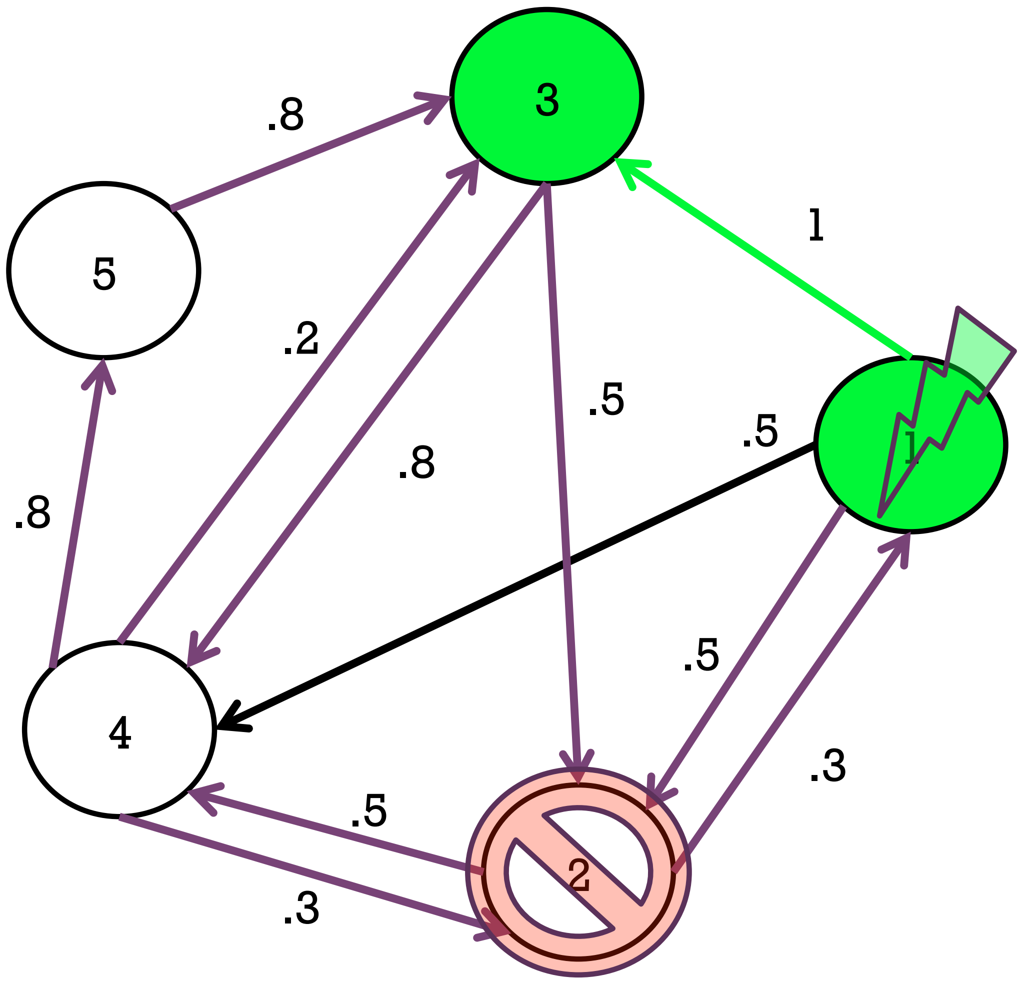
Activations and Suppressions



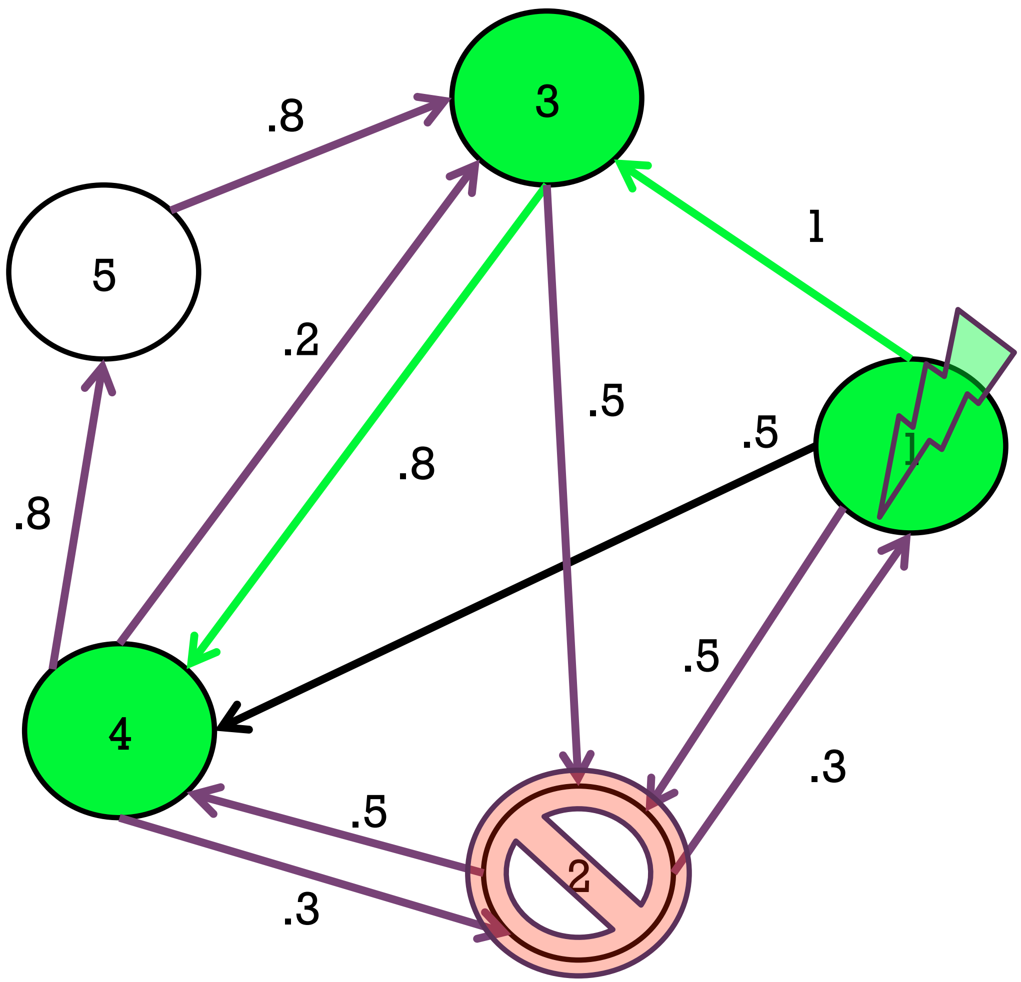
Activations and Suppressions



Activations and Suppressions

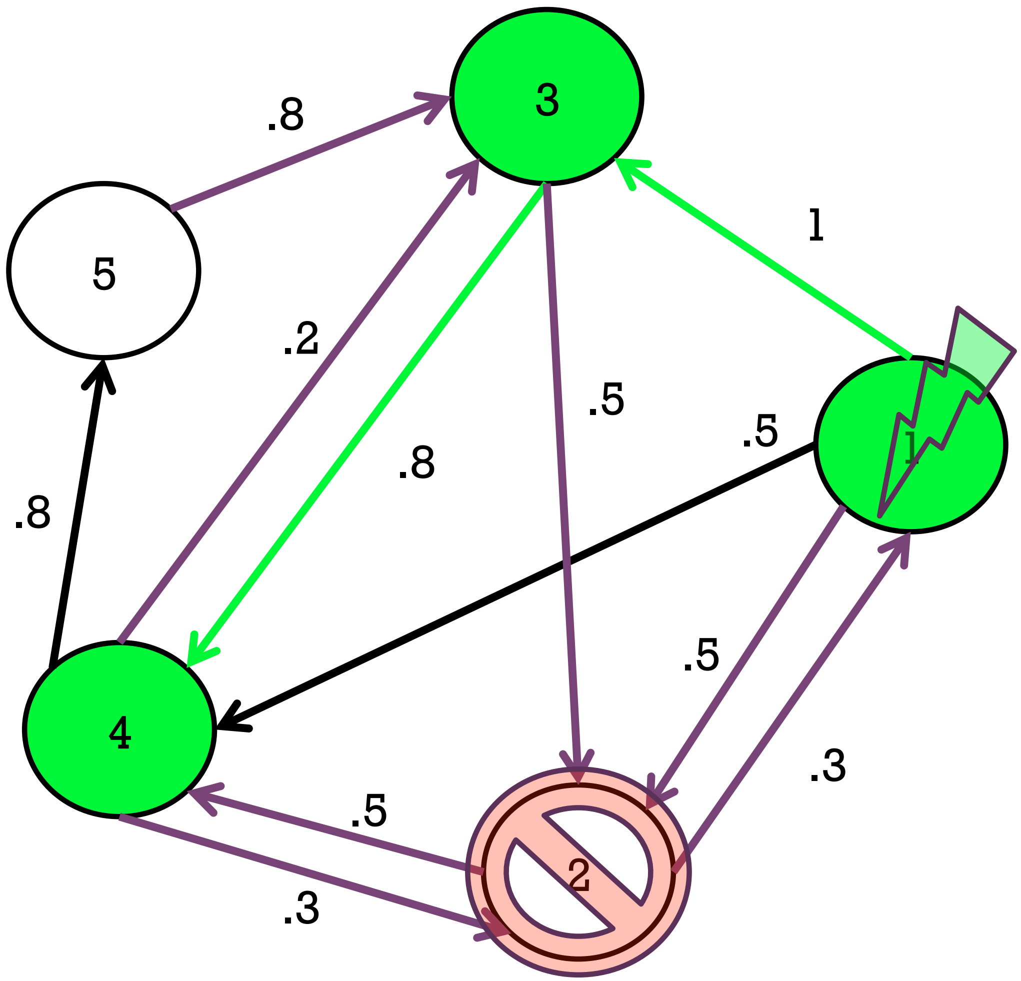


Activations and Suppressions

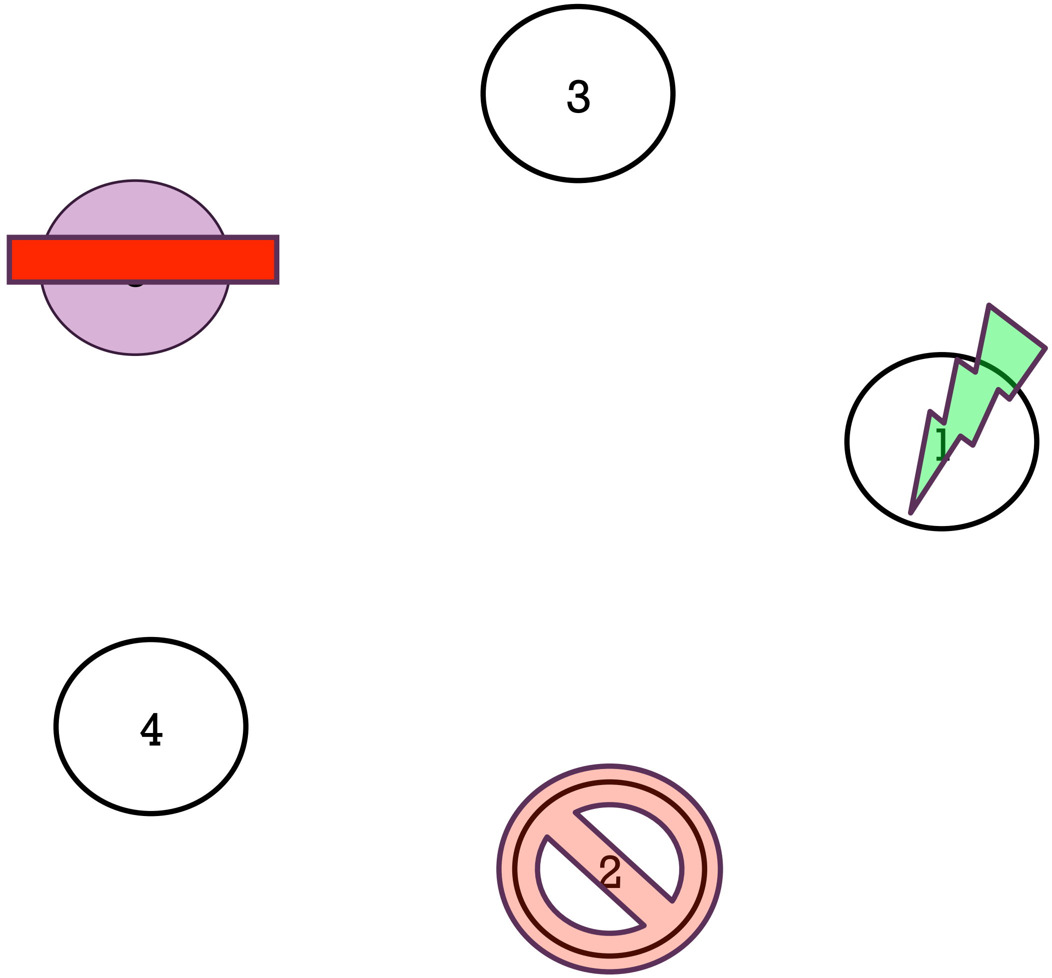


+

Activations and Suppressions

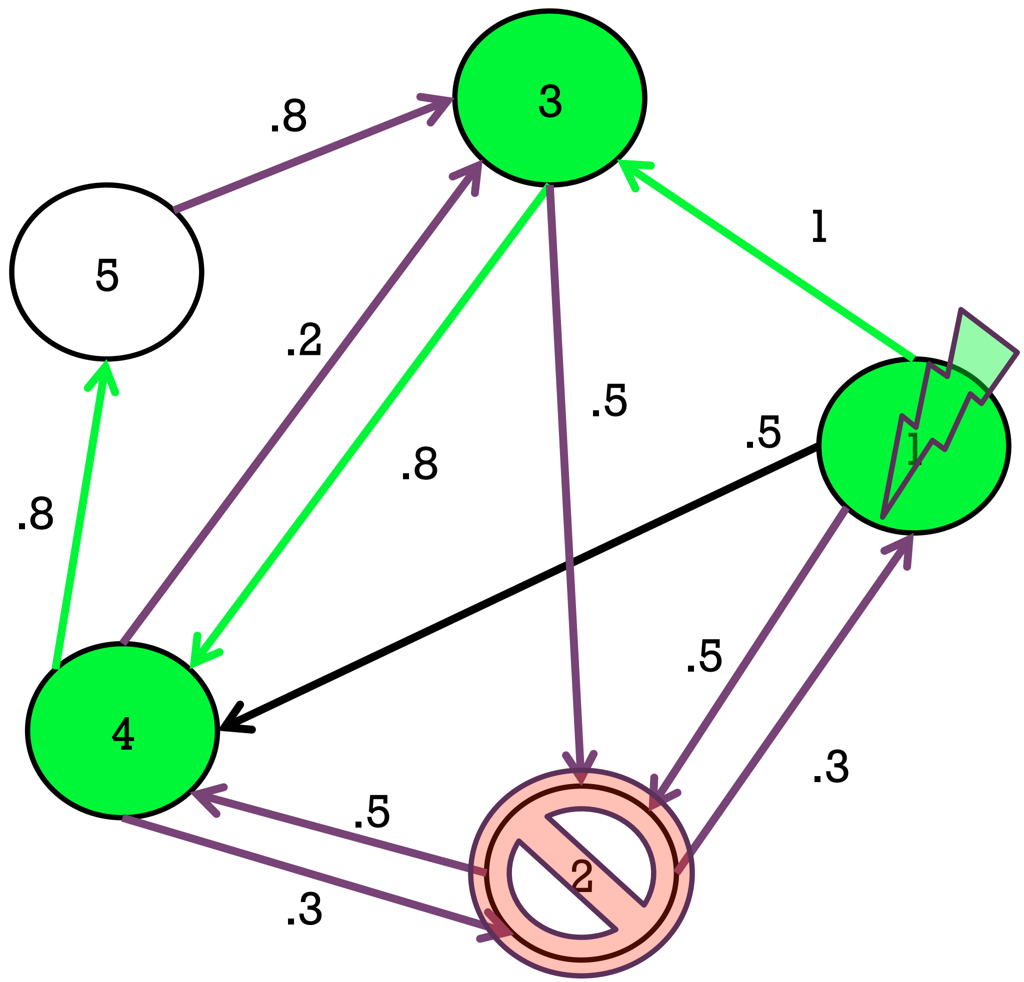


Activations and Suppressions

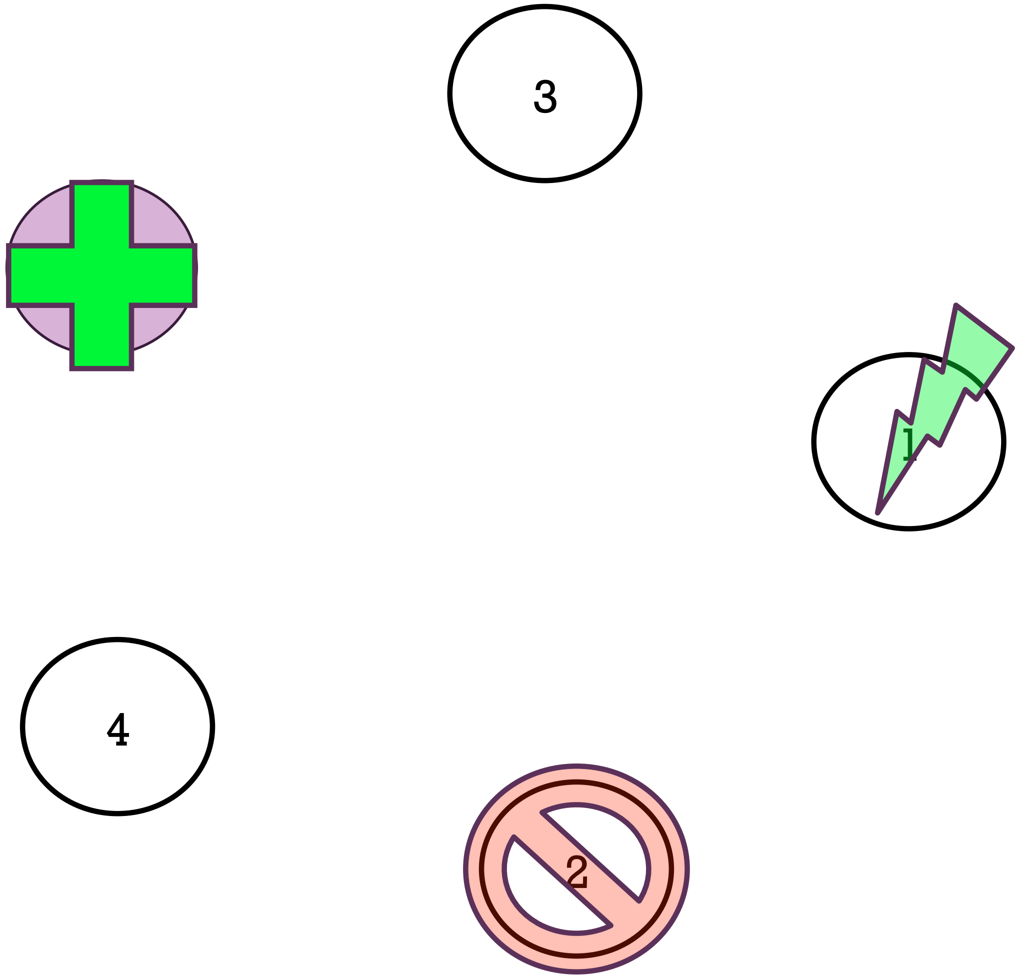


+

Activations and Suppressions

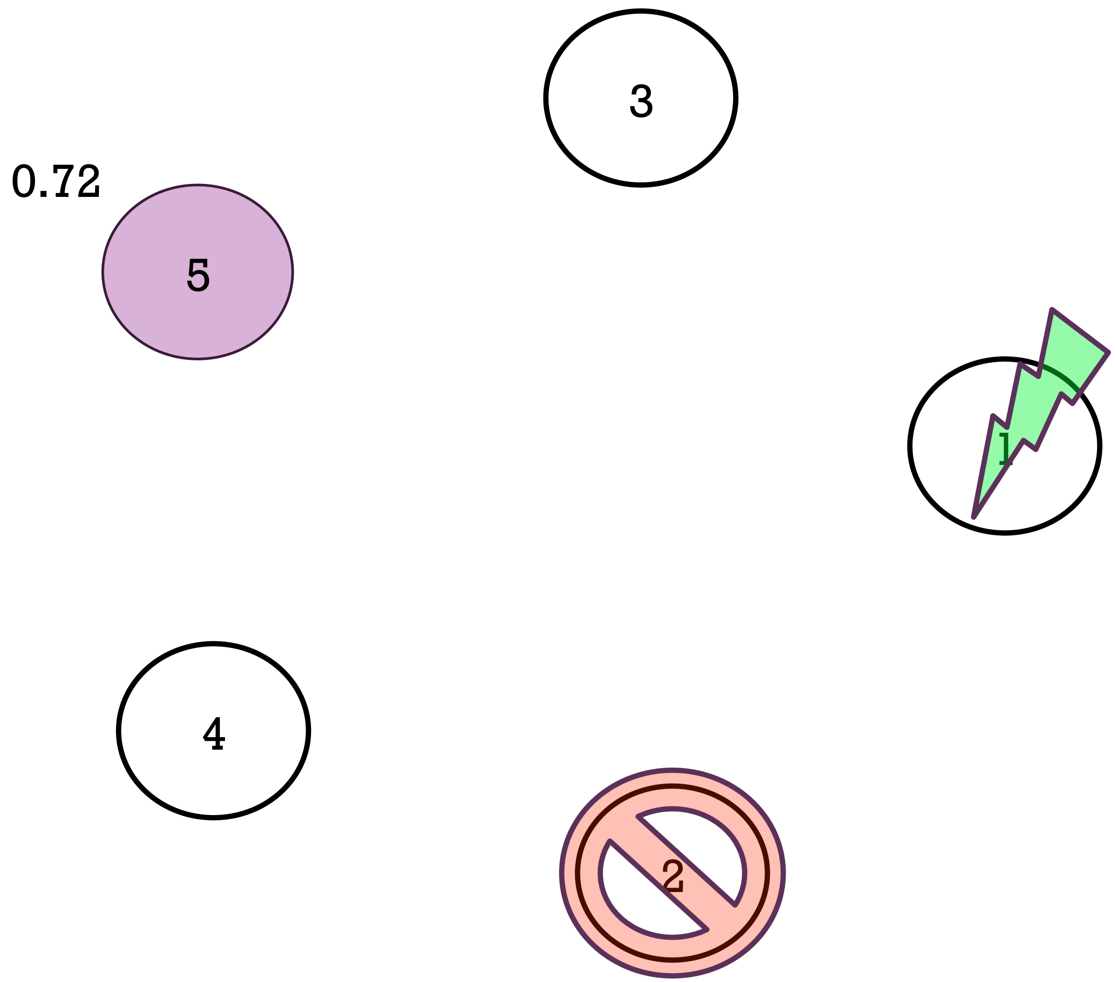


Activations and Suppressions



+

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)

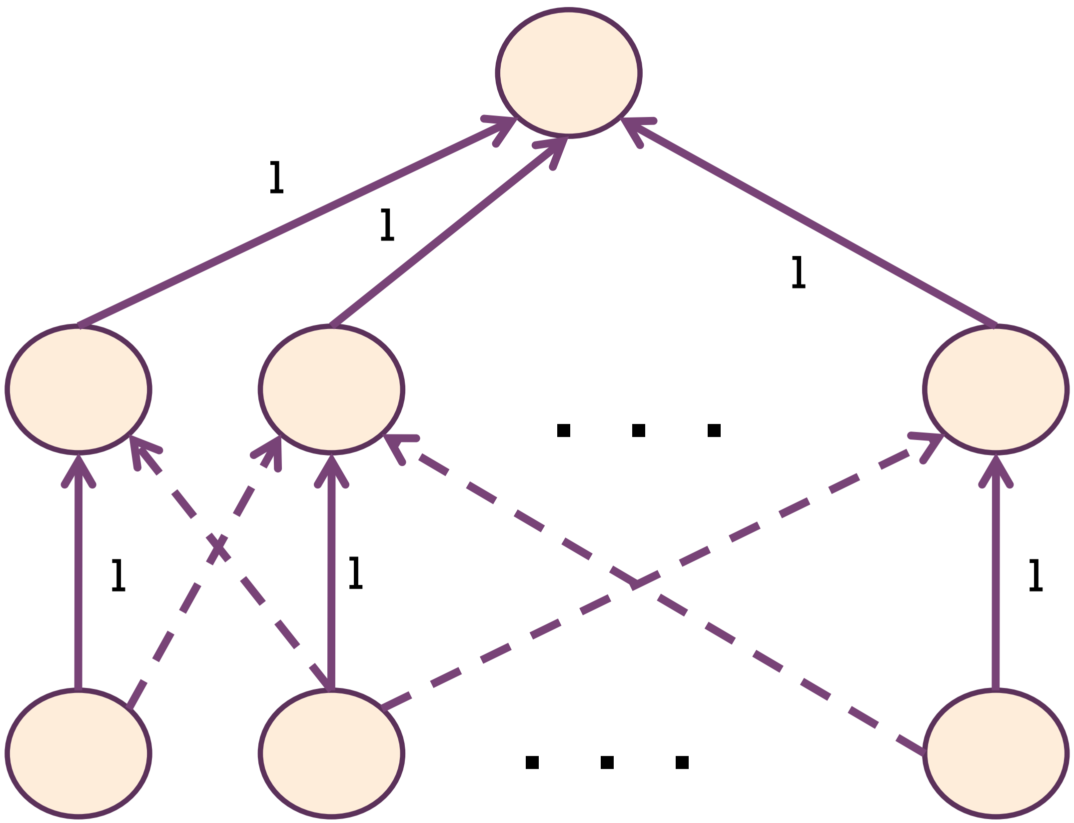
- Given access to a hidden social network S*, the learning problem is to find a social network S behaviorally equivalent to S* using value injection queries.

The Percolation Model

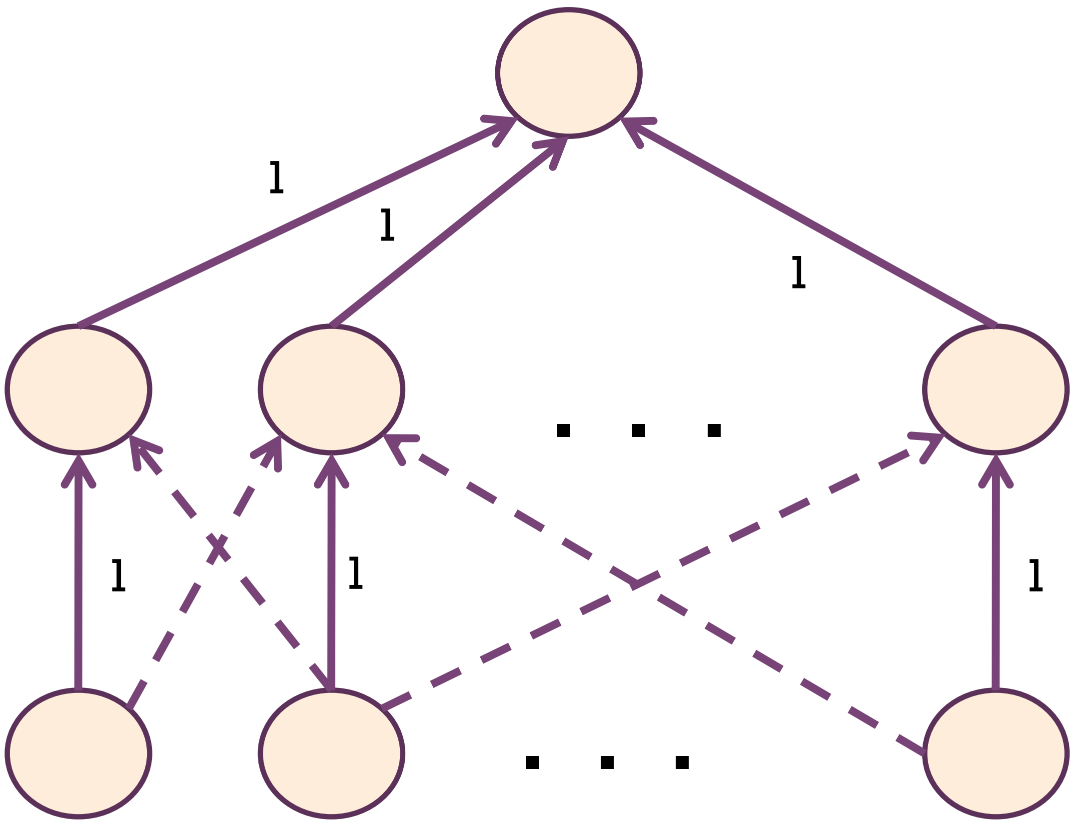
Given a network S and a VIQ

- 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

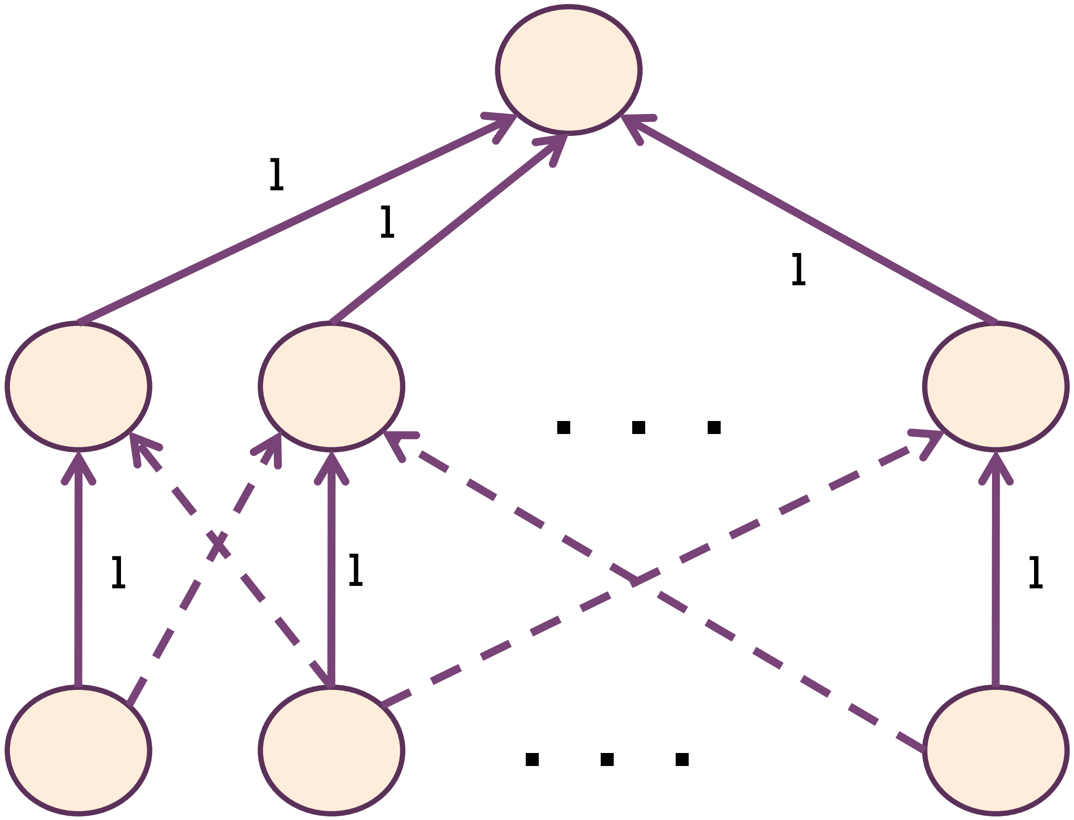


A Lower Bound



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

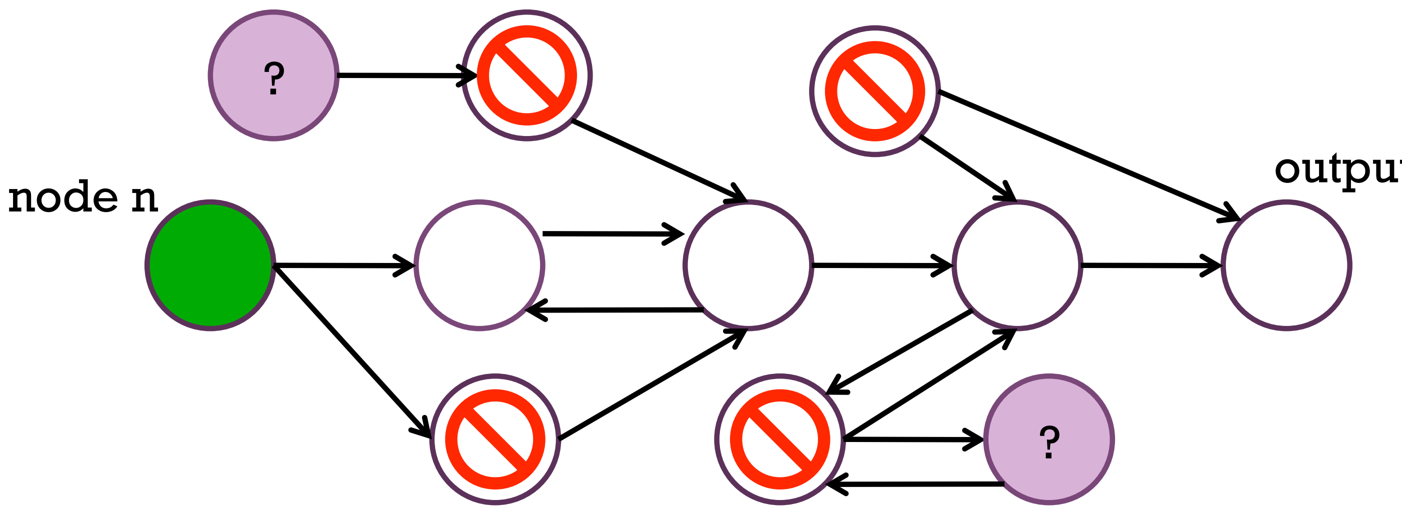
- The depth of a node is its distance to the root
- An Up edge is an edge from a node of larger depth to a node of smaller depth
- A Level edge is an edge between two nodes of same depth
- A Down edge is an edge from a node at smaller depth to a node at higher depth
- A leveled graph of a social network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

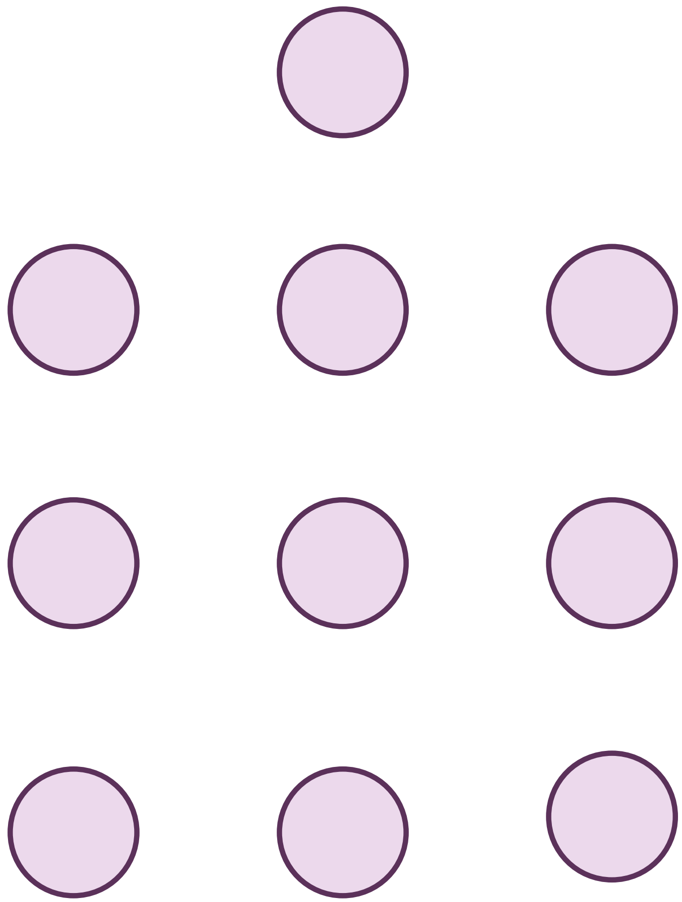
- We also have a shortest excitation path



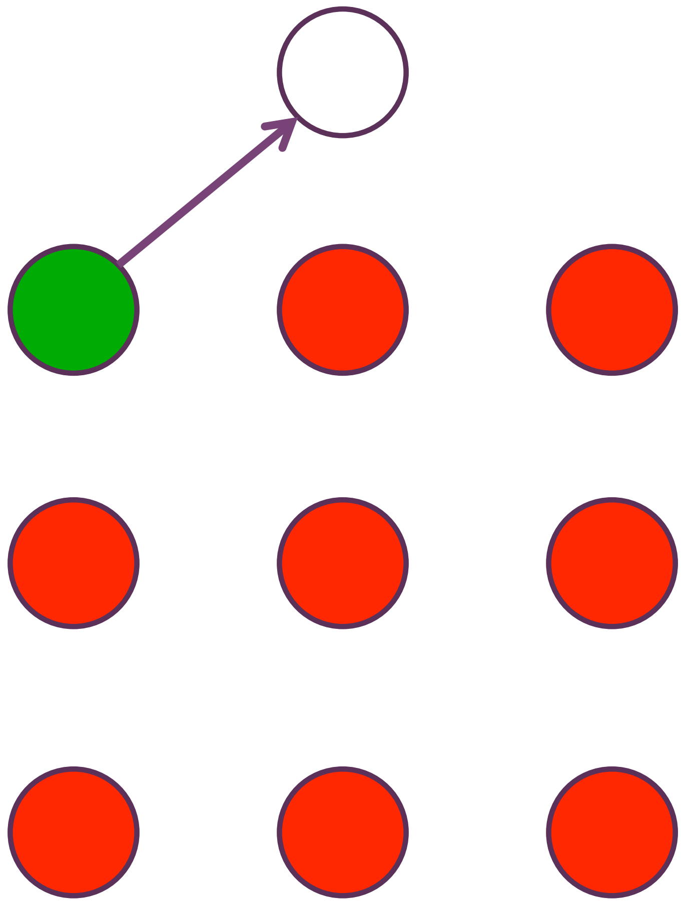
The Learning Algorithm For Networks Without 1 Edges

- 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

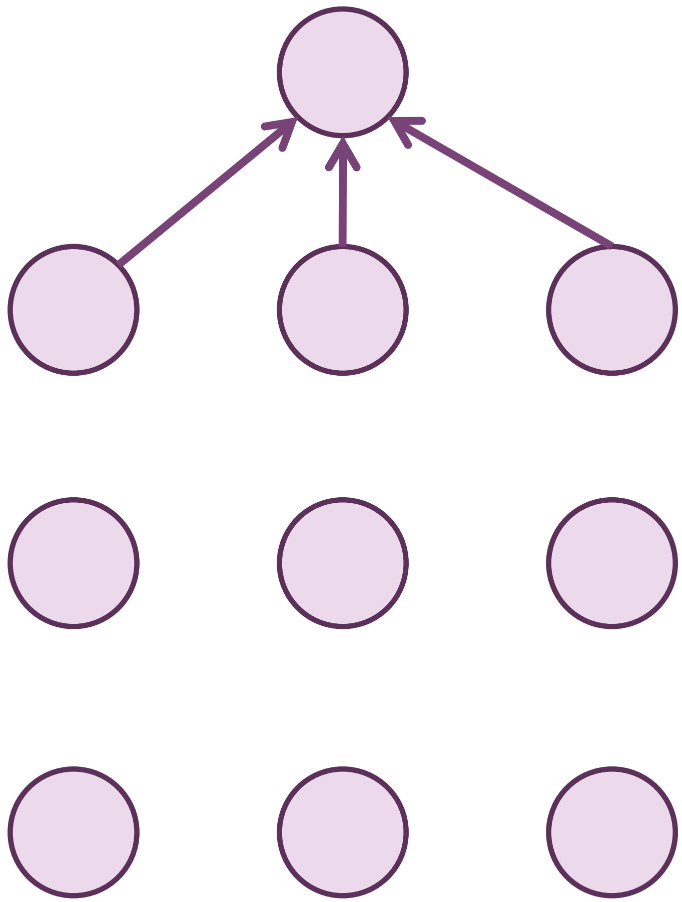
Find-Up-Edges



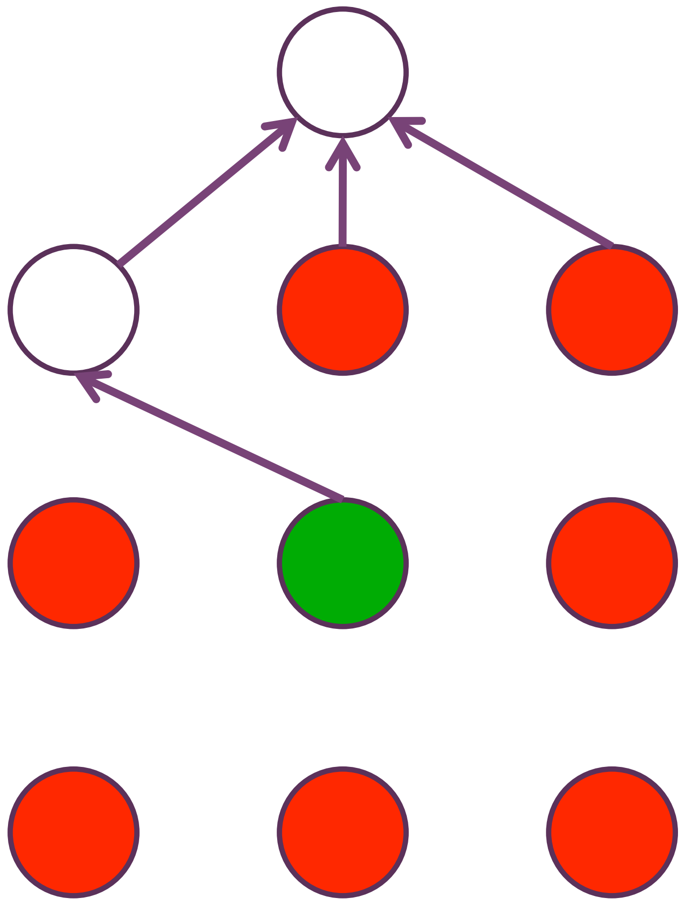
Find-Up-Edges



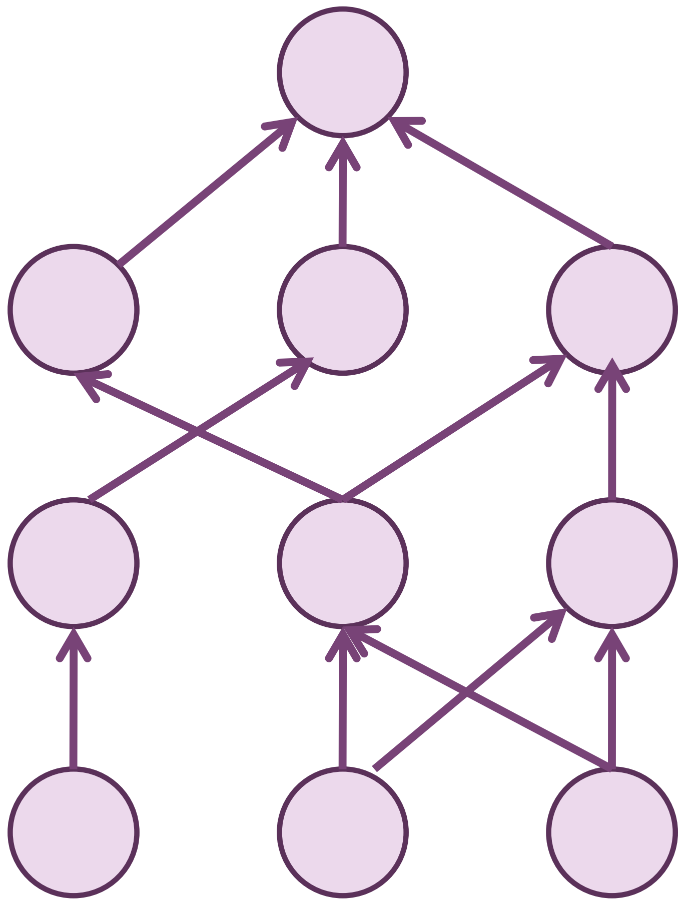
Find-Up-Edges



Find-Up-Edges

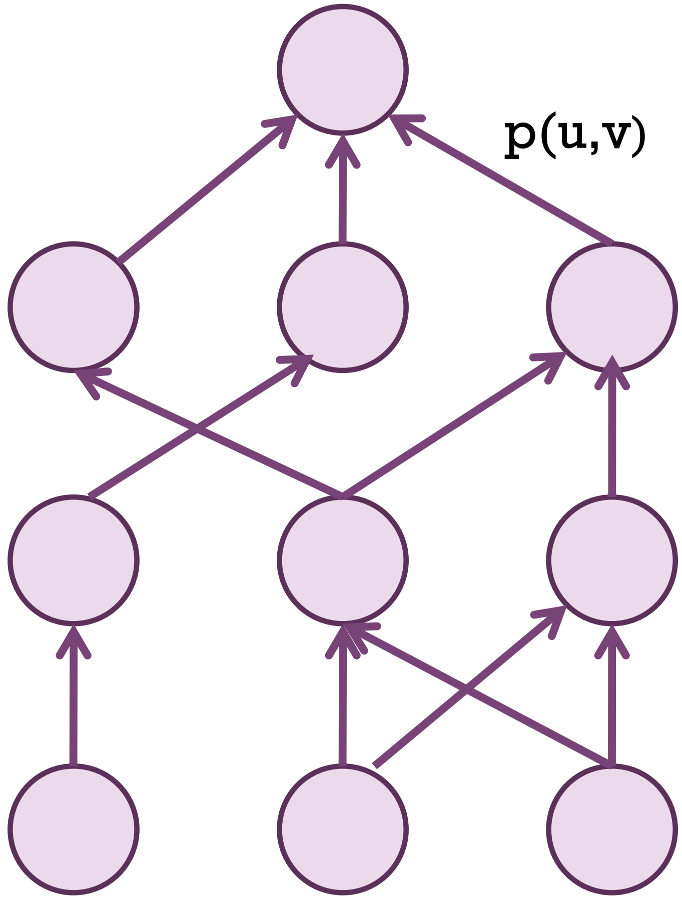


Find-Up-Edges

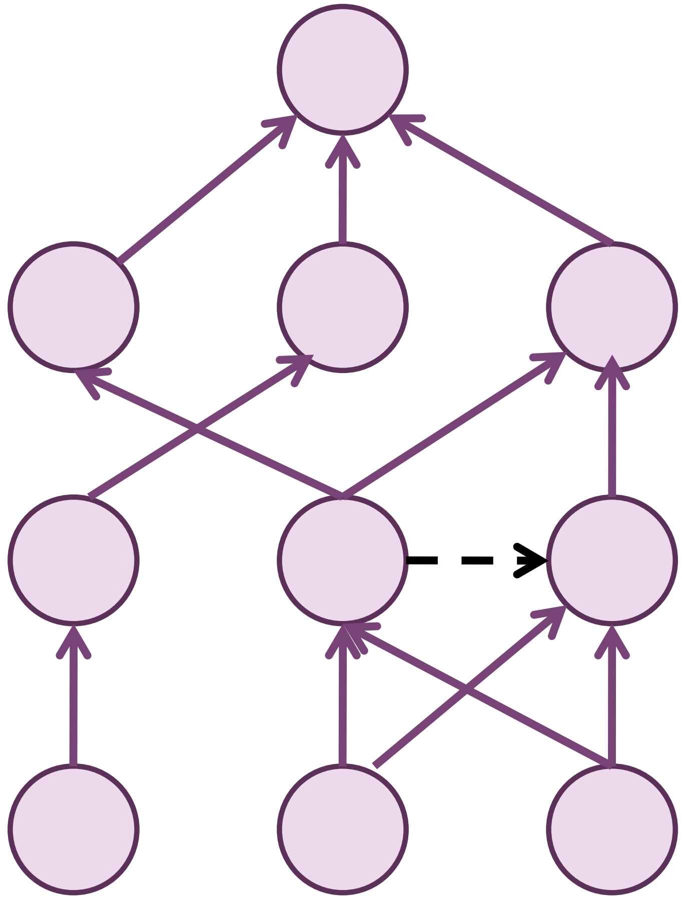


+

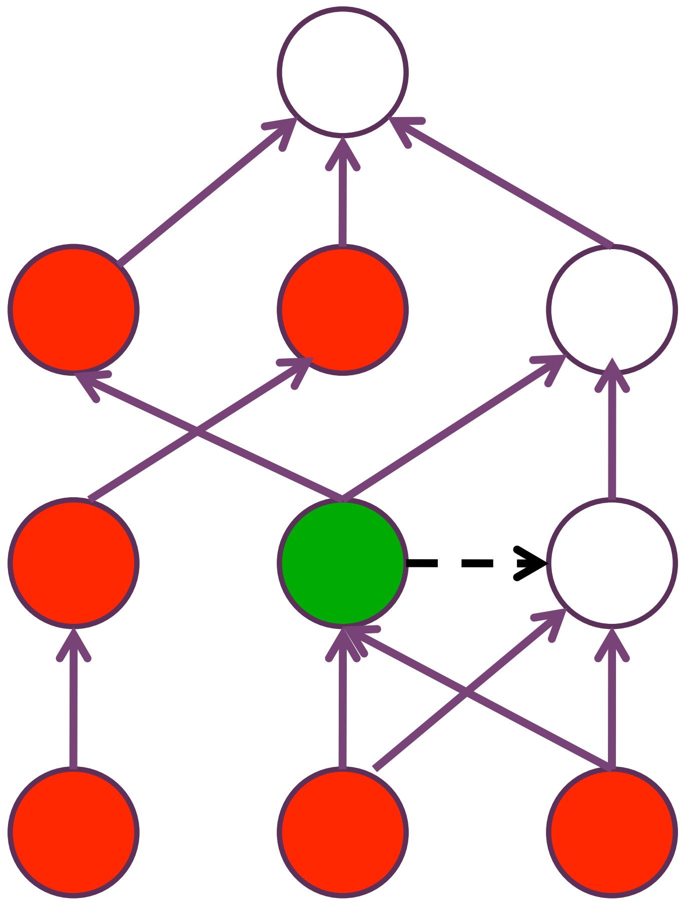
Find-Up-Edges



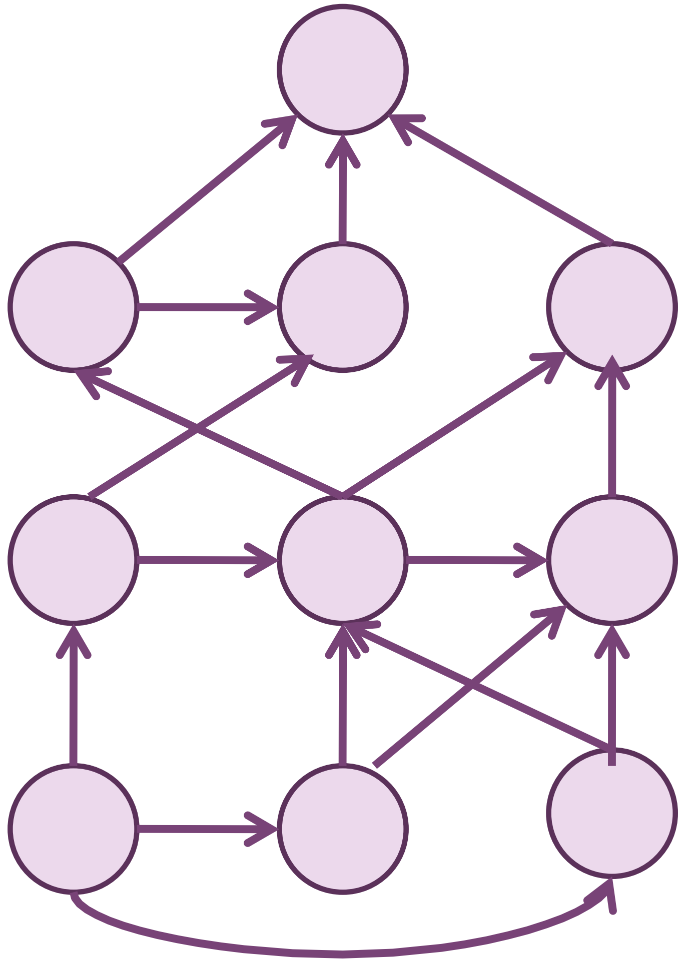
Find-Level-Edges



Find-Level-Edges

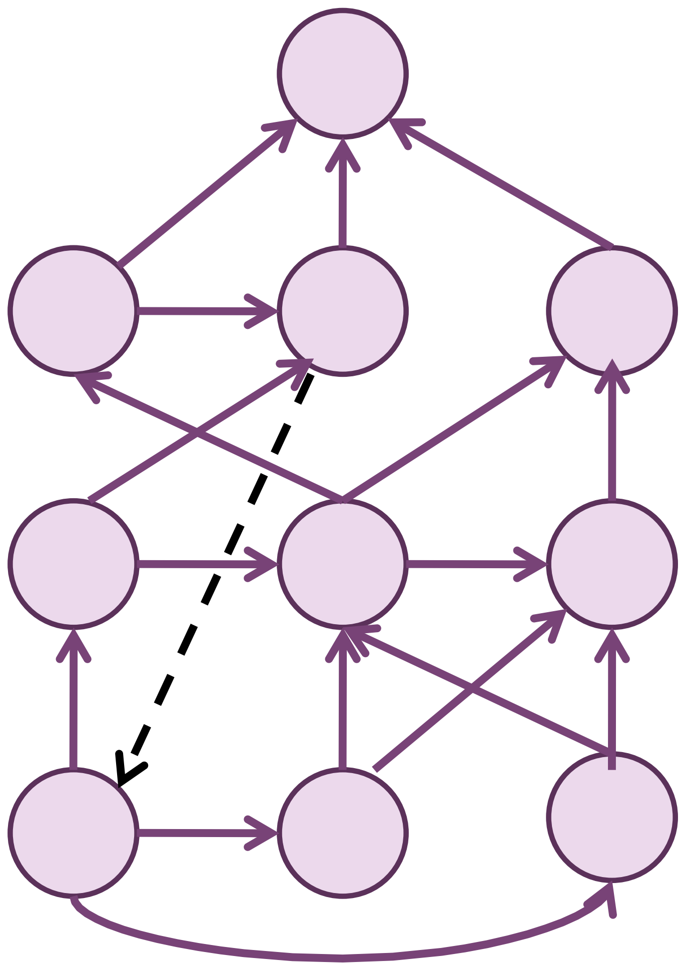


Find-Level-Edges

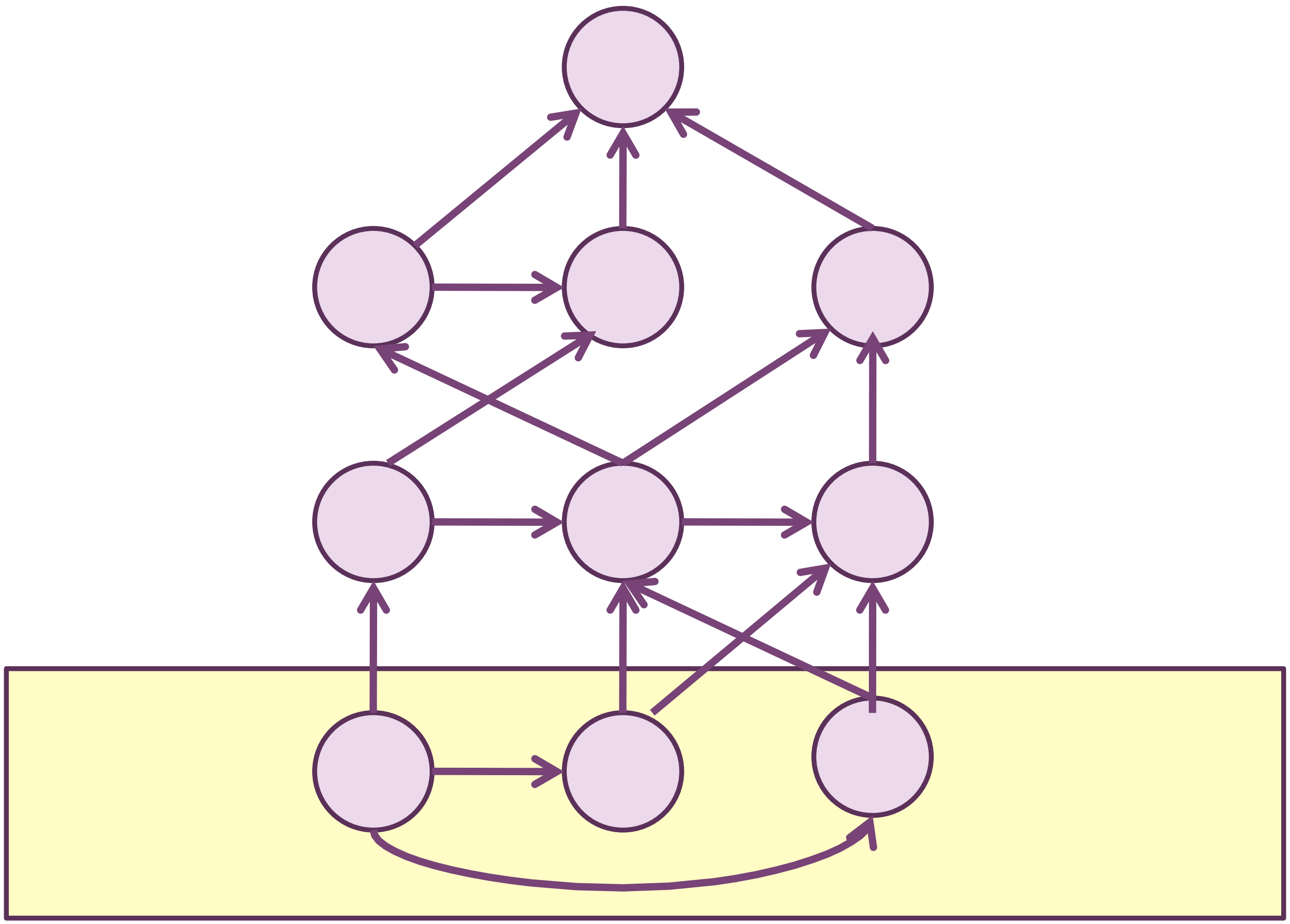


+

Find-Down-Edges

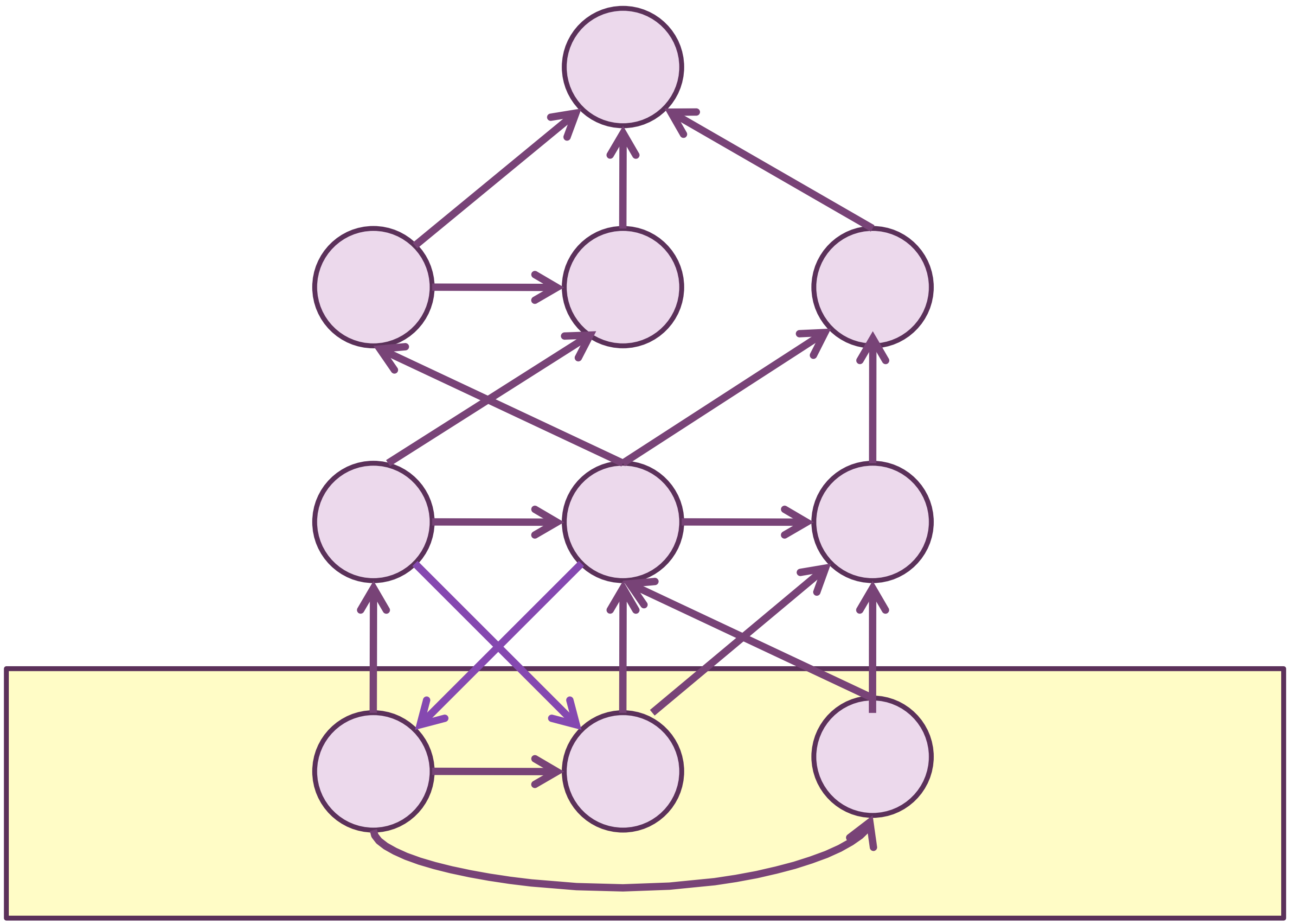


Find-Down-Edges

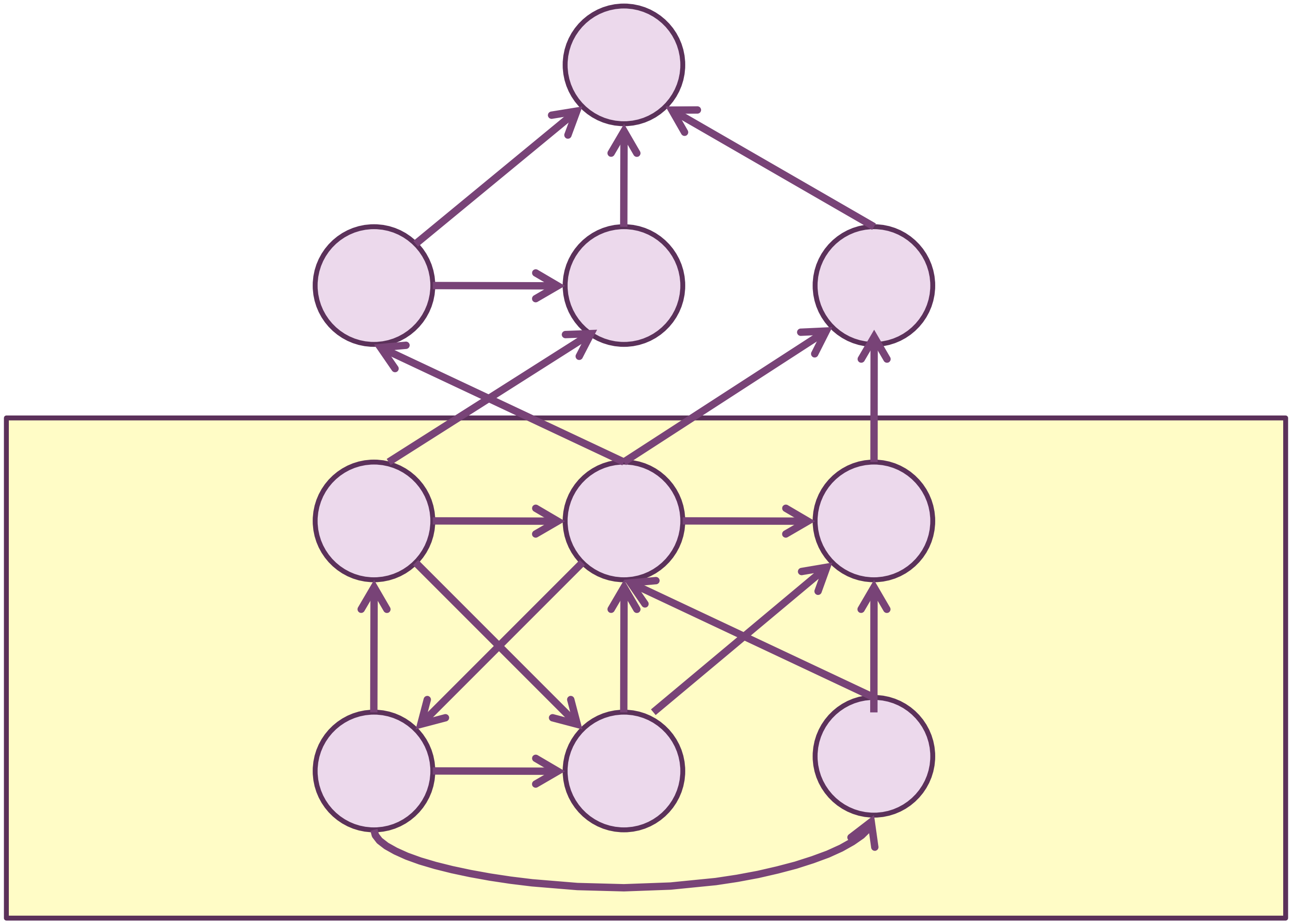


+

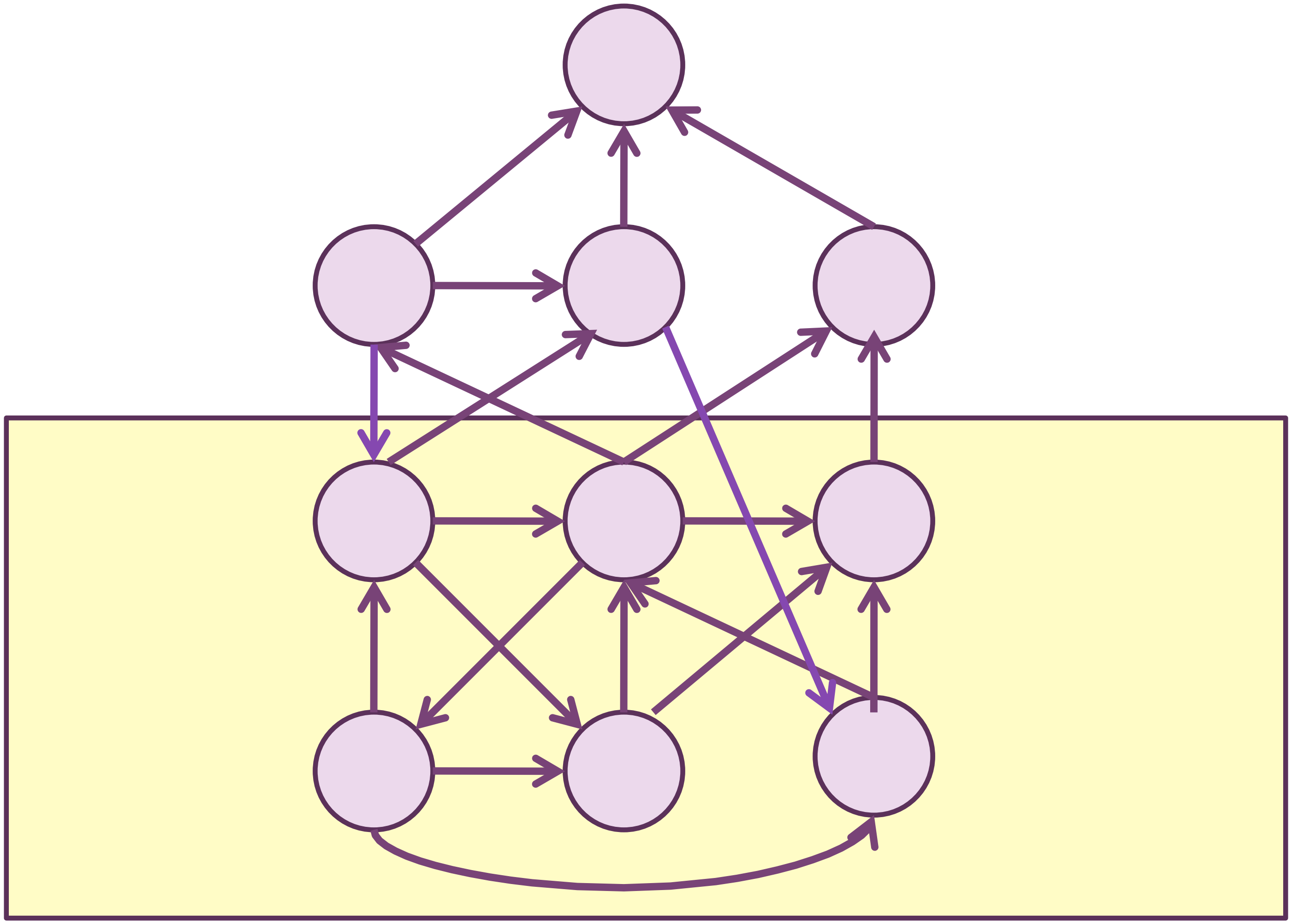
Find-Down-Edges



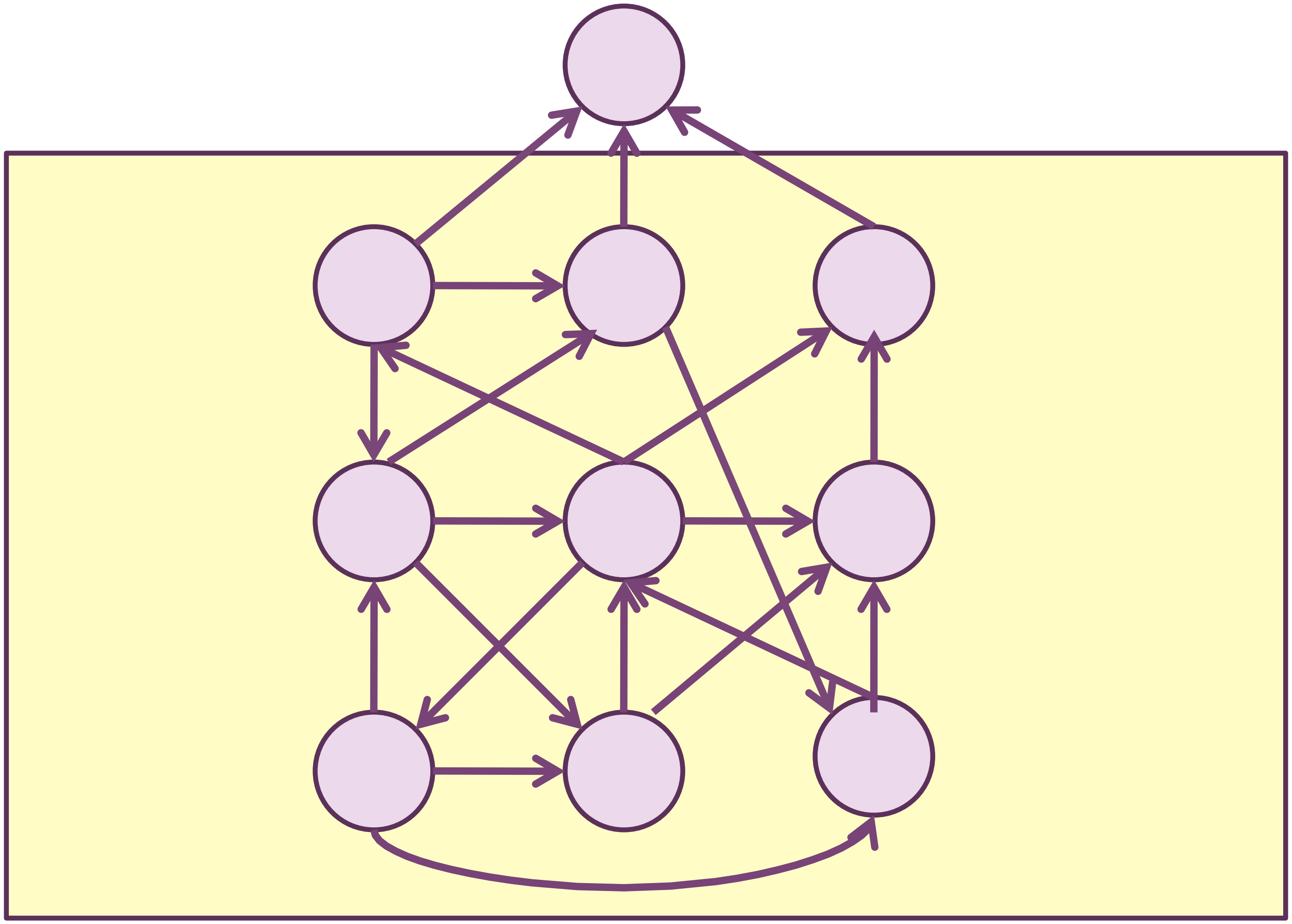
Find-Down-Edges



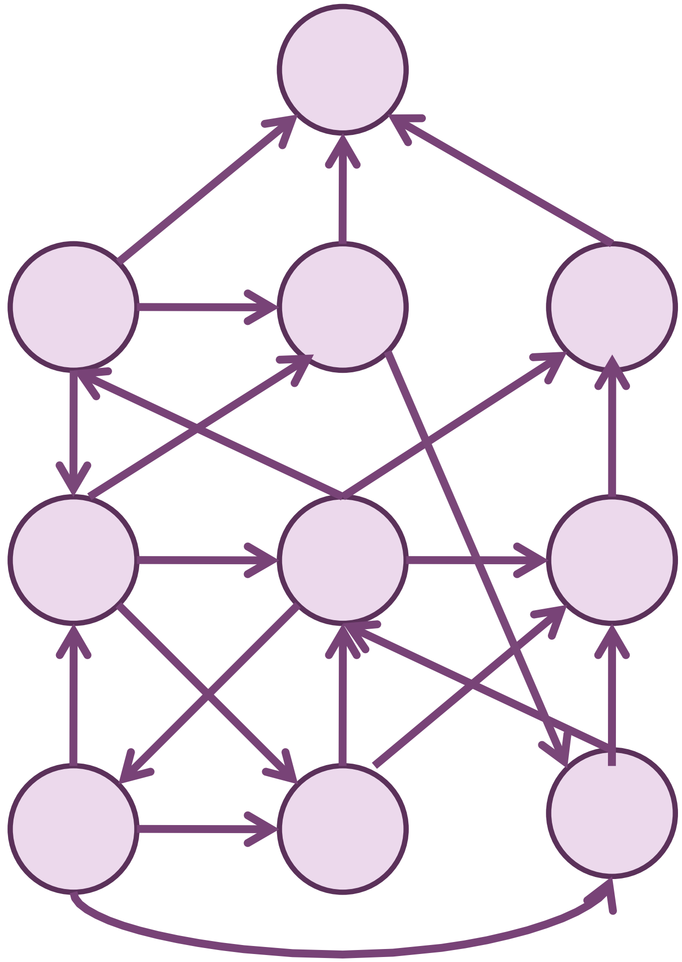
Find-Down-Edges



Find-Down-Edges



Find-Down-Edges

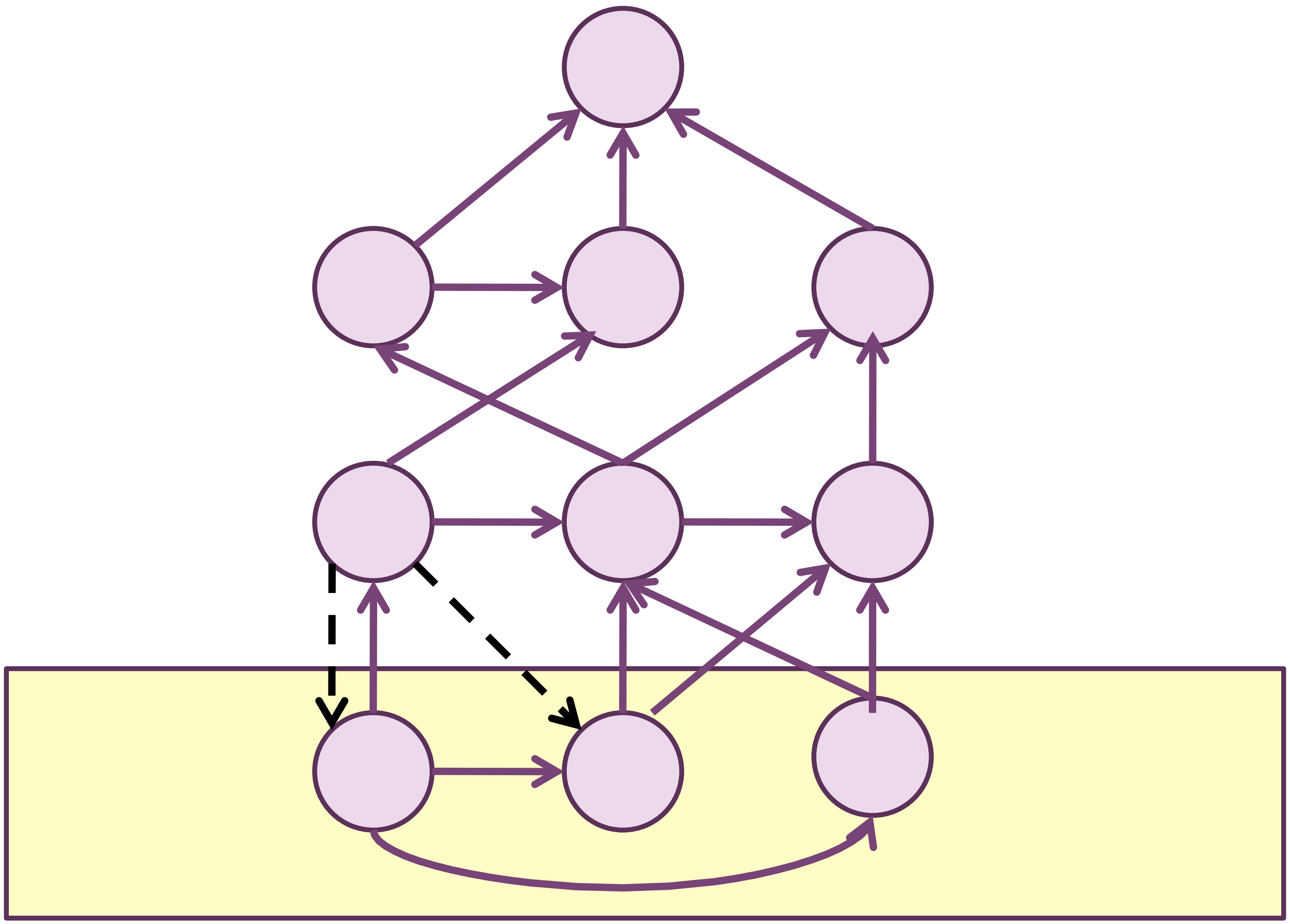


Find-Down-Edges

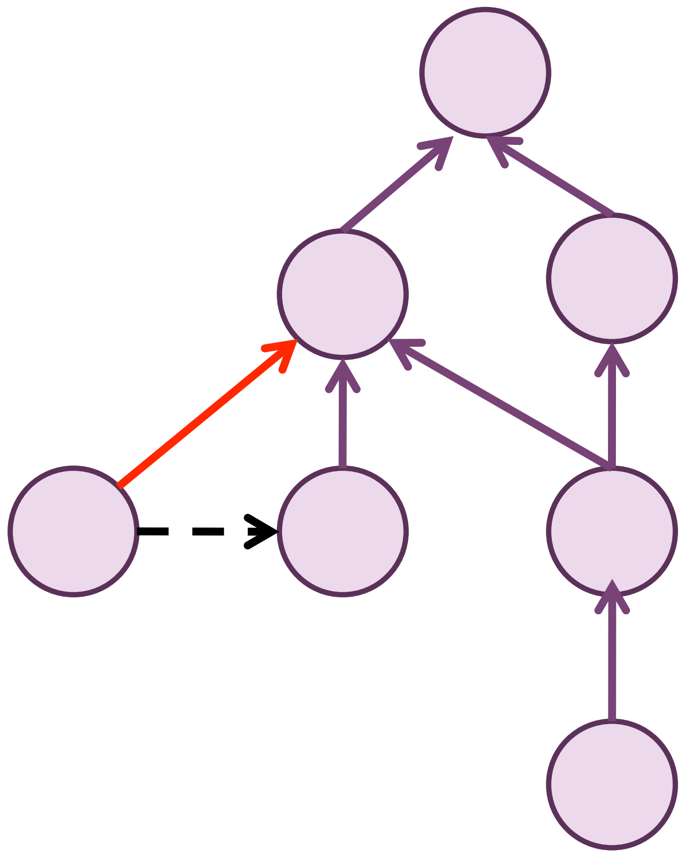
- For each node u at current level
  - Sort each node v_i in C (complete set) by distance to the root in G – {u}
  - Let $v_1 \ldots v_k$ be the sorted v_is
  - Let $pi_1 \ldots pi_k$ be their corresponding shortest paths to the root in G – {u}
  - For i from 1 to k
    - Do experiment of firing u, leaving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

+

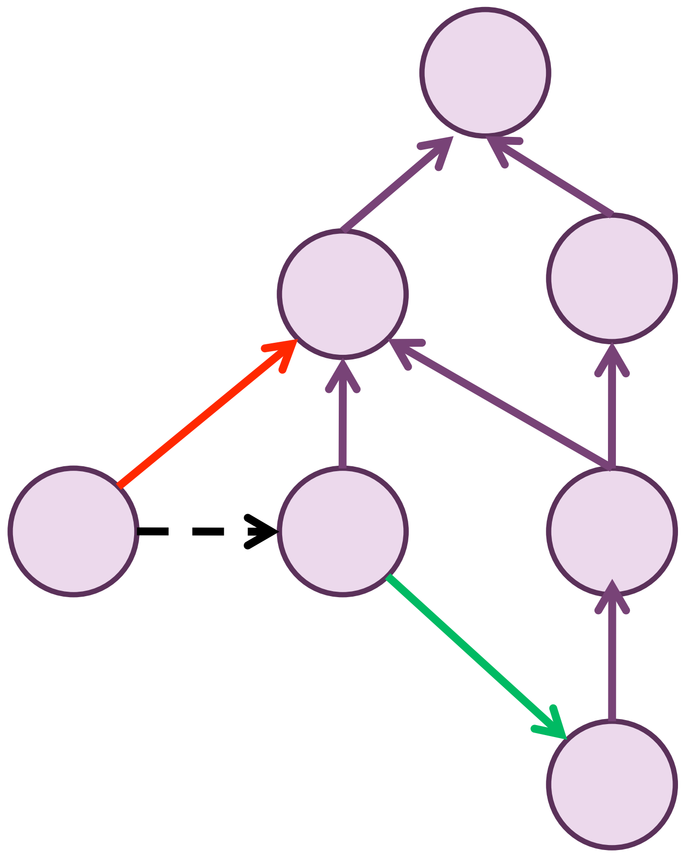
For Example



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 $(p - \varepsilon)$
- 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

- Motivated by real-world problems.
- A new and interesting ways to analyze circuit learning!
- Interesting (and surprising) learnability boundaries!
- Many questions open
  - Restricting the number of non-free gates in an experiment.
  - More realistic models of circuits (ie social networks).
  - Exact vs non-exact queries.
  - Connections to complexity theory.