



Learning Hidden Graphs and Circuits with Query Access

Machine Learning Talk
At University of Massachusetts Amherst

Lev Reyzin
Yale University

Main Papers

- Lev Reyzin and Nikhil Srivastava **Learning and Verifying Graphs Using Queries with a Focus on Edge Counting** In *Proceedings of the 18th International Conference on Algorithmic Learning Theory* (ALT 2007).
- Dana Angluin, James Aspnes, Jiang Chen, and Lev Reyzin **Learning Large-Alphabet and Analog Circuits with Value Injection Queries** In *Proceedings of the 20th Annual Conference on Learning Theory* (COLT 2007).

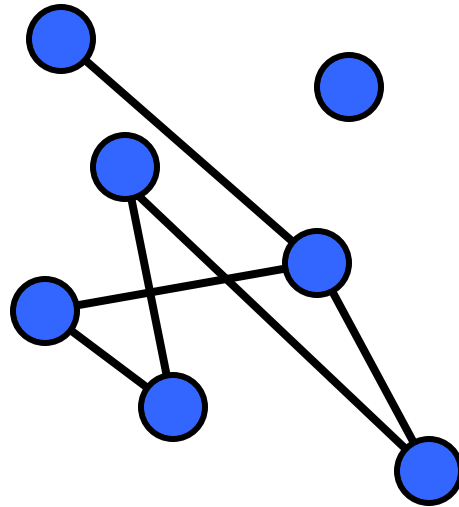
Plan of Talk

- **Graph Learning and Verification**
 - introduce ED and EC queries
 - learning partition with EC and ED queries
 - verifying with EC queries
- **Circuit Learning with Value Injection**
 - introduce value injection model
 - hardness result for large alphabet circuits
 - poly time algorithm for transitively reduced circuits
- **Summary**

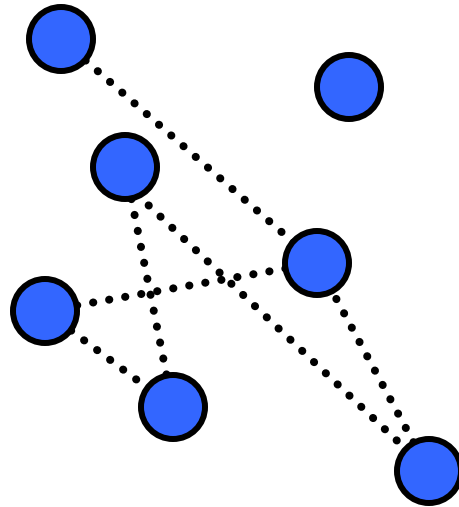
Plan of Talk

- **Graph Learning and Verification**
 - introduce ED and EC queries
 - learning partition with EC and ED queries
 - verifying with EC queries
- **Circuit Learning with Value Injection**
 - introduce value injection model
 - hardness result for large alphabet circuits
 - poly time algorithm for transitively reduced circuits
- **Summary**

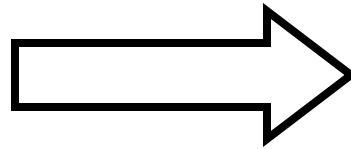
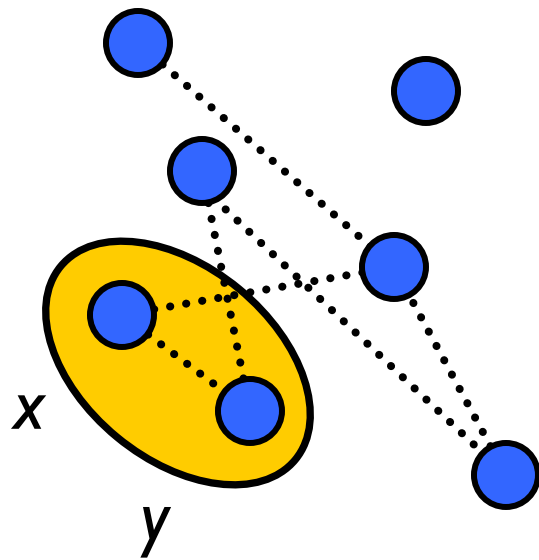
Queries



Queries

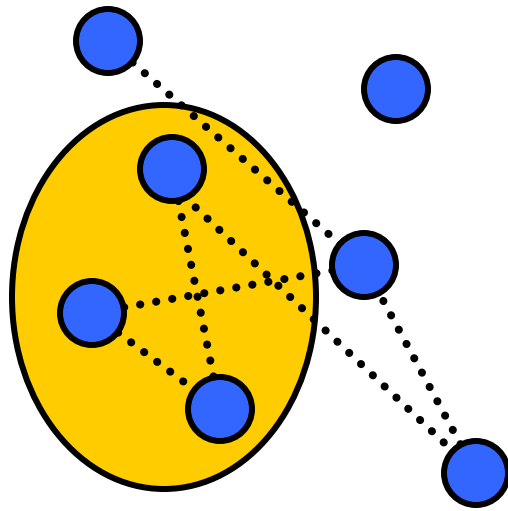


Queries



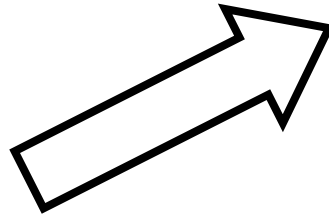
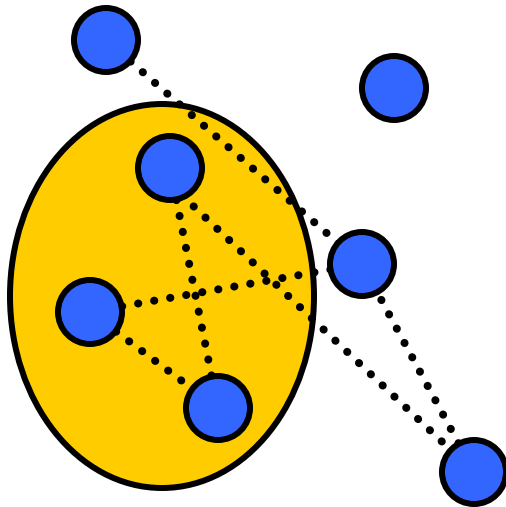
$\{ xy^2 \in E \}$

Queries



S in V

Queries

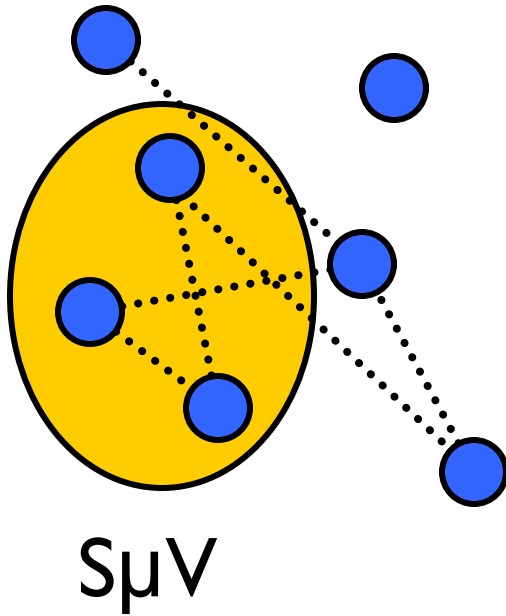


$ED(S) =$

$\{S \text{ contains an edge}\}$

$S \mu V$

Queries



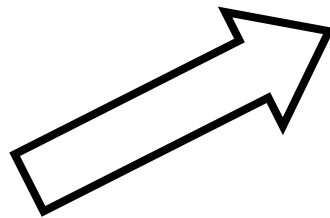
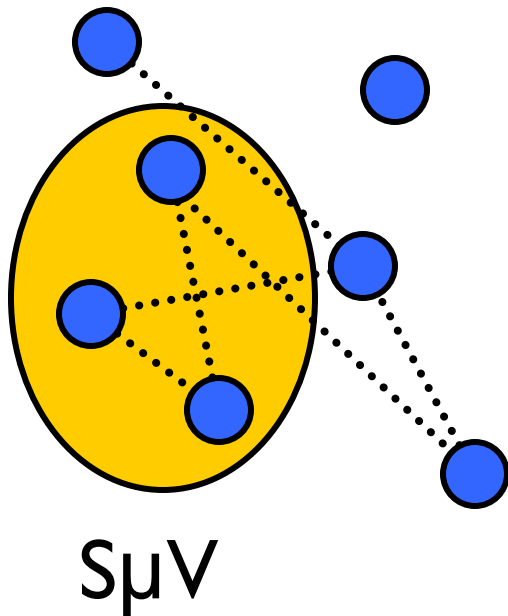
$$ED(S) =$$

$$|\{S \text{ contains an edge}\}|$$

$$EC(S) =$$

(# edges induced by S)

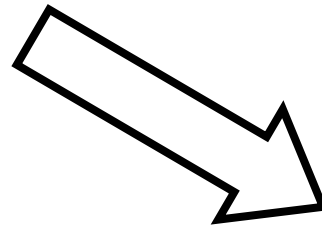
Queries



$$ED(S) =$$

$$|\{S \text{ contains an edge}\}|$$

[AC04,06]



$$EC(S) =$$

(# edges induced by S)

[GK99,00,05]

Motivation

- **ED queries – example from Chemistry.**
 - Nodes are chemicals
 - Edges are chemical interactions
 - Queries \leftrightarrow mixing chemicals in experiments
 - Goal: finding all reaction pairs
- **EC queries – PCR (Polymerase Chain Reactions)**
 - Nodes are primers
 - Edges represent adjacency in DNA
 - Queries \leftrightarrow multiplex primer testing
 - Goal: sequencing genome of an organism

Learning Tasks

- Learning:
 - Find G .
 - Find the connected components of G .
 - Also: *girth, chromatic number, diameter, expansion, edge-connectivity, etc.*
- Verification:
 - Is $G=G_0$ for a given G_0 ?

Some Previously Known Results

- Edge Detection
 - Can learn graphs in $O(E \log n)$. Matches the lower bound. [Angluin, Chen]
- Edge Counting
 - Can learn graphs in $O(dn)$ and $O(n^2/\log n)$. Both are lower bounds. [Grebinskiy, Kucherov]

Plan of Talk

- **Graph Learning and Verification**
 - introduce ED and EC queries
 - **learning partition with EC and ED queries**
 - verifying with EC queries
- **Circuit Learning with Value Injection**
 - introduce value injection model
 - hardness result for large alphabet circuits
 - poly time algorithm for transitively reduced circuits
- **Summary**

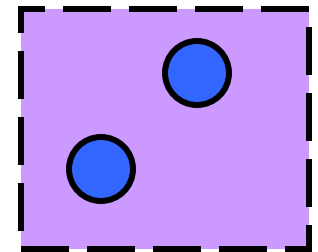
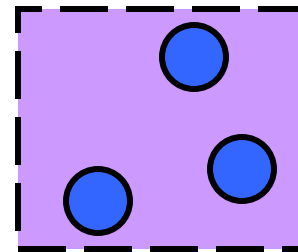
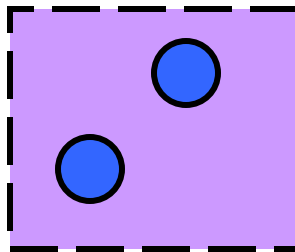
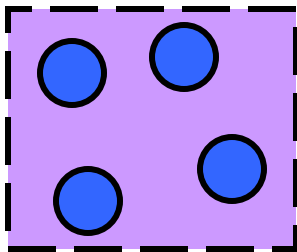
Partition

Thm: Can learn partition in $O(n \log n)$ EC queries.

Partition

Thm: Can learn partition in $O(n \log n)$ EC queries:

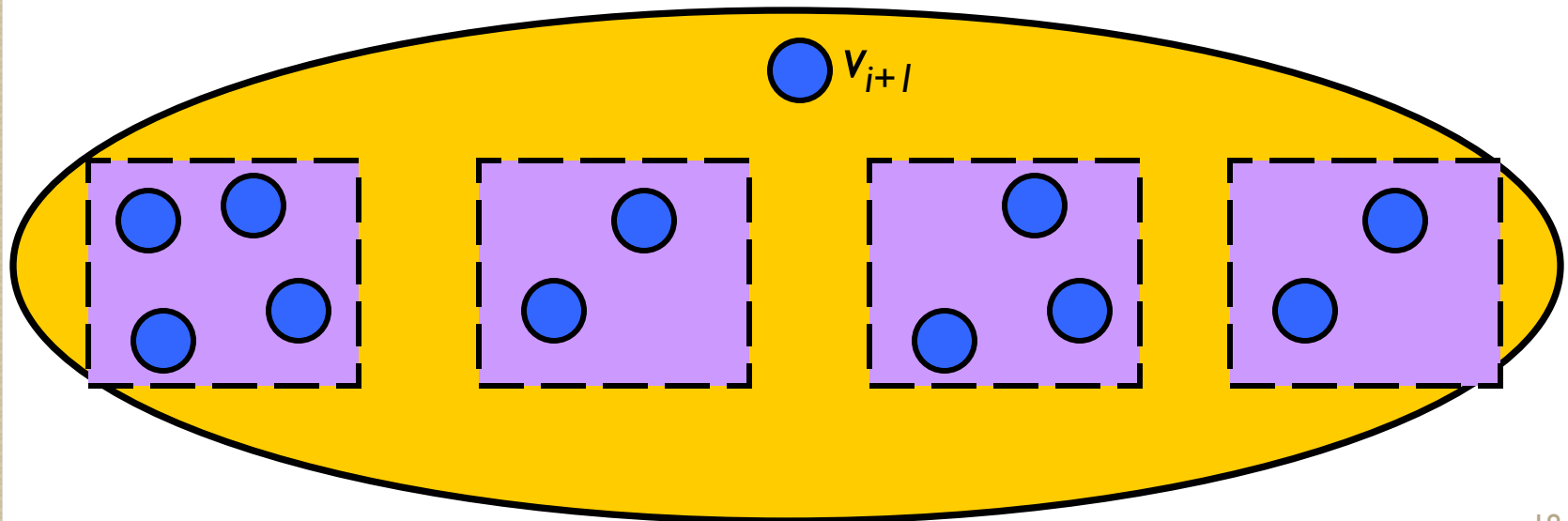
1. Let $V = \{v_1 \dots v_n\}$
2. Suppose we know components in $\{v_1 \dots v_i\}$.



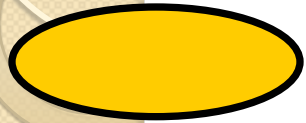
Partition

Thm: Can learn partition in $O(n \log n)$ EC queries:

1. Let $V = \{v_1 \dots v_n\}$
2. Suppose we know components in $\{v_1 \dots v_i\}$.

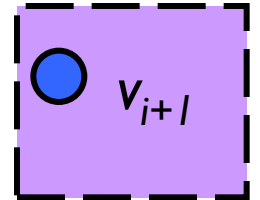
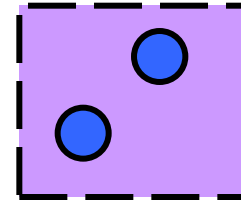
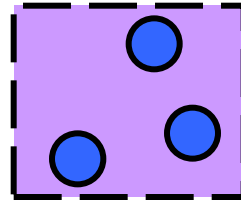
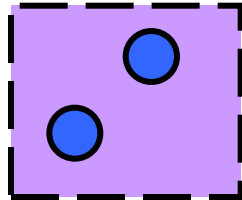
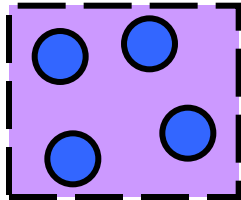


Partition

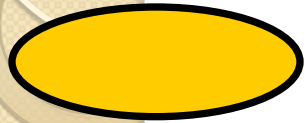


= 0

create new component

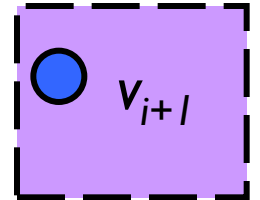
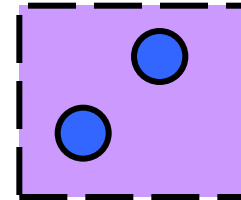
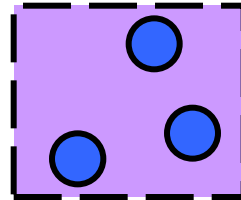
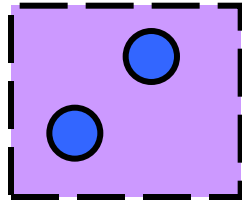
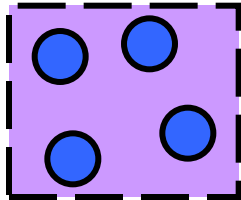


Partition



= 0

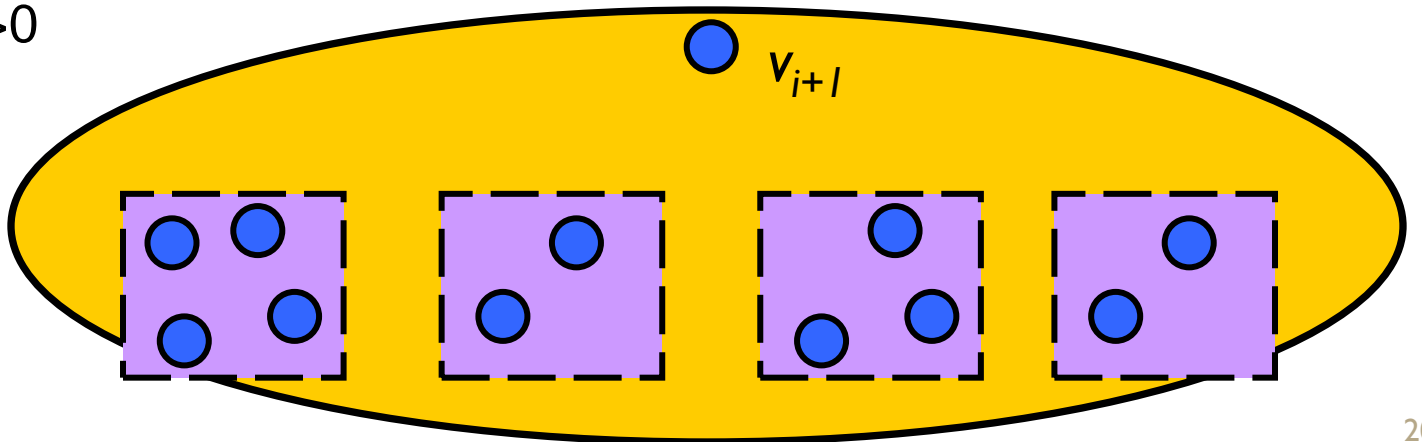
create new component



recurse on half until edge is found



> 0

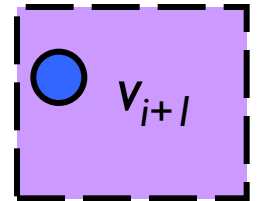
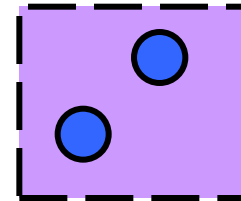
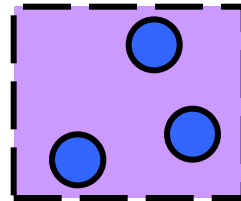
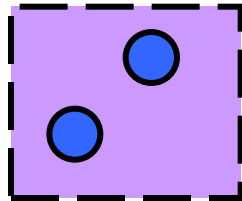
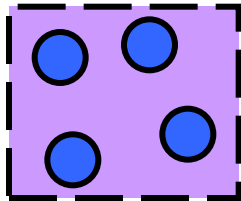


Partition



= 0

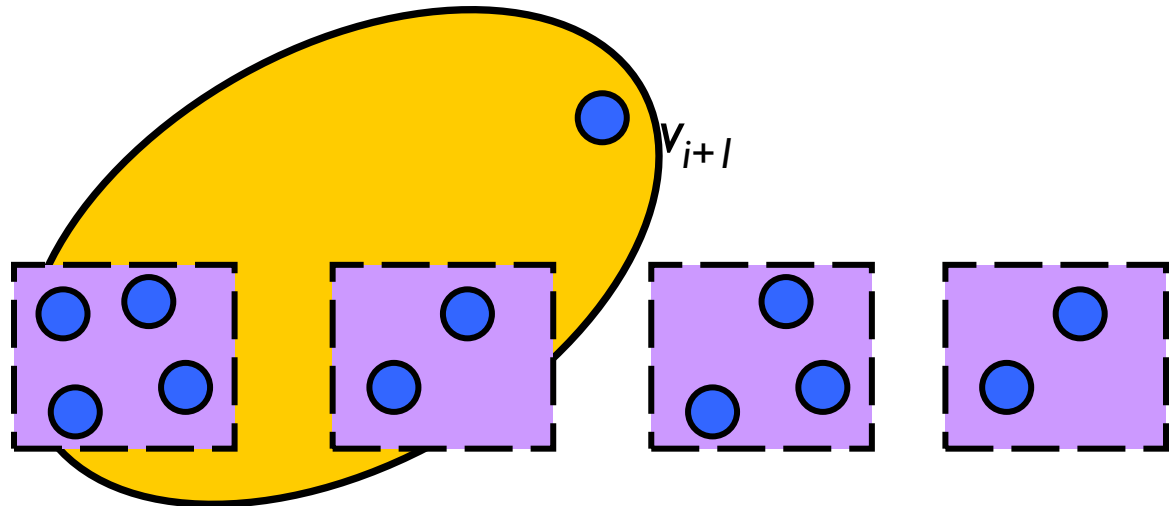
create new component



recurse on half until edge is found



> 0

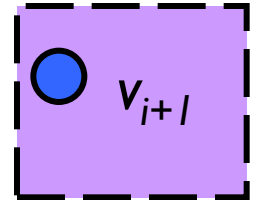
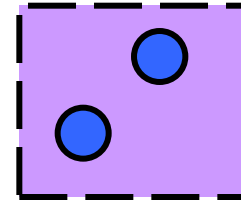
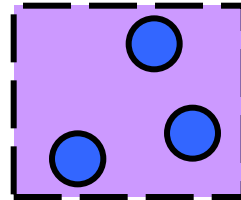
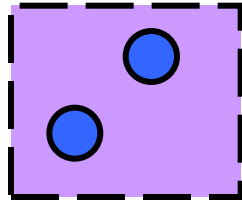
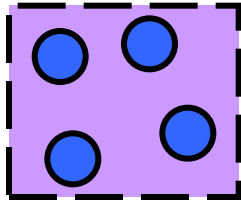


Partition



= 0

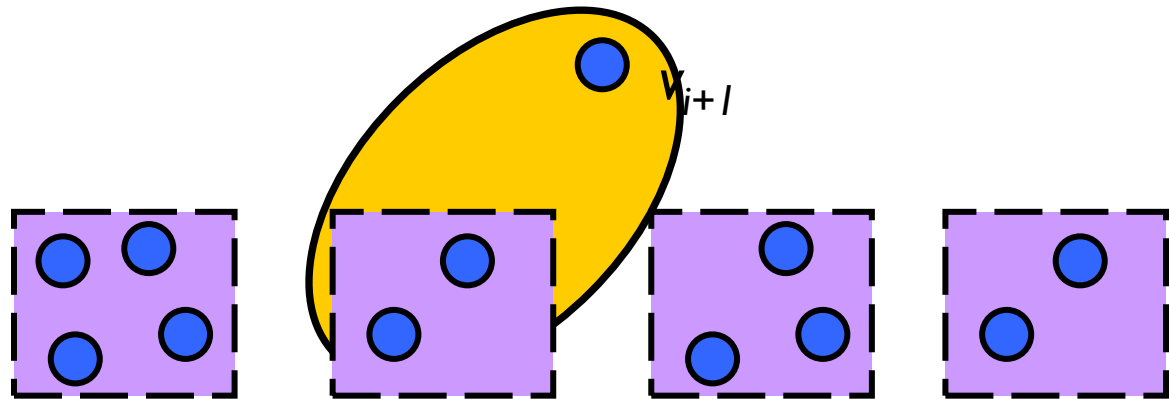
create new component



recurse on half until edge is found



> 0

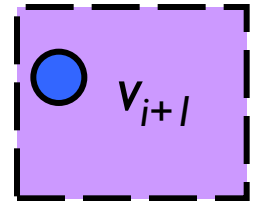
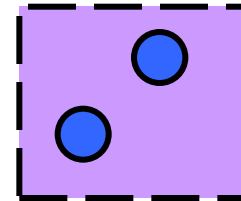
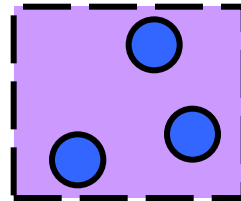
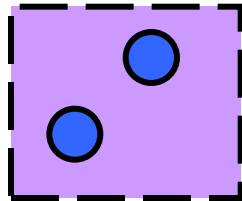
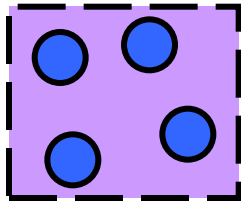


Partition



= 0

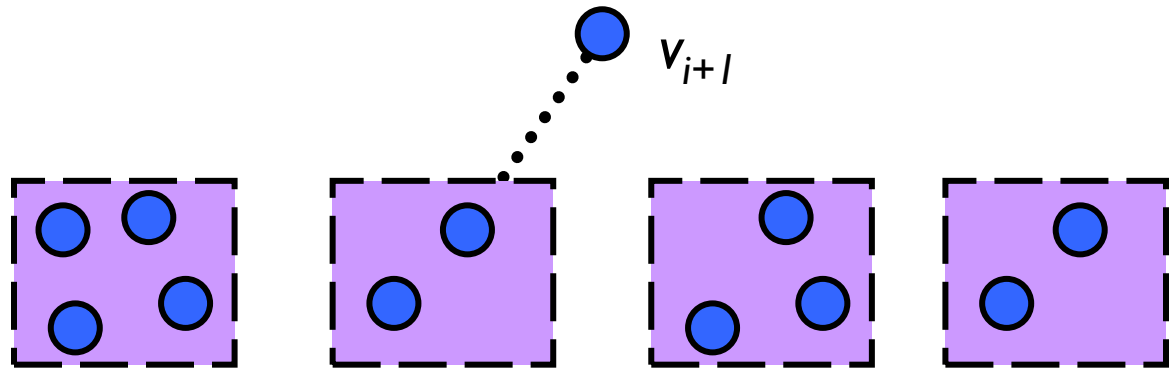
create new component



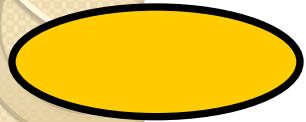
recurse on half until edge is found



> 0

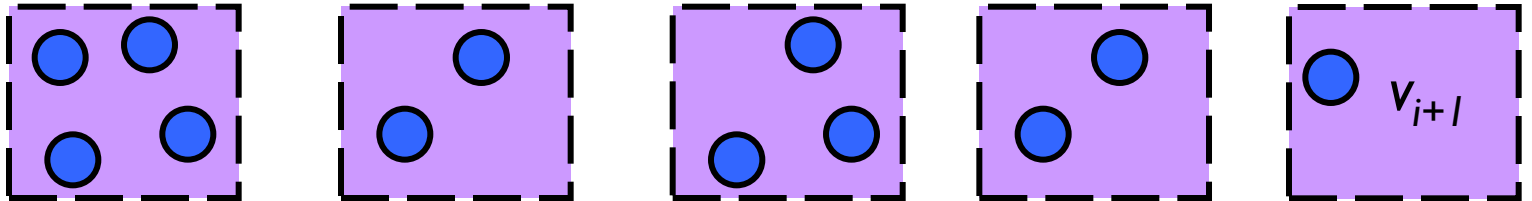


Partition



= 0

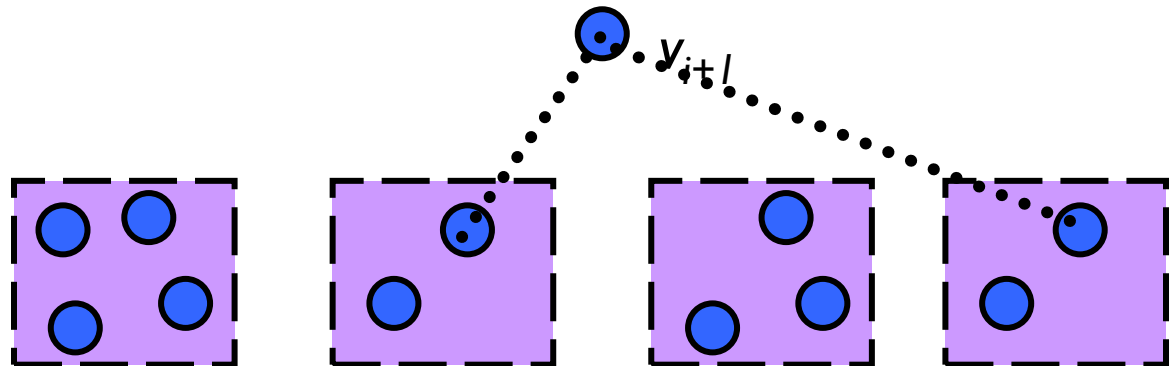
create new component



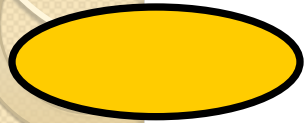
find one edge to each component



> 0

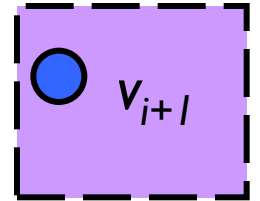
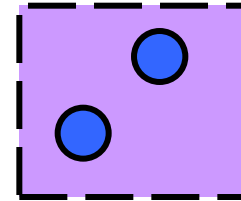
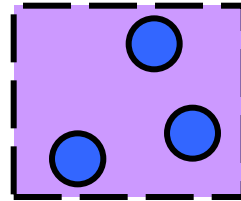
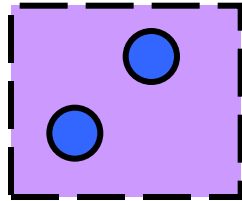
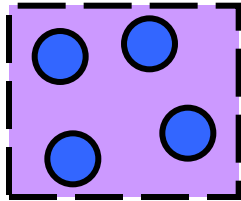


Partition



= 0

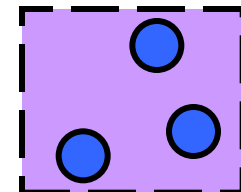
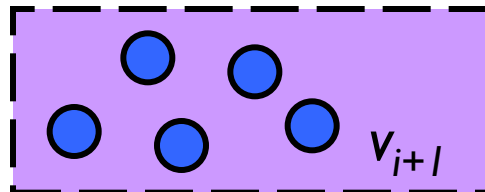
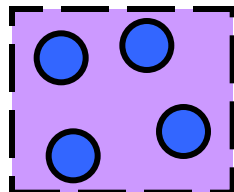
create new component



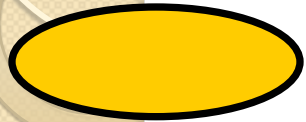
merge and continue



> 0

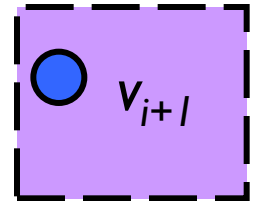
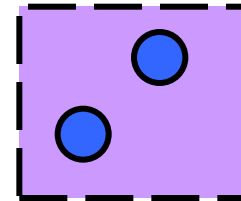
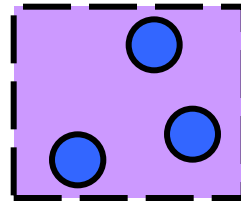
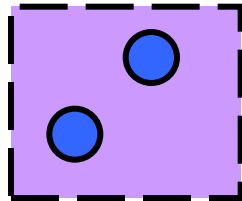
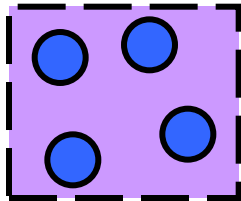


Partition



= 0

create new component

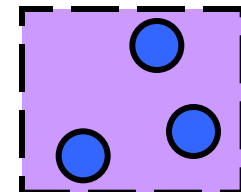
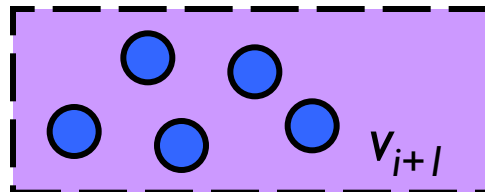
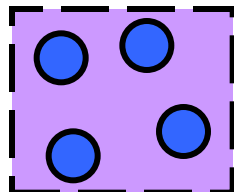


merge and continue

we never add cycles



> 0

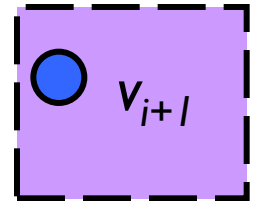
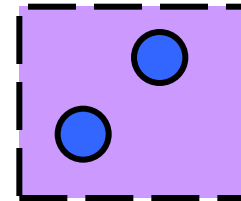
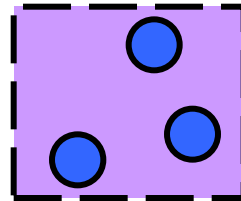
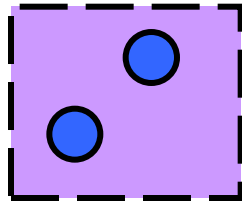
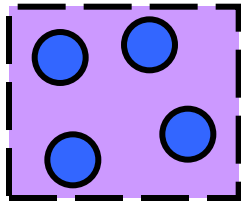


Partition



= 0

create new component

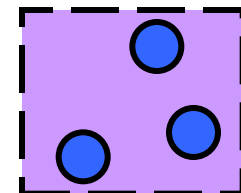
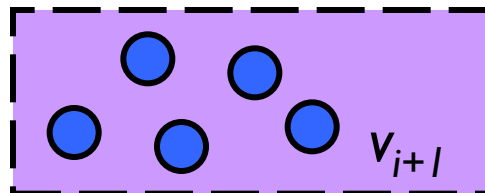
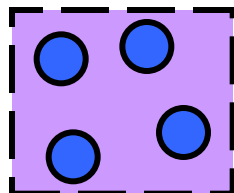


merge and continue



> 0

*algorithm creates a spanning forest.
logn queries per added edge.*

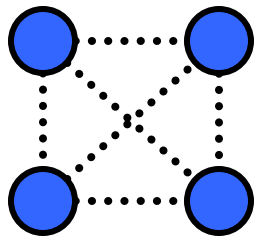


Partition with EC

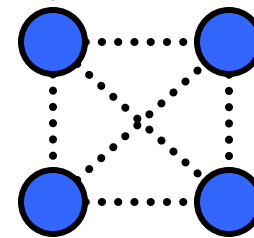
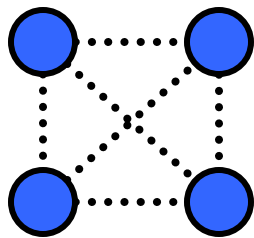
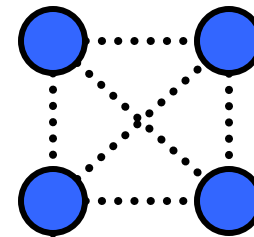
- Algorithm – upper bound:
 - $O(n \log n)$ queries are sufficient
- Information theoretic lower bound:
 - # partitions = B_n
 - $\log B_n = n \log n$
 - Each query gives $\log(n^2) = 2 \log n$ bits
 - Need $\Omega(n)$ queries.

Partition: ED Lowerbound

- $\Omega(n^2)$ ED queries:



Vs.



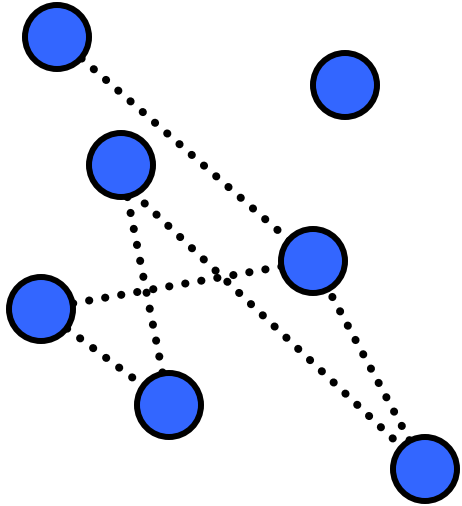
If $|S| > 2$, $ED(S) = 1$

Plan of Talk

- **Graph Learning and Verification**
 - introduce ED and EC queries
 - learning partition with EC and ED queries
 - **verifying with EC queries**
- **Circuit Learning with Value Injection**
 - introduce value injection model
 - hardness result for large alphabet circuits
 - poly time algorithm for transitively reduced circuits
- **Summary**

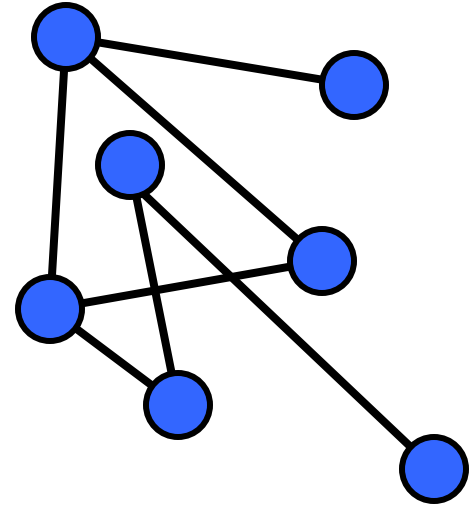
Verification

G



=?

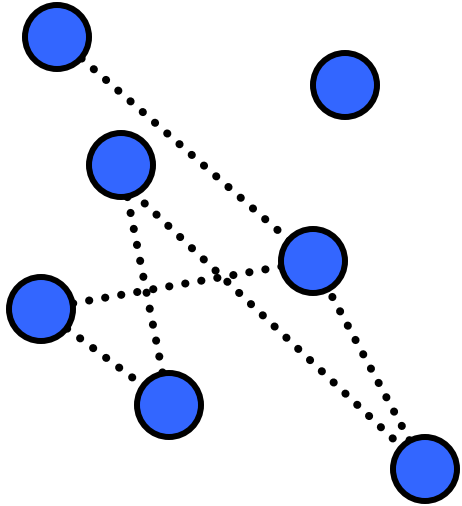
H



motivation: check for errors in learning

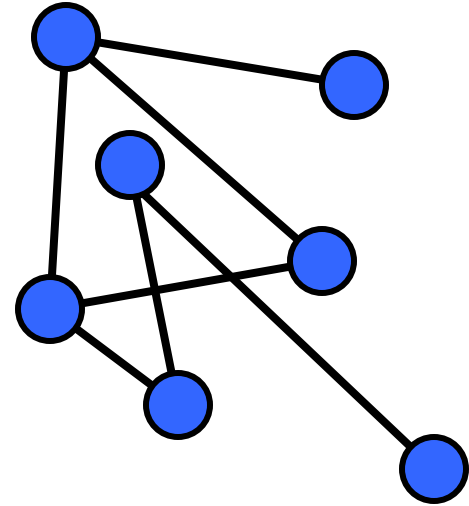
Verification

G



=?

H



no harder than learning

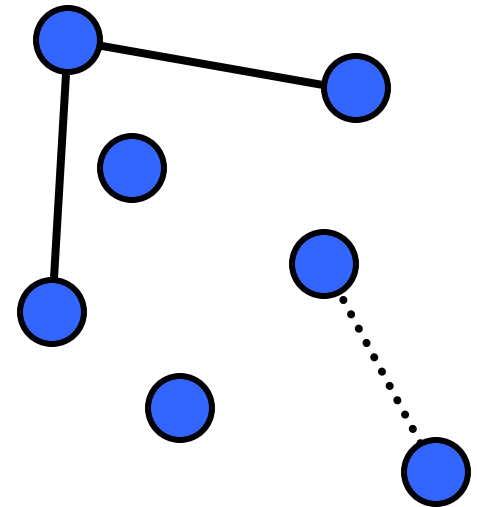
Verification

Thm: If $G \neq H$ then with probability $\frac{1}{4}$
 $EC_G(S) \neq EC_H(S)$ for a random subset S

Verification

Thm: If $G \neq H$ then with probability $\frac{1}{4}$
 $EC_G(S) \neq EC_H(S)$ for a random subset S

Pf: If $G \Delta H \neq \emptyset$, then with prob. $\frac{1}{4}$ $G \Delta H(S)$
has an **odd** number of edges.

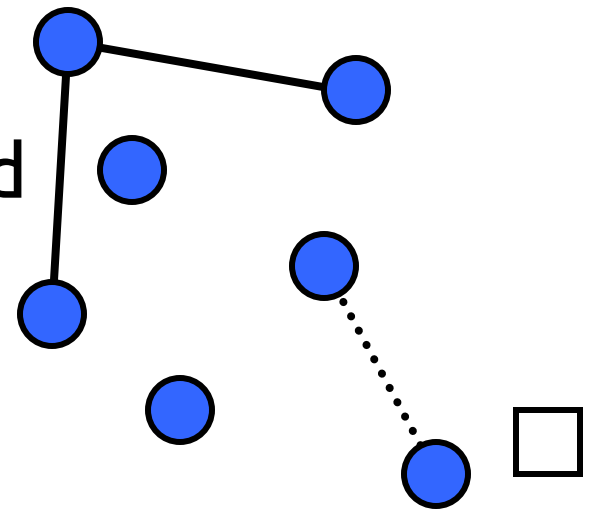


Verification

Thm: If $G \neq H$ then with probability $1/4$
 $EC_G(S) \neq EC_H(S)$ for a random subset S

Pf: If $G \Delta H \neq \emptyset$, then with prob. $1/4$ $G \Delta H(S)$
has an **odd** number of edges.

So $EC_G(S) + EC_H(S)$ is odd
 $\rightarrow EC_G(S) \neq EC_H(S)$



Verification

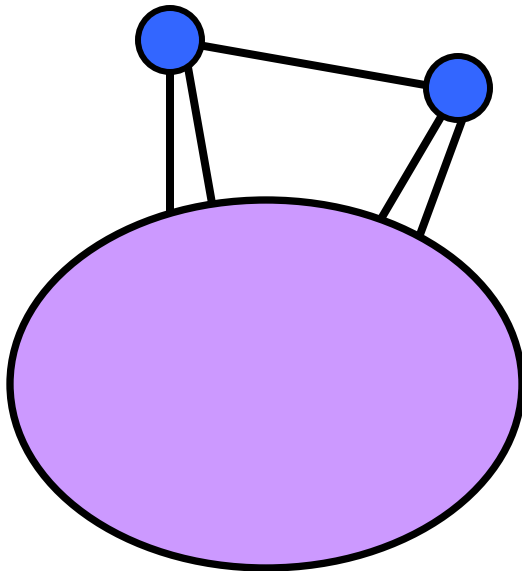
- **Lemma:** A random subset of vertices of a non-empty graph induces an **odd** number of edges w.p. at least $1/4$.

Verification

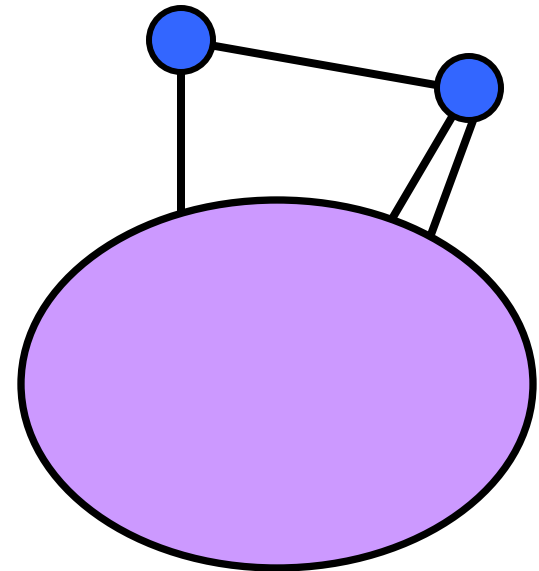
- **Pf of Lemma:** Order the vertices $\{v_1 \dots v_n\}$ so (v_{n-1}, v_n) is an edge. Choose in order with Pr. $1/2$.

Verification

- **Pf of Lemma:** Order the vertices $\{v_1 \dots v_n\}$ so $(v_{n-1}, v_n) \notin E$. Choose in order with Pr. $1/2$.

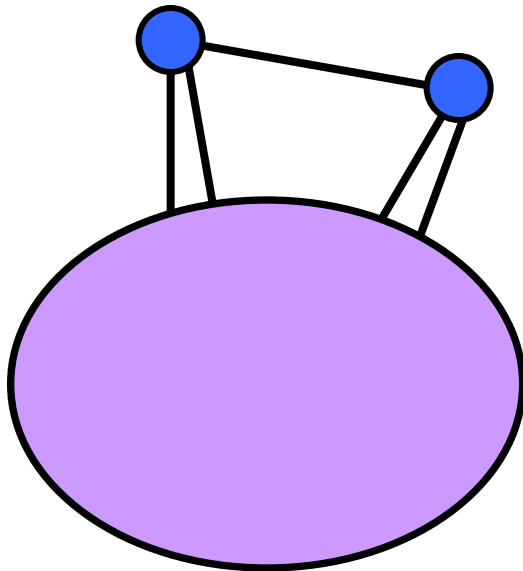


2 cases



Verification

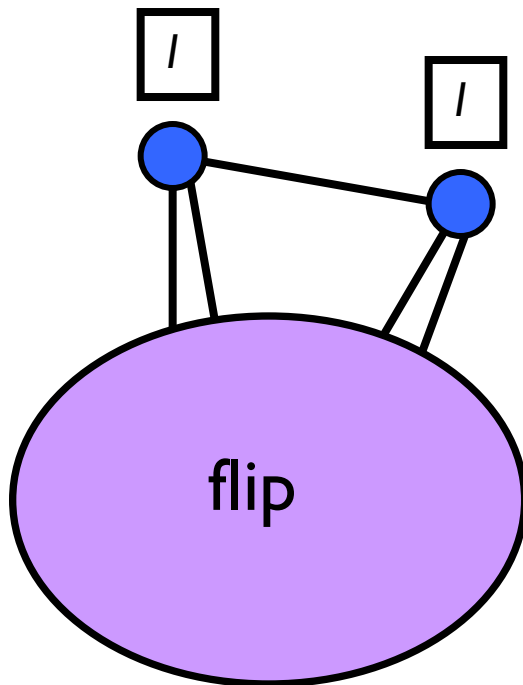
- **Pf of Lemma:** Order the vertices $\{v_1 \dots v_n\}$ so $(v_{n-1}, v_n) \notin E$. Choose in order with Pr. $1/2$.



even case

Verification

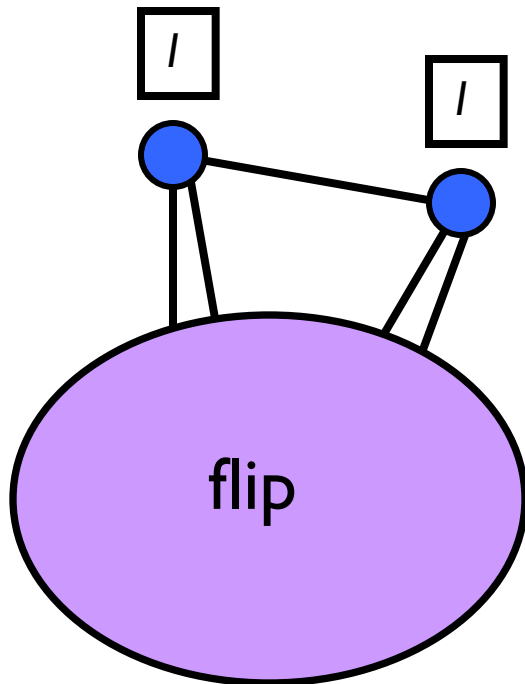
- **Pf of Lemma:** Order the vertices $\{v_1 \dots v_n\}$ so $(v_{n-1}, v_n) \notin E$. Choose in order with Pr. $1/2$.



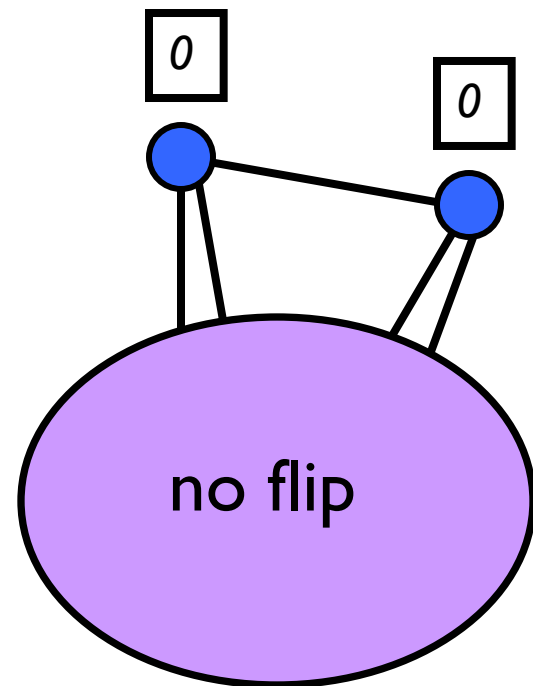
even case

Verification

- **Pf of Lemma:** Order the vertices $\{v_1 \dots v_n\}$ so $(v_{n-1}, v_n) \in E$. Choose in order with Pr. $1/2$.

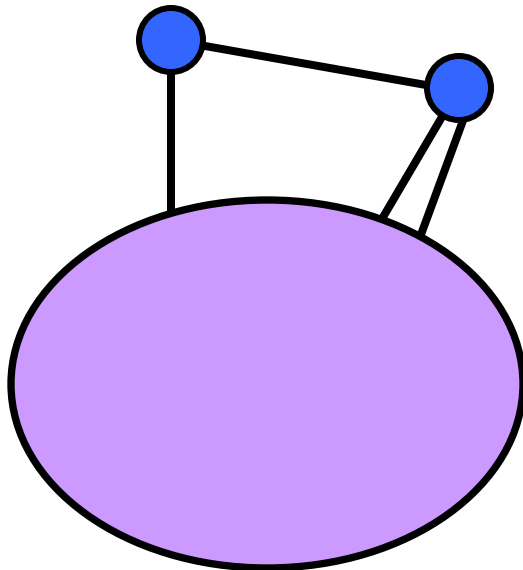


even case



Verification

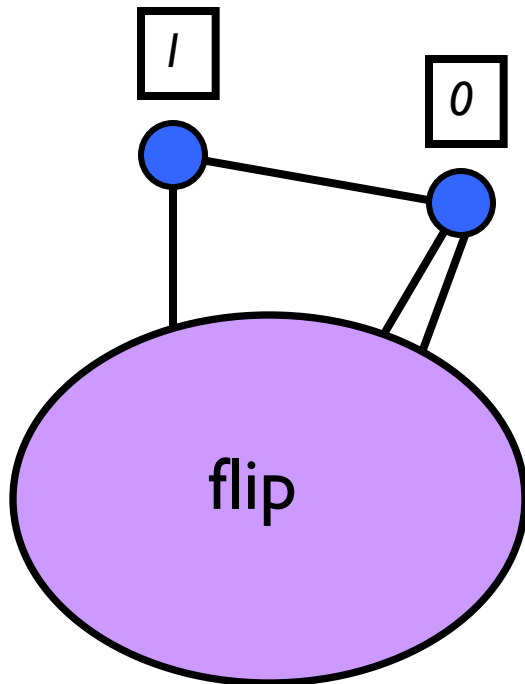
- **Pf of Lemma:** Order the vertices $\{v_1 \dots v_n\}$ so $(v_{n-1}, v_n) \notin E$. Choose in order with Pr. $1/2$.



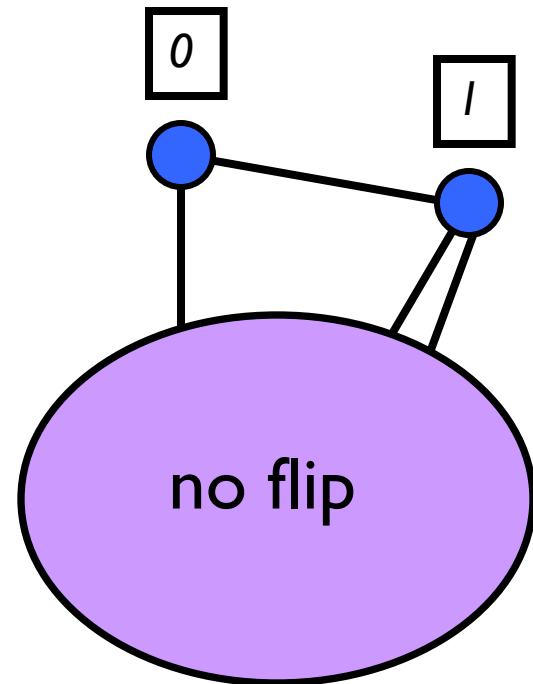
odd case

Verification

- **Pf of Lemma:** Order the vertices $\{v_1 \dots v_n\}$ so $(v_{n-1}, v_n) \in E$. Choose in order with Pr. $1/2$.

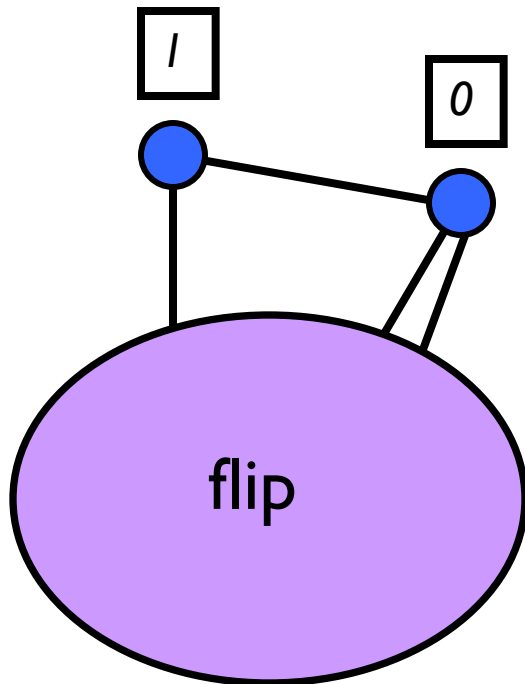


odd case

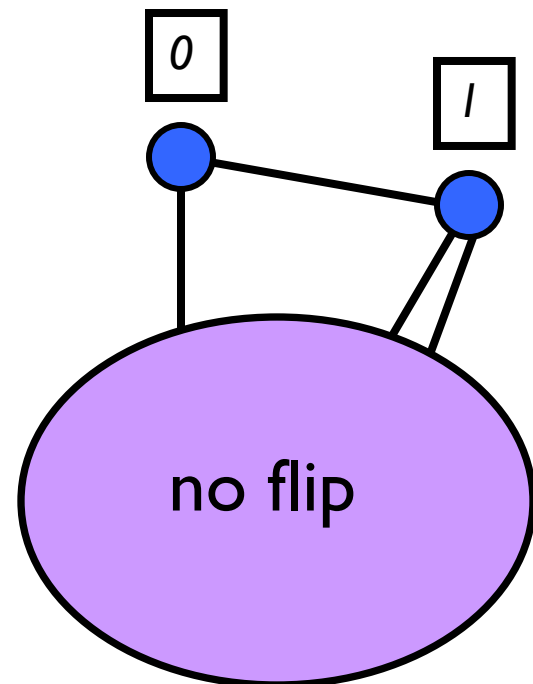


Verification

- **Pf of Lemma:** Order the vertices $\{v_1 \dots v_n\}$ so $(v_{n-1}, v_n) \in E$. Choose in order with Pr. $1/2$.



odd case



Fingerprinting

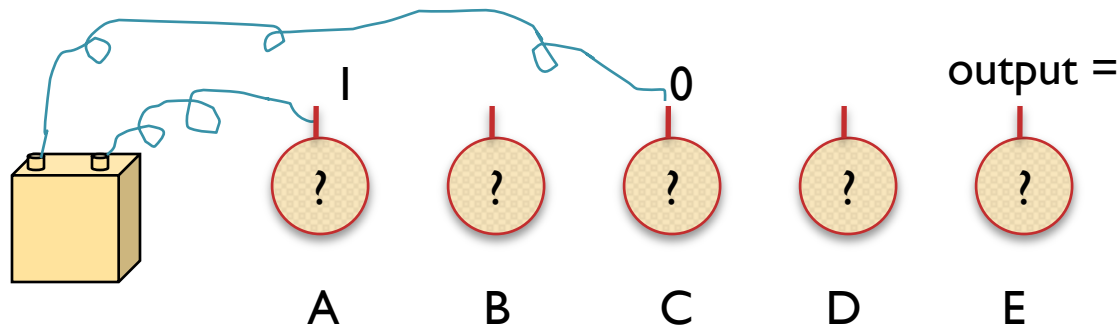
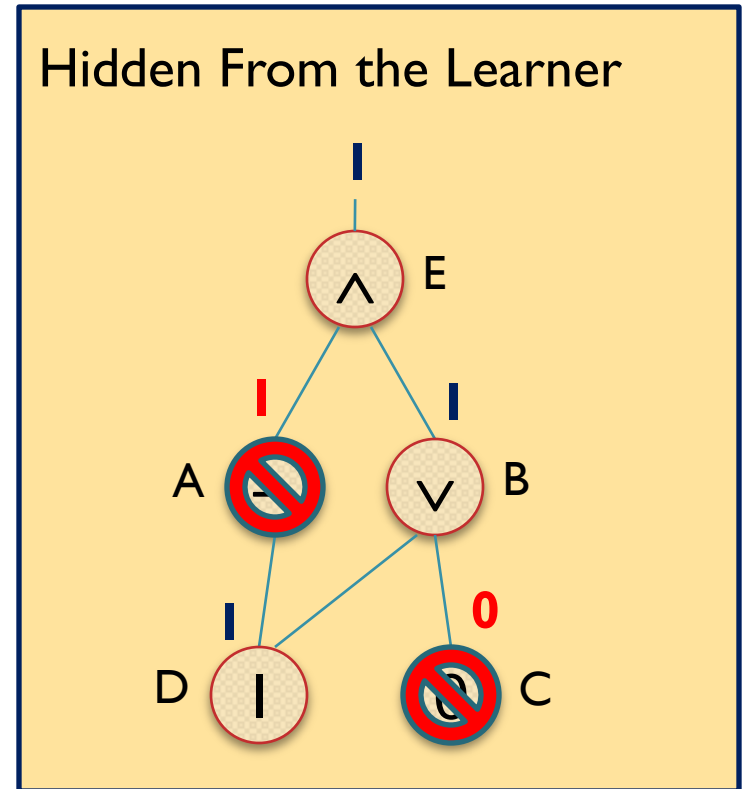
- Freivalds' Fingerprinting:
 - If $A \neq B$ then for random x , with prob. $\frac{1}{2}$ $Ax \neq Bx$
 - Corollary: If $A \neq B$ then with prob. $\frac{1}{4}$ $x^T A y \neq x^T B y$ for random independent x, y .
- This result:
 - If $A^T + A \neq B^T + B$ then $x^T A x \neq x^T B x$ with prob. $\frac{1}{4}$.
 - Weighted, directed graphs.

Plan of Talk

- Graph Learning and Verification
 - introduce ED and EC queries
 - learning partition with EC and ED queries
 - verifying with EC queries
- Circuit Learning with Value Injection
 - introduce value injection model
 - hardness result for large alphabet circuits
 - poly time algorithm for transitively reduced circuits
- Summary

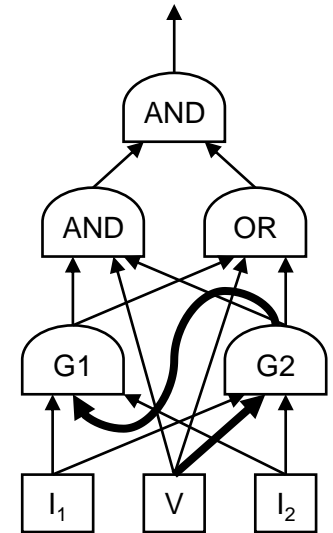
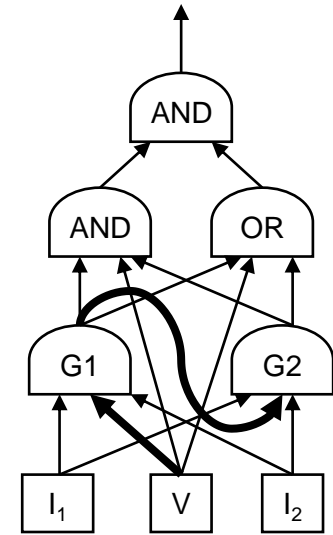
The Value Injection Query Model

- Introduced [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



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.
[AACW '06] model	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)

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.

What Happens For Large-Alphabet Circuits?

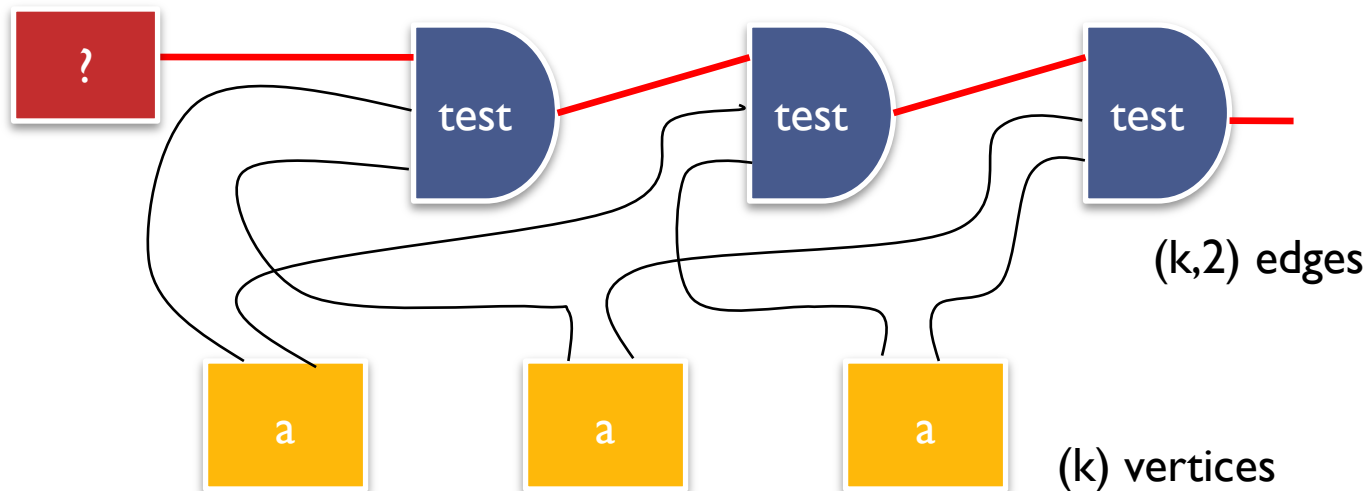
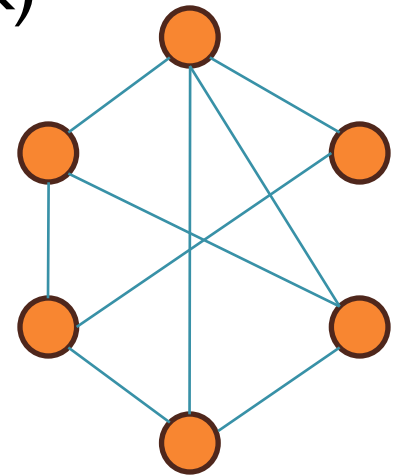
- There is evidence that learning log depth, constant fan-in large-alphabet circuits may be computationally intractable
- **Transitively reduced** and **bounded shortcut width** circuits can be learned in time polynomial in the number of wires and the alphabet size.
 - We can approximately learn bounded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

Plan of Talk

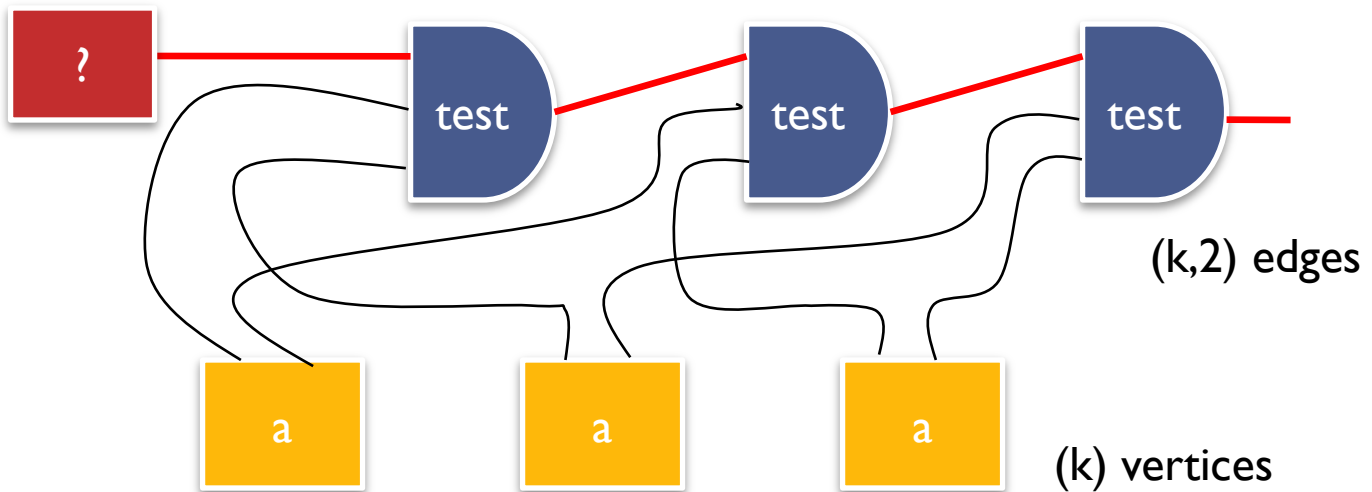
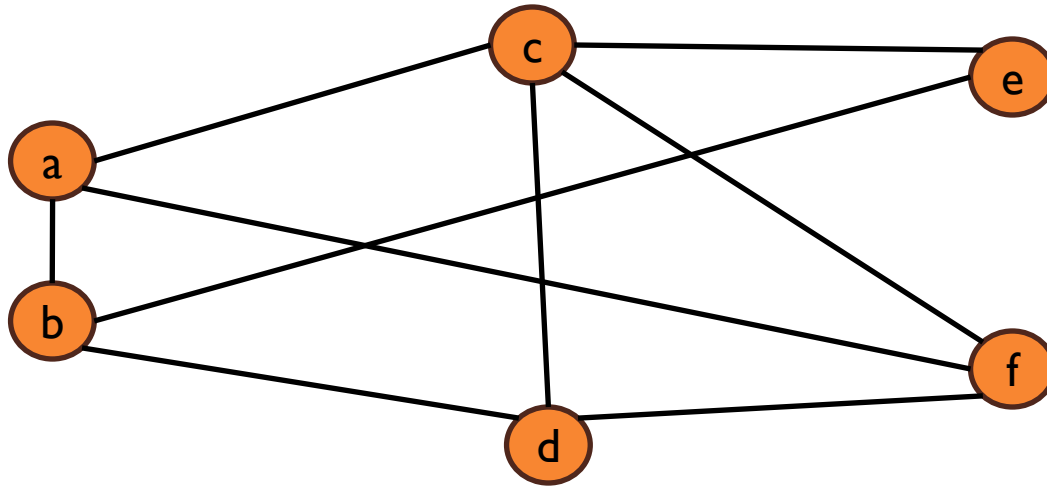
- Graph Learning and Verification
 - introduce ED and EC queries
 - learning partition with EC and ED queries
 - verifying with EC queries
- Circuit Learning with Value Injection
 - introduce value injection model
 - **hardness result for large alphabet circuits**
 - poly time algorithm for transitively reduced circuits
- Summary

Hardness of Learning Large Alphabet Circuits

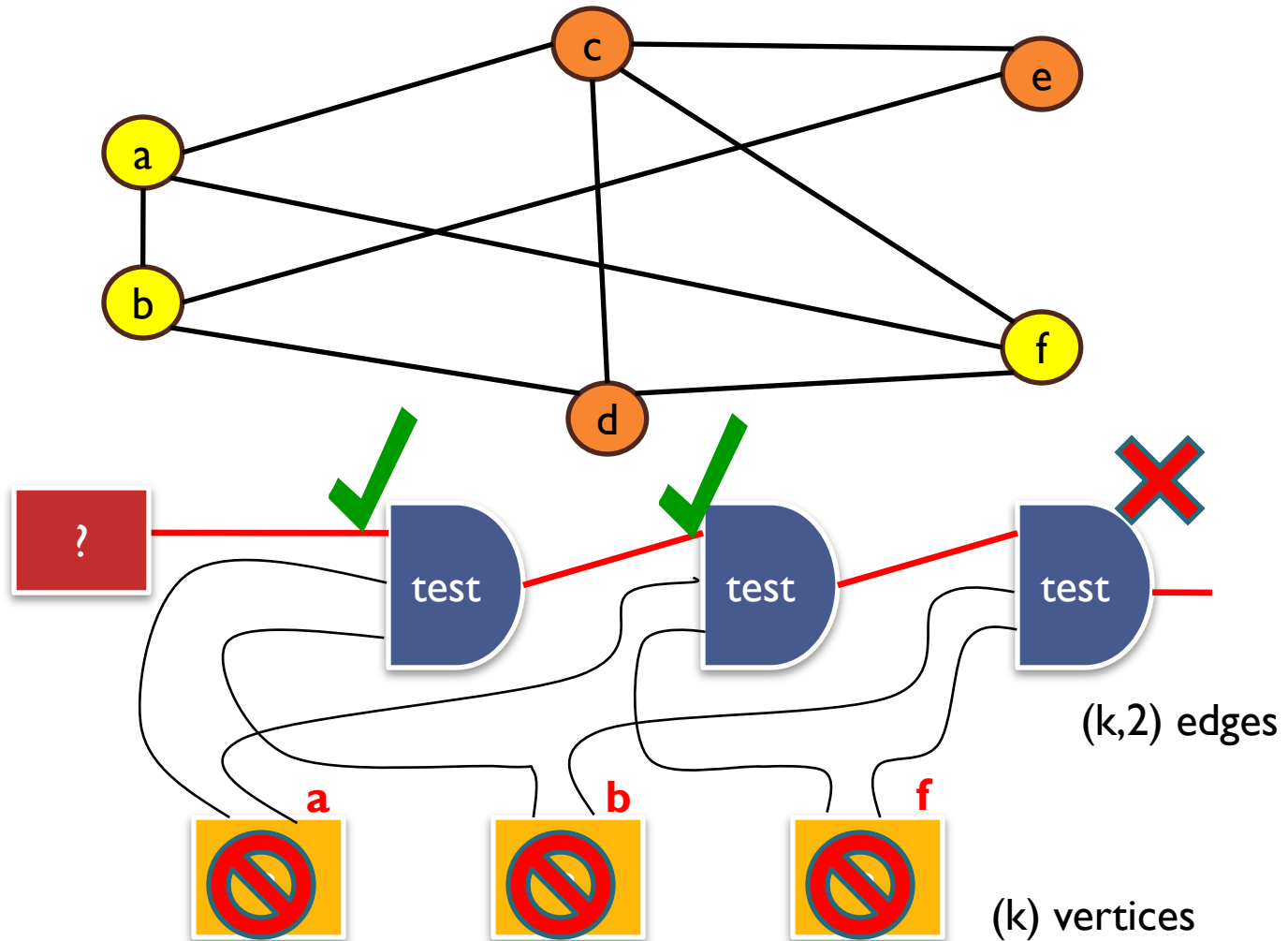
- Consider the problem on input (G,k) of telling whether the graph G on n vertices has a clique of size k
- We give a reduction that turns a large-alphabet circuit learning algorithm into a clique tester



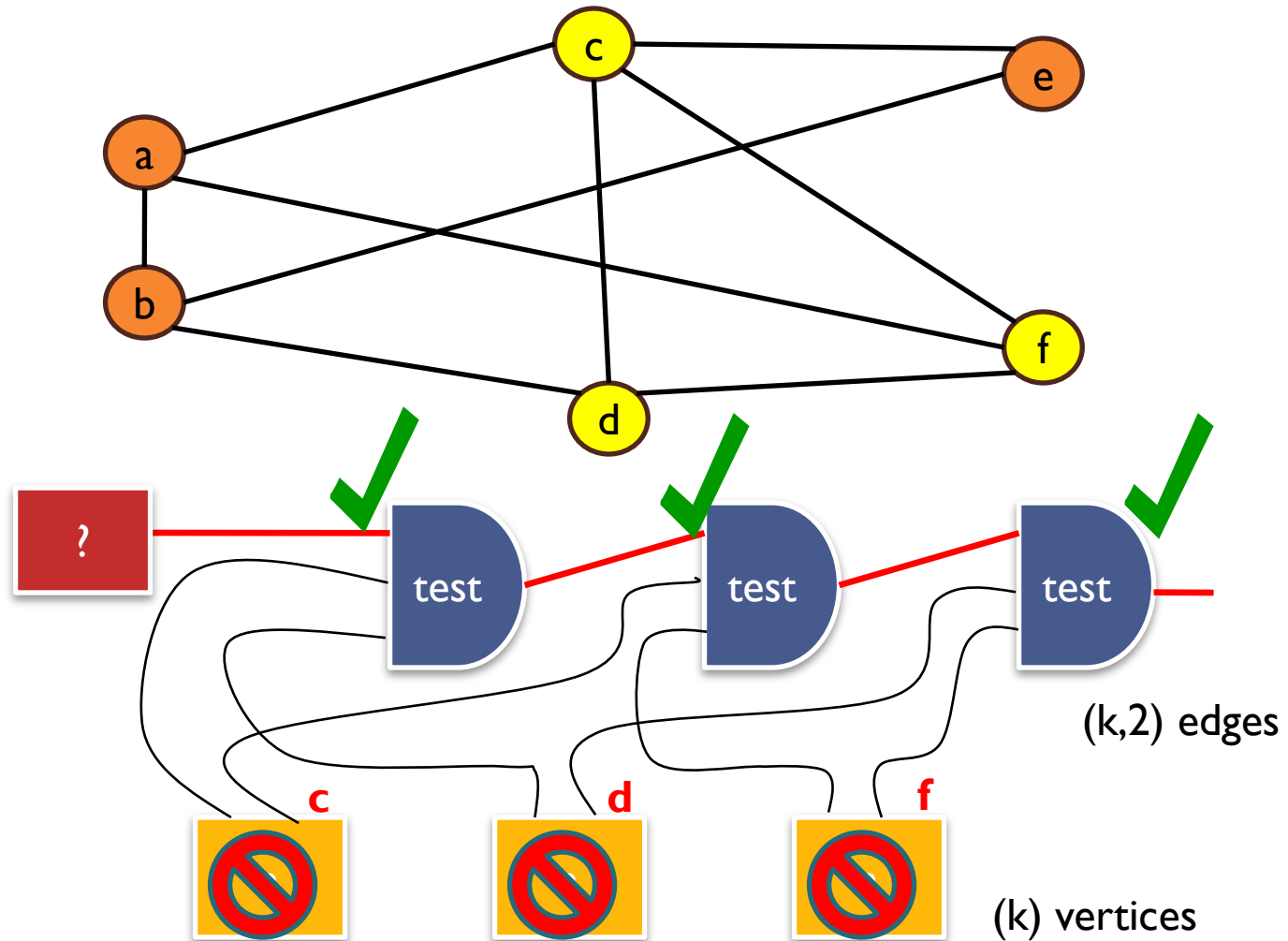
Reducing the Clique Problem to Circuit Learning



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

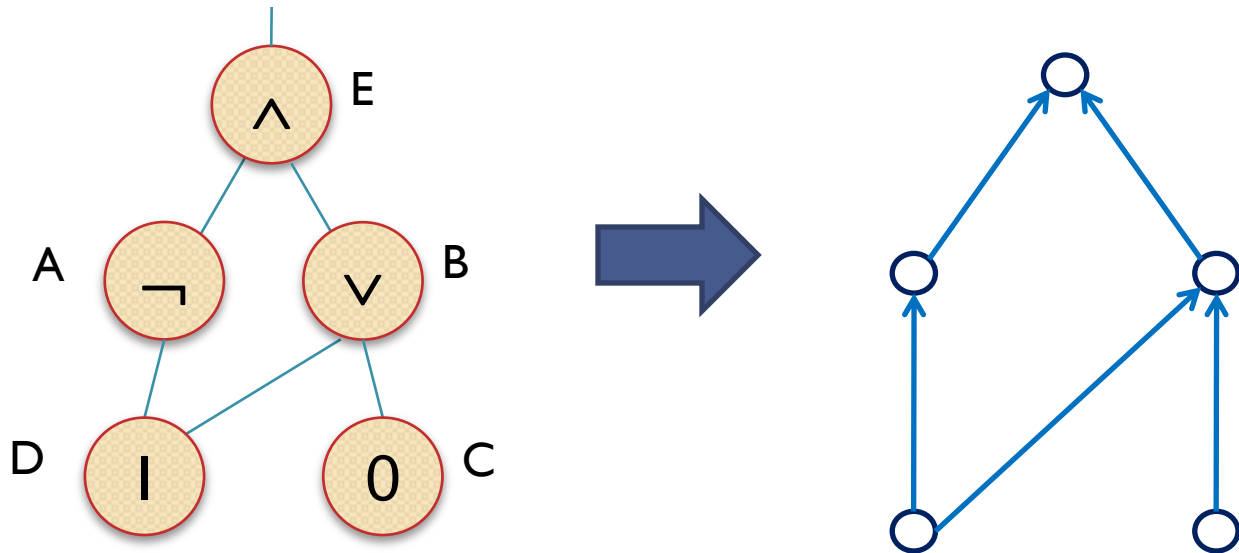
This motivates looking at classes of large-alphabet circuits with restricted topology

Plan of Talk

- Graph Learning and Verification
 - introduce ED and EC queries
 - learning partition with EC and ED queries
 - verifying with EC queries
- Circuit Learning with Value Injection
 - introduce value injection model
 - hardness result for large alphabet circuits
 - poly time algorithm for transitively reduced circuits
- Summary

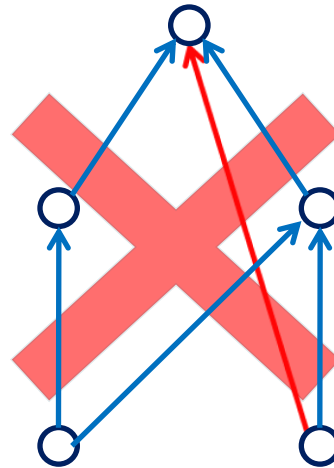
A Circuit's Underlying Graph

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



Transitively Reduced Circuits

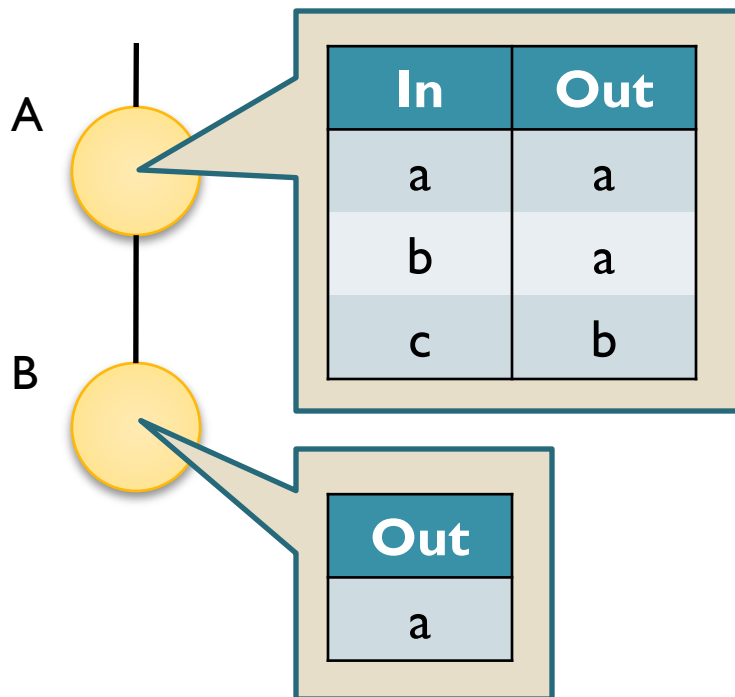
A circuit is transitively reduced if its underlying directed graph has no shortcuts. If (u,v) is an edge and there is a path of length ≥ 2 from u to v , then (u,v) is a **shortcut edge**



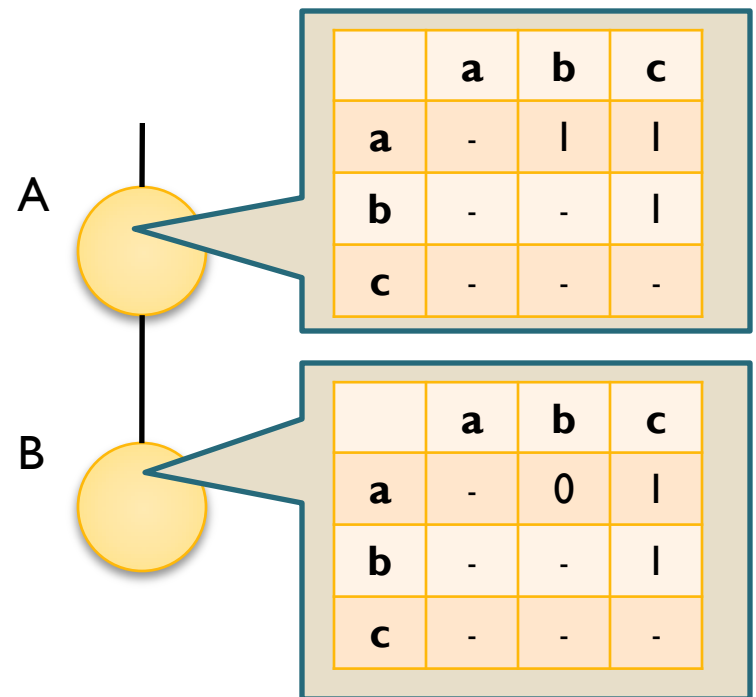
Distinguishing Tables

- 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.”

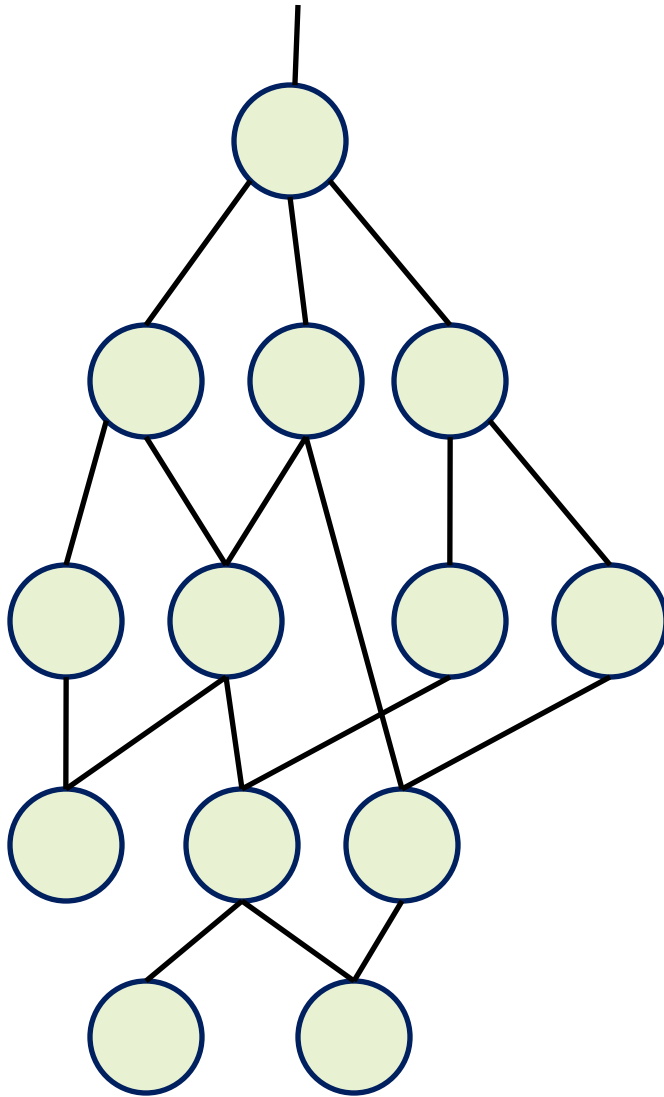
Gate functions



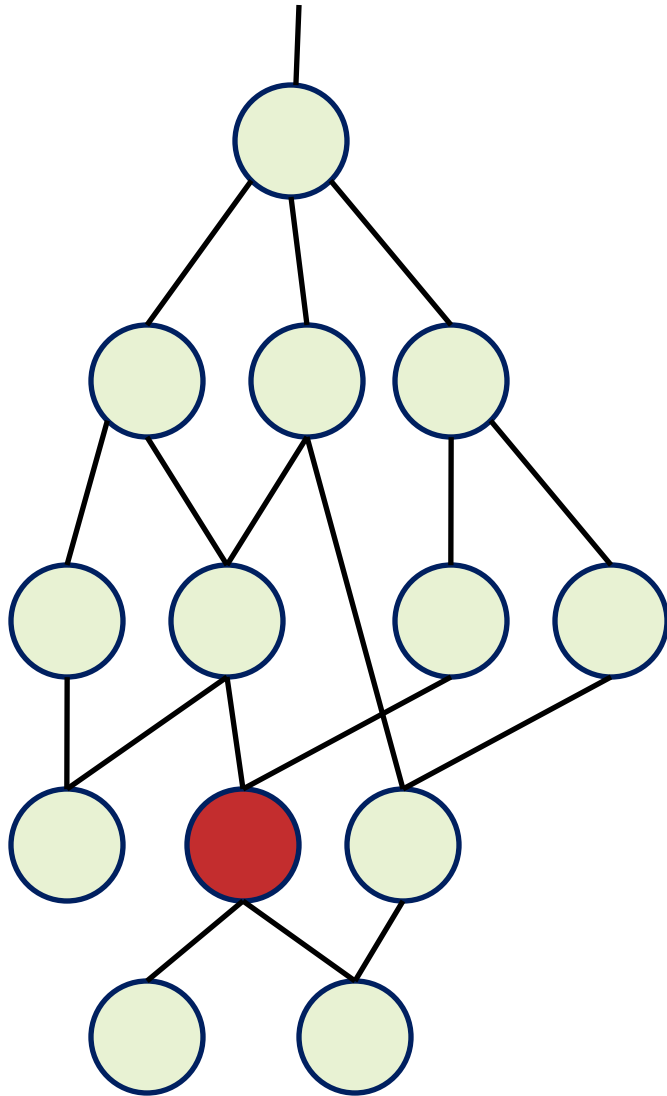
Distinguishing Tables



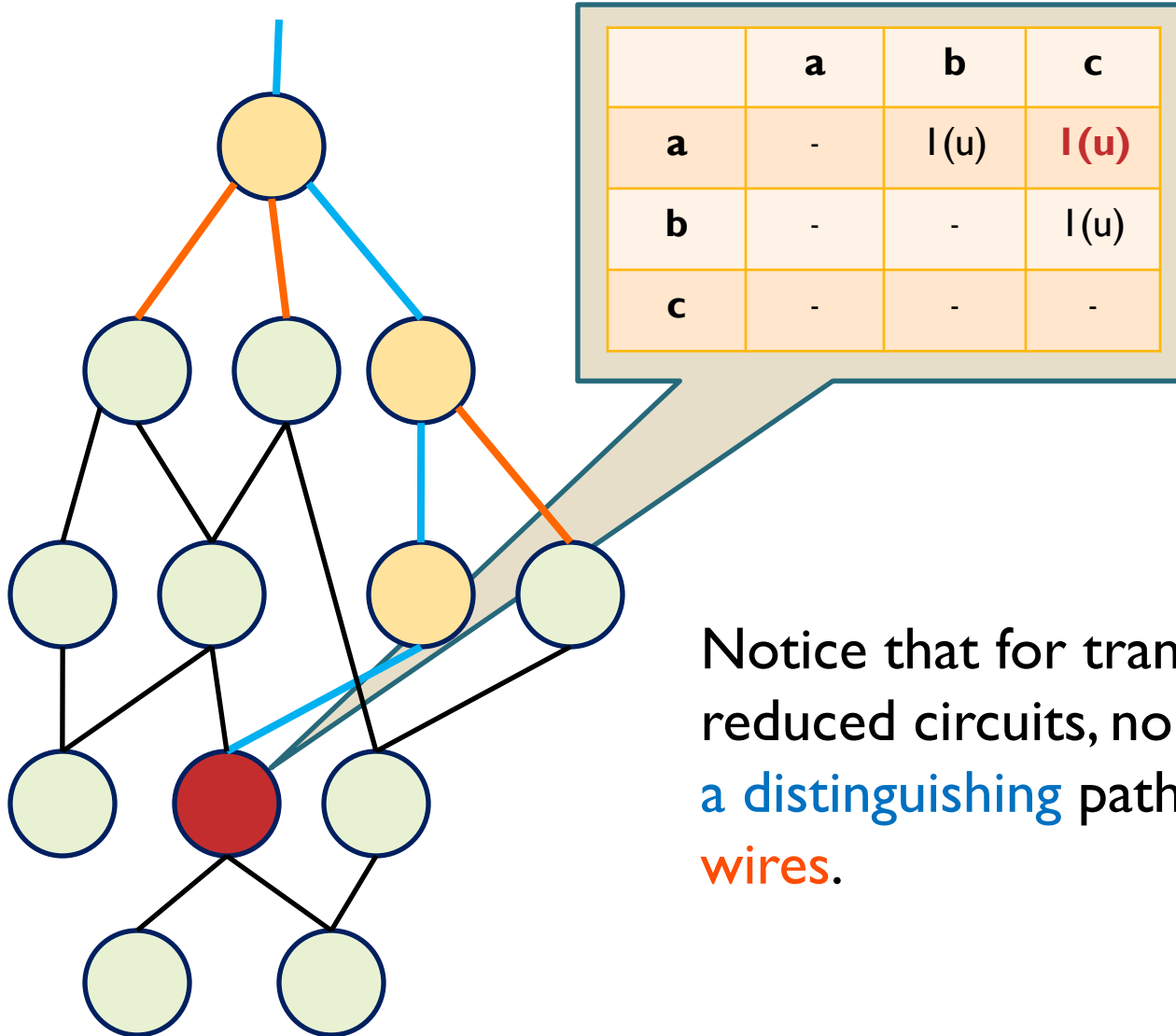
Distinguishing Paths



Distinguishing Paths

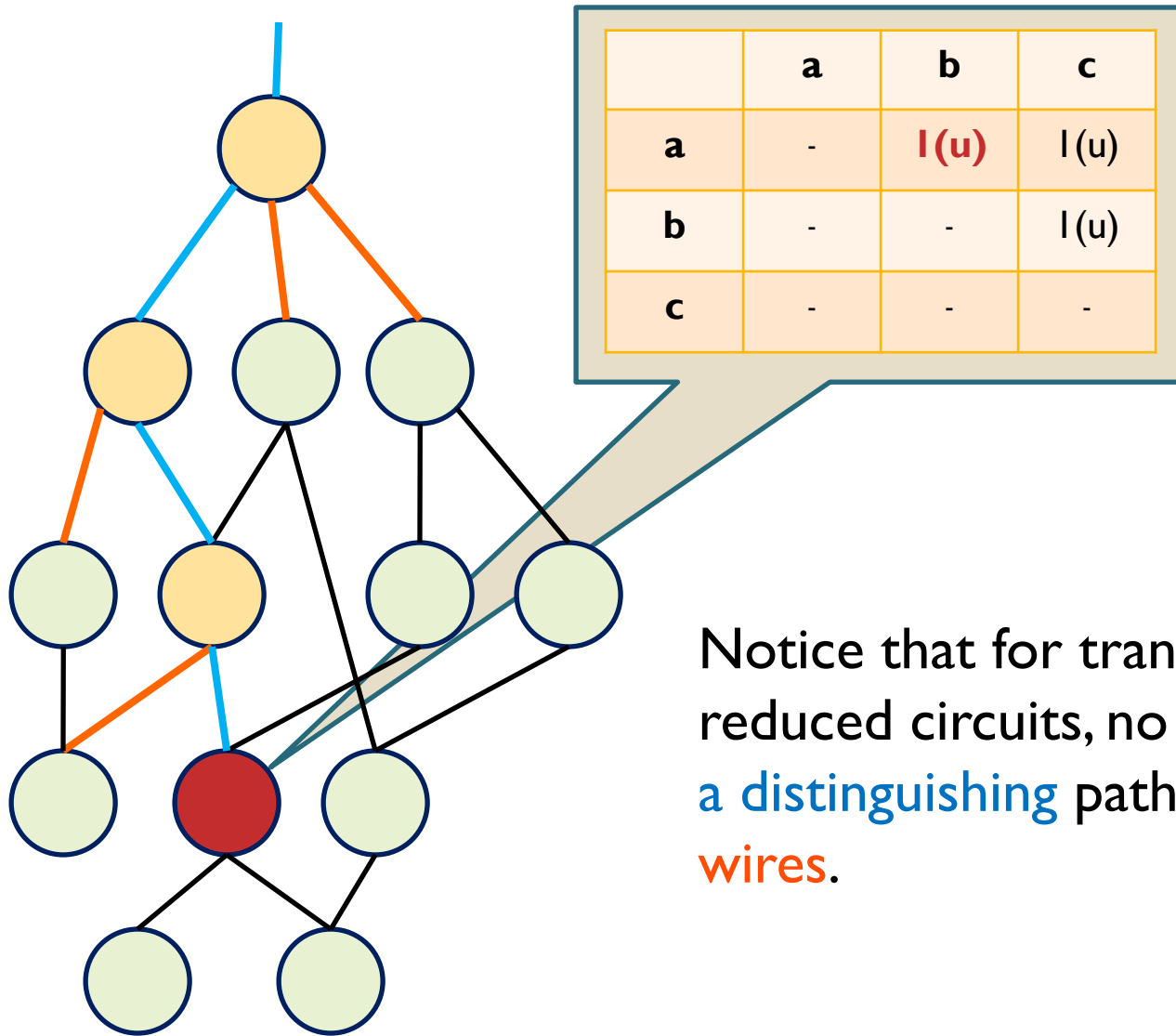


Distinguishing Paths



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

Distinguishing Paths

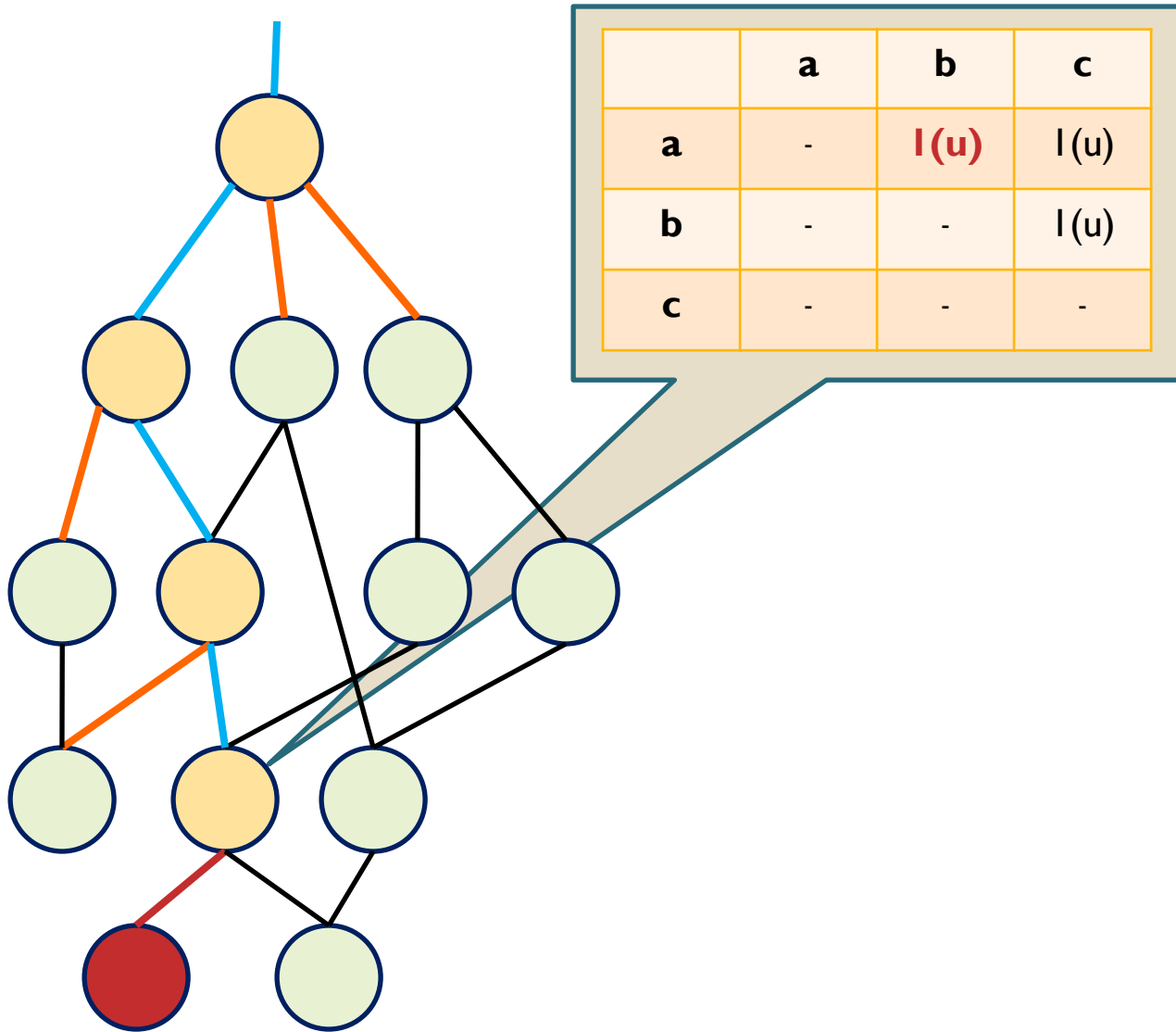


Notice that for transitively reduced circuits, no **wires** along a distinguishing path are **side wires**.

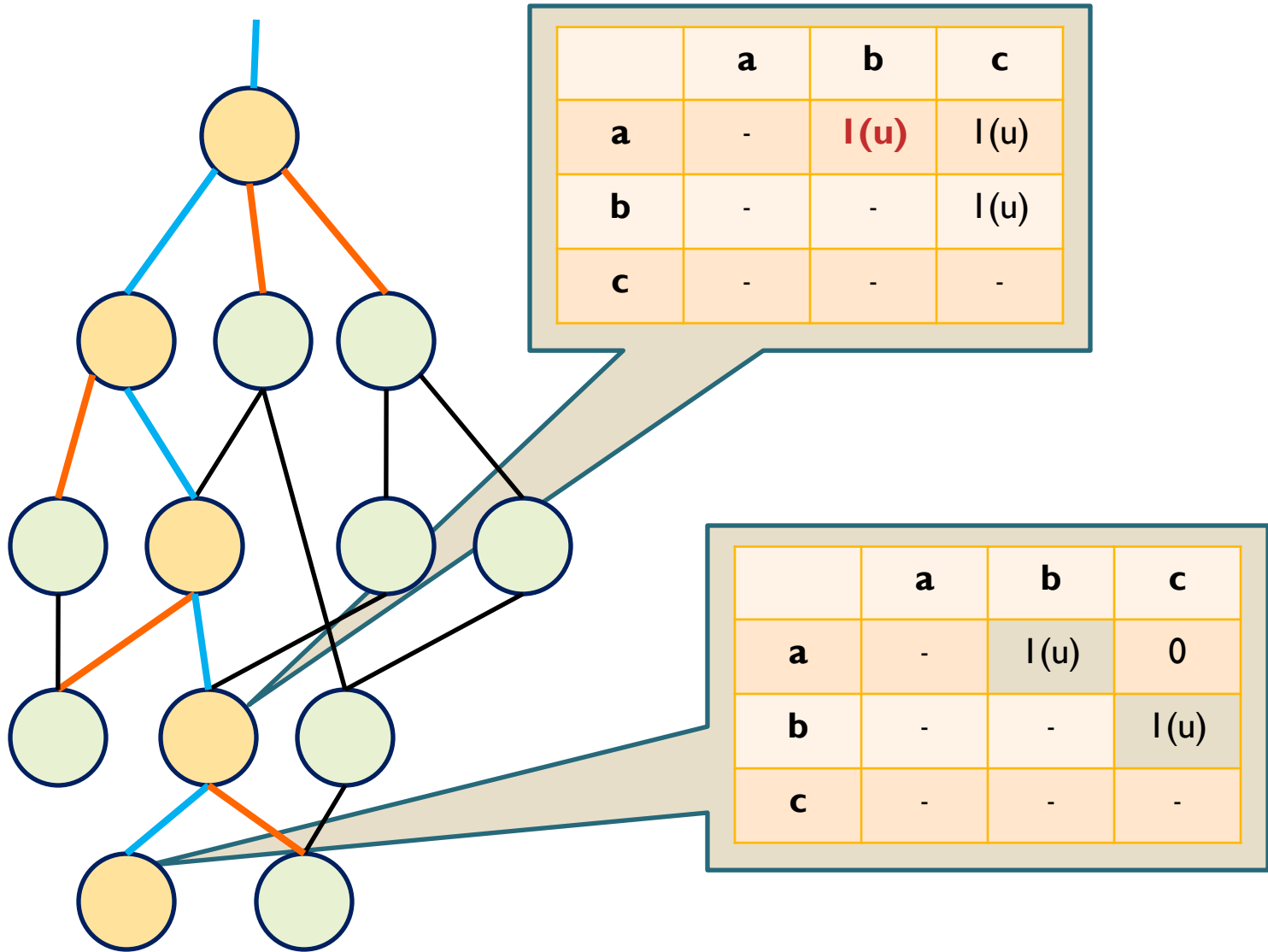
The Distinguishing Paths Algorithm (Outline)

- 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

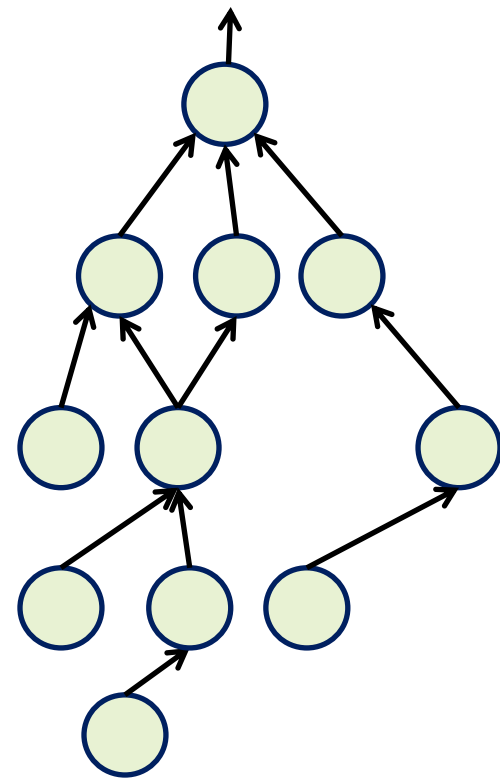
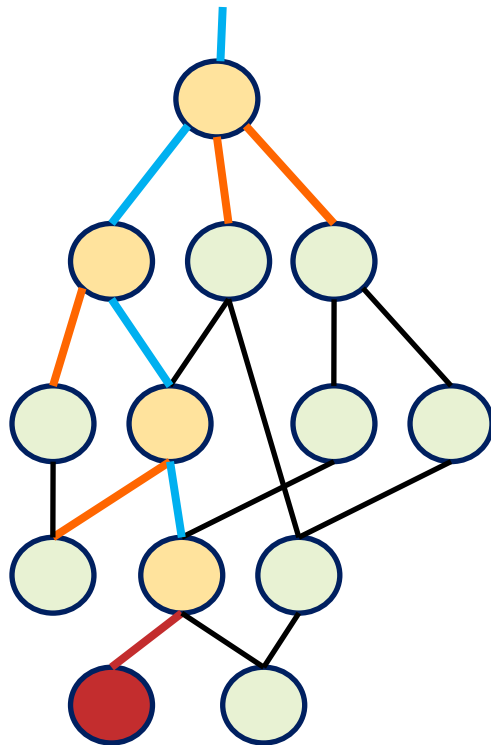


Find Inputs and Extend Paths



Reconstructing Transitively Reduced Circuits

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

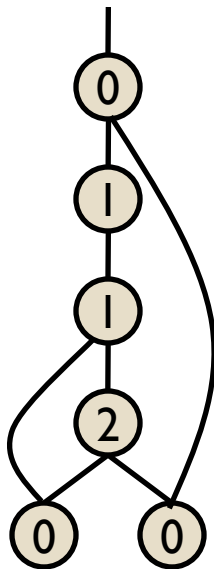


Reconstructing Transitively Reduced Circuits

- 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

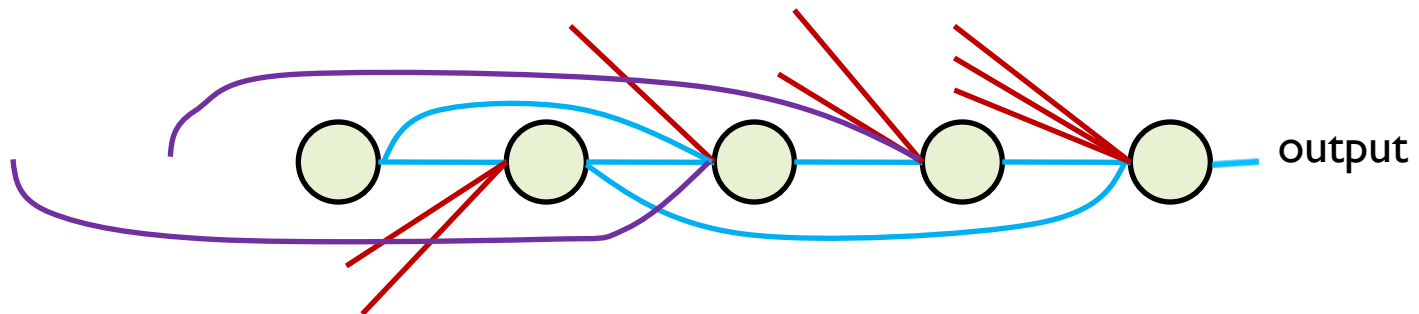
- Generalization of transitive reduction.
- The shortcut width of a wire w_i is the number of wires w_j such that w_j is both an ancestor of w_i and an input of a descendent of w_i .
 - Transitively reduced = shortcut width 0.



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 generalize the notion of distinguishing tables to include cut wires.

Learning Circuits of Bounded Shortcut Width

- When all I 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 Lipschitz 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, Lipschitz gate functions, and shortcut width bounded by a constant.**

Plan of Talk

- **Graph Learning and Verification**
 - introduce ED and EC queries
 - learning partition with EC and ED queries
 - verifying with EC queries
- **Circuit Learning with Value Injection**
 - introduce value injection model
 - hardness result for large alphabet circuits
 - poly time algorithm for transitively reduced circuits
- **Summary**

Graphs

- Nice combinatorial problem with real world applications
- Edge Counting gives more power than Edge Detecting
- Asymptotic gap of $\log(n)$ for partition w/ EC queries
- Can you learn any graph in $O(E)$ EC queries?
- Can you verify trees with ED queries in $o(n)$ queries? Other classes?
- Costs depending on query size
- Mixing queries

Circuits

- Also a nice combinatorial problem with real world applications
- Learnability lines can be quite different with small changes to the problem
- How about probabilistic circuits (Bayesian networks)?
- Social Networks
- Quantum Circuits...