Learning Large-Alphabet and Analog Circuits with Value Injection Queries

Dana Angluin[1], James Aspnes[1,⋆], Jiang Chen[2,⋆⋆], and Lev Reyzin[1,⋆⋆⋆]

[1] Computer Science Department, Yale University
{angluin,aspnes}@cs.yale.edu, lev.reyzin@yale.edu
[2] Center for Computational Learning Systems, Columbia University
criver@cs.columbia.edu

Abstract. We consider the problem of learning an acyclic discrete circuit with n wires, fan-in bounded by k and alphabet size s using value injection queries. For the class of transitively reduced circuits, we develop the Distinguishing Paths Algorithm, that learns such a circuit using $(ns)^{O(k)}$ value injection queries and time polynomial in the number of queries. We describe a generalization of the algorithm to the class of circuits with shortcut width bounded by b that uses $(ns)^{O(k+b)}$ value injection queries. Both algorithms use value injection queries that fix only $O(kd)$ wires, where d is the depth of the target circuit. We give a reduction showing that without such restrictions on the topology of the circuit, the learning problem may be computationally intractable when $s = n^{\Theta(1)}$, even for circuits of depth $O(\log n)$. We then apply our large-alphabet learning algorithms to the problem of approximate learning of analog circuits whose gate functions satisfy a Lipschitz condition. Finally, we consider models in which behavioral equivalence queries are also available, and extend and improve the learning algorithms of [5] to handle general classes of gates functions that are polynomial time learnable from counterexamples.

1 Introduction

We consider learning large-alphabet and analog acyclic circuits in the value injection model introduced in [5]. In this model, we may inject values of our choice on any subset of wires, but we can only observe the one output of the circuit. However, the value injection query algorithms in that paper for boolean and constant alphabet networks do not lift to the case when the size of the alphabet is polynomial in the size of the circuit.

One motivation for studying the boolean network model includes gene regulatory networks. In a boolean model, each node in a gene regulatory network can represent a gene whose state is either active or inactive. However, genes may have a large number of states of activity. Constant-alphabet network models

⋆ Supported in part by NSF grant CNS-0435201.
⋆⋆ Supported in part by a research contract from Consolidated Edison.
⋆⋆⋆ Supported by a Yahoo! Research Kern Family Scholarship.

N. Bshouty and C. Gentile (Eds.): COLT 2007, LNAI 4539, pp. 51–65, 2007.

may not adequately capture the information present in these networks, which motivates our interest in larger alphabets.

Akutsu et al. [2] and Ideker, Thorsson, and Karp [9] consider the discovery problem that models the experimental capability of gene disruption and overexpression. In such experiments, it is desirable to manipulate as few genes as possible. In the particular models considered in these papers, node states are fully observable – the gene expression data gives the state of every node in the network at every time step. Their results show that in this model, for bounded fan-in or sufficiently restricted gene functions, the problem of learning the structure of a network is tractable.

In contrast, there is ample evidence that learning boolean circuits solely from input-output behaviors may be computationally intractable. Kearns and Valiant [12] show that specific cryptographic assumptions imply that **NC1** circuits and **TC0** circuits are not PAC learnable in polynomial time. These negative results have been strengthened to the setting of PAC learning with membership queries [6], even with respect to the uniform distribution [13]. Furthermore, positive learnability results exist only for fairly limited classes, including propositional Horn formulas [3], general read once Boolean formulas [4], and decision trees [7], and those for specific distributions, including **AC0** circuits [14], DNF formulas [10], and **AC0** circuits with a limited number of majority gates [11].[1]

Thus, Angluin et al. [5] look at the relative contributions of full observation and full control of learning boolean networks. Their model of value injection allows full control and restricted observation, and it is the model we study in this paper. Interestingly, their results show that this model gives the learner considerably more power than with only input-output behaviors but less than the power with full observation. In particular, they show that with value injection queries, **NC1** circuits and **AC0** circuits are exactly learnable in polynomial time, but their negative results show that depth limitations are necessary.

A second motivation behind our work is to study the relative importance of the parameters of the models for learnability results. The impact of alphabet size on learnability becomes a natural point of inquiry, and ideas from fixed parameter tractability are very relevant [8,15].

2 Preliminaries

2.1 Circuits

We give a general definition of acyclic circuits whose wires carry values from a set Σ. For each nonnegative integer k, a **gate function** of arity k is a function from Σ^k to Σ. A **circuit** C consists of a finite set of wires $w_1, \ldots, w_n$, and for each wire w_i, a gate function g_i of arity k_i and an ordered k_i-tuple $w_{\sigma(i,1)}, \ldots, w_{\sigma(i,k_i)}$ of wires, the **inputs** of w_i. We define w_n to be the **output wire** of the circuit. We may think of wires as outputs of gates in C.

[1] Algorithms in both [14] and [11] for learning **AC0** circuits and their variants run in quasi-polynomial time.

The **unpruned graph** of a circuit C is the directed graph whose *vertices* are the wires and whose *edges* are pairs (w_i, w_j) such that w_i is an input of w_j in C. A wire w_i is **output-connected** if there is a directed path in the unpruned graph from that wire to the output wire. Wires that are not output-connected cannot affect the output value of a circuit. The **graph** of a circuit C is the subgraph of its unpruned graph induced by the output-connected wires.

A circuit is **acyclic** if its graph is acyclic. In this paper we consider only acyclic circuits. If u and v are vertices such that $u \neq v$ and there is a directed path from u to v, then we say that u is an **ancestor** of v and that v is a **descendant** of u. The **depth** of an output-connected wire w_i is the length of a longest path from w_i to the output wire w_n. The depth of a circuit is the maximum depth of any output-connected wire in the circuit. A wire with no inputs is an **input wire**; its **default value** is given by its gate function, which has arity 0 and is constant.

We consider the property of being transitively reduced [1] and a generalization of it: bounded shortcut width. Let G be an acyclic directed graph. An edge (u, v) of G is a **shortcut edge** if there exists a directed path in G of length at least two from u to v. G is **transitively reduced** if it contains no shortcut edges. A circuit is transitively reduced if its graph is transitively reduced.

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 descendant of w_i. (Note that we are counting wires, not edges.) The **shortcut width** of a circuit C is the maximum shortcut width of any output-connected wire in C. A circuit is transitively reduced if and only if it has shortcut width 0. A circuit's shortcut width turns out to be a key parameter in its learnability by value injection queries.

2.2 Experiments on Circuits

Let C be a circuit. An **experiment** e is a function mapping each wire of C to $\Sigma \cup \{*\}$, where $*$ is not an element of Σ. If $e(w_i) = *$, then the wire w_i is **free** in e; otherwise, w_i is **fixed** in e. If e is an experiment that assigns $*$ to wire w, and $\sigma \in \Sigma$, then $e|_{w=\sigma}$ is the experiment that is equal to e on all wires other than w, and fixes w to σ. We define an ordering $\preceq$ on $\Sigma \cup \{*\}$ in which all elements of Σ are incomparable and precede $*$, and lift this to the componentwise ordering on experiments. Then $e_1 \preceq e_2$ if every wire that e_2 fixes is fixed to the same value by e_1, and e_1 may fix some wires that e_2 leaves free.

For each experiment e we inductively define the value $w_i(e) \in \Sigma$, of each wire w_i in C under the experiment e as follows. If $e(w_i) = \sigma$ and $\sigma \neq *$, then $w_i(e) = \sigma$. Otherwise, if the values of the input wires of w_i have been defined, then $w_i(e)$ is defined by applying the gate function g_i to them, that is, $w_i(e) = g_i(w_{\sigma(i,1)}(e), \ldots, w_{\sigma(i,k_i)}(e))$. Because C is acyclic, this uniquely defines $w_i(e) \in \Sigma$ for all wires w_i. We define the value of the circuit to be the value of its output wire, that is, $C(e) = w_n(e)$ for every experiment e.

Let C and C' be circuits with the same set of wires and the same value set Σ. If $C(e) = C'(e)$ for every experiment e, then we say that C and C' are **behaviorally equivalent**. To define approximate equivalence, we assume that

there is a metric d on Σ mapping pairs of values from Σ to a real-valued distance between them. If $d(C(e), C'(e)) \leq \epsilon$ for every experiment e, then we say that C and C' are **ϵ-equivalent**.

We consider two principal kinds of circuits. A **discrete circuit** is a circuit for which the set Σ of wire values is a finite set. An **analog circuit** is a circuit for which $\Sigma = [0, 1]$. In this case we specify the distance function as $d(x, y) = |x - y|$.

2.3 The Learning Problems

We consider the following general learning problem. There is an unknown target circuit C^* drawn from a known class of possible target circuits. The set of wires $w_1, \ldots, w_n$ and the value set Σ are given as input. The learning algorithm may gather information about C^* by making calls to an oracle that will answer value injection queries. In a **value injection query**, the algorithm specifies an experiment e and the oracle returns the value of $C^*(e)$. The algorithm makes a value injection query by listing a set of wires and their fixed values; the other wires are assumed to be free, and are not explicitly listed. The goal of a learning algorithm is to output a circuit C that is either exactly or approximately equivalent to C^*.

In the case of learning discrete circuits, the goal is behavioral equivalence and the learning algorithm should run in time polynomial in n. In the case of learning analog circuits, the learning algorithm has an additional parameter $\epsilon > 0$, and the goal is ϵ-equivalence. In this case the learning algorithm should run in time polynomial in n and $1/\epsilon$. In Section 5.1, we consider algorithms that may use **equivalence queries** in addition to value injection queries.

3 Learning Large-Alphabet Circuits

In this section we consider the problem of learning a discrete circuit when the alphabet Σ of possible values is of size $n^{O(1)}$. In Section 4 we reduce the problem of learning an analog circuit whose gate functions satisfy a Lipschitz condition to that of learning a discrete circuit over a finite value set Σ; the number of values is $n^{\Theta(1)}$ for an analog circuit of depth $O(\log n)$. Using this approach, in order to learn analog circuits of even moderate depth, we need learning algorithms that can handle large alphabets.

The algorithm Circuit Builder [5] uses value injection queries to learn acyclic discrete circuits of unrestricted topology and depth $O(\log n)$ with constant fan-in and constant alphabet size in time polynomial in n. However, the approach of [5] to building a sufficient set of experiments does not generalize to alphabets of size $n^{O(1)}$ because the total number of possible settings of side wires along a test path grows superpolynomially. In fact, we give evidence in Section 3.1 that this problem becomes computationally intractable for an alphabet of size $n^{\Theta(1)}$.

In turn, this negative result justifies a corresponding restriction on the topology of the circuits we consider. We first show that a natural top-down algorithm using value-injection queries learns transitively reduced circuits with arbitrary depth, constant fan-in and alphabet size $n^{O(1)}$ in time polynomial in n. We then

give a generalization of this algorithm to circuits that have a constant bound on their shortcut width. The topological restrictions do not result in trivial classes; for example, every levelled graph is transitively reduced.

3.1 Hardness for Large Alphabets with Unrestricted Topology

We give a reduction that turns a large-alphabet circuit learning algorithm into a clique tester. Because the clique problem is complete for the complexity class $W[1]$ (see [8,15]), this suggests the learning problem may be computationally intractable for classes of circuits with large alphabets and unrestricted topology.

The Reduction. Suppose the input is (G, k), where $k \geq 2$ is an integer and $G = (V, E)$ is a simple undirected graph with $n \geq 3$ vertices, and the desired output is whether G contains a clique of size k. We construct a circuit C of depth $d = \binom{k}{2}$ as follows. The alphabet Σ is V; let v_0 be a particular element of V. Define a gate function g with three inputs s, u, and v as follows: if (u, v) is an edge of G, then the output of g is equal to the input s; otherwise, the output is v_0. The wires of C are $s_1, \ldots, s_{d+1}$ and $x_1, x_2, \ldots, x_k$. The wires x_j have no inputs; their gate functions assign them the default value v_0. For $i = 1, \ldots, d$, the wire s_{i+1} has corresponding gate function g, where the s input is s_i, and the u and v inputs are the i-th pair (x_ℓ, x_m) with $\ell < m$ in the lexicographic ordering. Finally, the wire s_1 has no inputs, and is assigned some default value from $V - \{v_0\}$. The output wire is s_{d+1}.

To understand the behavior of C, consider an experiment e that assigns values from V to each of $x_1, \ldots, x_k$, and leaves the other wires free. The gates g pass along the default value of s_1 as long as the values $e(x_\ell)$ and $e(x_m)$ are an edge of G, but if any of those checks fail, the output value will be v_0. Thus the default value of s_1 will be passed all the way to the output wire if and only if the vertex values assigned to $x_1, \ldots, x_k$ form a clique of size k in G.

We may use a learning algorithm as a clique tester as follows. Run the learning algorithm using C to answer its value-injection queries e. If for some queried experiment e, the values $e(x_1), \ldots, e(x_k)$ form a clique of k vertices in G, stop and output the answer "yes." If the learning algorithm halts and outputs a circuit without making such a query, then output the answer "no." Clearly a "yes" answer is correct, because we have a witness clique. And if there is a clique of size k in G, the learning algorithm must make such a query, because in that case, the default value assigned to s_1 cannot otherwise be learned correctly; thus, a "no" answer is correct. Then we have the following.

Theorem 1. *If for some nonconstant computable function $d(n)$ an algorithm using value injection queries can learn the class of circuits of at most n wires, alphabet size s, fan-in bound 3, and depth bound $d(n)$ in time polynomial in n and s, then there is an algorithm to decide whether a graph on n vertices has a clique of size k in time $f(k)n^\alpha$, for some function f and constant α.*

Because the clique problem is complete for the complexity class $W[1]$, a polynomial time learning algorithm as hypothesized in the theorem for any nonconstant computable function $d(n)$ would imply fixed-parameter tractability of

all the problems in $W[1]$ [8,15]. However, we show that restricting the circuit to be transitively reduced (Theorem 2), or more generally, of bounded shortcut width (Theorem 3), avoids the necessity of a depth bound at all.[2]

3.2 Distinguishing Paths

This section develops some properties of distinguishing paths, making no assumptions about shortcut width. Let C^* be a circuit with n wires, an alphabet Σ of cardinality s, and fan-in bounded by a constant k. An arbitrary gate function for such a circuit can be represented by a **gate table** with s^k entries, giving the value of the gate function for each possible k-tuple of input symbols.

Experiment e **distinguishes** σ from τ for w if e sets w to $*$ and $C^*(e|_{w=\sigma}) \neq C^*(e|_{w=\tau})$. If such an experiment exists, the values σ and τ are **distinguishable** for wire w; otherwise, σ and τ are **indistinguishable** for w.

A **test path** π for a wire w in C^* consists of a directed path of wires from w to the output wire, together with an assignment giving fixed values from Σ to some set S of other wires; S must be disjoint from the set of wires in the path, and each element of S must be an input to some wire beyond w along the path. The wires in S are the **side wires** of the test path π. The **length** of a test path is the number of edges in its directed path. There is just one test path of length 0, consisting of the output wire and no side wires.

We may associate with a test path π the partial experiment p_π that assigns $*$ to each wire on the path, and the specified value from Σ to each wire in S. An experiment e **agrees with** a test path π if e extends the partial experiment p_π, that is, p_π is a subfunction of e. We also define the experiment e_π that extends p_π by setting all the other wires to $*$.

If π is a test path and V is a set of wires disjoint from the side wires of π, then V is **functionally determining** for π if for any experiment e agreeing with π and leaving the wires in V free, for any experiment e' obtained from e by setting the wires in V to fixed values, the value of $C^*(e')$ depends only on the values assigned to the wires in V. That is, the values on the wires in V determine the output of the circuit, given the assignments specified by p_π. A test path π for w is **isolating** if $\{w\}$ is functionally determining for π.

Lemma 1. *If π is an isolating test path for w then the set V of inputs of w is functionally determining for π.*

We define a **distinguishing path** for wire w and values $\sigma, \tau \in \Sigma$ to be an isolating test path π for w such that e_π distinguishes between σ and τ for w. The significance of distinguishing paths is indicated by the following lemma, which is analogous to Lemma 10 of [5].

Lemma 2. *Suppose σ and τ are distinguishable for wire w. Then for any minimal experiment e distinguishing σ from τ for w, there is a distinguishing path π for wire w and values σ and τ such that the free wires of e are exactly the wires of the directed path of π, and e agrees with π.*

[2] The target circuit C constructed in the reduction is of shortcut width $k-1$.

Conversely, a shortest distinguishing path yields a minimal distinguishing experiment, as follows. This does not hold for circuits of general topology without the restriction to a shortest path.

Lemma 3. *Let π be a shortest distinguishing path for wire w and values σ and τ. Then the experiment e obtained from p_π by setting every unspecified wire to an arbitrary fixed value is a minimal experiment distinguishing σ from τ for w.*

3.3 The Distinguishing Paths Algorithm

In this section we develop the Distinguishing Paths Algorithm.

Theorem 2. *The Distinguishing Paths Algorithm learns any transitively reduced circuit with n wires, alphabet size s, and fan-in bound k, with $O(n^{2k+1}s^{2k+2})$ value injection queries and time polynomial in the number of queries.*

Lemma 4. *If C^* is a transitively reduced circuit and π is a test path for w in C^*, then none of the inputs of w is a side wire of π.*

The Distinguishing Paths Algorithm builds a directed graph G whose vertices are the wires of C^*, in which an edge (v, w) represents the discovery that v is an input of w in C^*. The algorithm also keeps for each wire w a **distinguishing table** T_w with $\binom{s}{2}$ entries, one for each unordered pair of values from Σ. The entry for (σ, τ) in T_w is 1 or 0 according to whether or not a distinguishing path has been found to distinguish values σ and τ on wire w. Stored together with each 1 entry is a corresponding distinguishing path and a bit marking whether the entry is processed or unprocessed.

At each step, for each distinguishing table T_w that has unprocessed 1 entries, we try to extend the known distinguishing paths to find new edges to add to G and new 1 entries and corresponding distinguishing paths for the distinguishing tables of inputs of w. Once every 1 entry in every distinguishing table has been marked processed, the construction of distinguishing tables terminates. Then a circuit C is constructed with graph G by computing gate tables for the wires; the algorithm outputs C and halts.

To extend a distinguishing path for a wire w, it is necessary to find an input wire of w. Given a distinguishing path π for wire w, an input v of w is **relevant** with respect to π if there are two experiments e_1 and e_2 that agree with π, that set the inputs of w to fixed values, that differ only by assigning different values to v, and are such that $C^*(e_1) \neq C^*(e_2)$. Let $V(\pi)$ denote the set of all inputs v of w that are relevant with respect to π. It is only relevant inputs of w that need be found, as shown by the following.

Lemma 5. *Let π be a distinguishing path for w. Then $V(\pi)$ is functionally determining for π.*

Given a distinguishing path π for wire w, we define its corresponding **input experiments** E_π to be the set of all experiments e that agree with π and set up to $2k$ additional wires to fixed values and set the rest of the wires free. Note

that each of these experiments fix at most $2k$ more values than are already fixed in the distinguishing path. Consider all pairs (V, Y) of disjoint sets of wires not set by p_π such that $|V| \leq k$ and $|Y| \leq k$; for every possible way of setting $V \cup Y$ to fixed values, there is a corresponding experiment in E_π.

Find-Inputs. We now describe a procedure, Find-Inputs, that uses the experiments in E_π to find all the wires in $V(\pi)$. Define a set V of at most k wires not set by p_π to be **determining** if for every disjoint set Y of at most k wires not set by p_π and for every assignment of values from Σ to the wires in $V \cup Y$, the value of C^* on the corresponding experiment from E_π is determined by the values assigned to wires in V, independent of the values assigned to wires in Y. Find-Inputs finds all determining sets V and outputs their intersection.

Lemma 6. *Given a distinguishing path π for w and its corresponding input experiments E_π, the procedure Find-Inputs returns $V(\pi)$.*

Find-Paths. We now describe a procedure, Find-Paths, that takes the set $V(\pi)$ of all inputs of w relevant with respect to π, and searches, for each triple consisting of $v \in V(\pi)$ and $\sigma, \tau \in \Sigma$, for two experiments e_1 and e_2 in E_π that fix all the wires of $V(\pi) - \{v\}$ in the same way, but set v to σ and τ, respectively, and are such that $C^*(e_1) \neq C^*(e_2)$. On finding such a triple, the distinguishing path π for w can be extended to a distinguishing path π' for v by adding v to the start of the path, and making all the wires in $V(\pi) - \{v\}$ new side wires, with values fixed as in e_1. If this gives a new 1 for entry (σ, τ) in the distinguishing paths table T_v, then we change the entry, add the corresponding distinguishing path for v to the table, and mark it unprocessed. We have to verify the following.

Lemma 7. *Suppose π' is a path produced by Find-Paths for wire v and values σ and τ. Then π' is a distinguishing path for wire v and values σ, τ.*

The Distinguishing Paths Algorithm initializes the simple directed graph G to have the set of wires of C^* as its vertex set, with no edges. It initializes T_w to all 0's, for every non-output wire w. Every entry in T_{w_n} is initialized to 1, with a corresponding distinguishing path of length 0 with no side wires, and marked as unprocessed. The Distinguishing Paths Algorithm is summarized in Algorithm 1; the procedure Construct-Circuit is described below.

We now show that when processing of the tables terminates, the tables T_w are correct and complete. We first consider the correctness of the 1 entries.

Lemma 8. *After the initialization, and after each new* 1 *entry is placed in a distinguishing table, every* 1 *entry in a distinguishing table T_w for (σ, τ) has a corresponding distinguishing path π for wire w and values σ and τ.*

A distinguishing table T_w is **complete** if for every pair of values $\sigma, \tau \in \Sigma$ such that σ and τ are distinguishable for w, T_w has a 1 entry for (σ, τ).

Lemma 9. *When the Distinguishing Paths Algorithm terminates, T_w is complete for every wire w in C^*.*

Algorithm 1. Distinguishing Paths Algorithm

```
Initialize G to have the wires as vertices and no edges.
Initialize T_{w_n} to all 1's, marked unprocessed.
Initialize T_w to all 0's for all non-output wires w.
while there is an unprocessed 1 entry (σ, τ) in some T_w do
  Let π be the corresponding distinguishing path.
  Perform all input experiments E_π.
  Use Find-Inputs to determine the set V(π).
  Add any new edges (v, w) for v ∈ V(π) to G.
  Use Find-Paths to find extensions of π for elements of V(π).
  for each extension π' that gives a new 1 entry in some T_v do
    Put the new 1 entry in T_v with distinguishing path π'.
    Mark this new 1 entry as unprocessed.
  Mark the 1 entry for (σ, τ) in T_w as processed.
Use Construct-Circuit with G and the tables T_w to construct a circuit C.
Output C and halt.
```

Construct-Circuit. Now we show how to construct a circuit C behaviorally equivalent to C^* given the graph G and the final distinguishing tables. G is the graph of C, determining the input relation between wires. Note that G is a subgraph of the graph of C^*, because edges are added only when relevant inputs are found.

Gate tables for wires in C will keep different combinations of input values and their corresponding output. Since some distinguishing tables for wires may have 0 entries, we will record values in gate tables up to equivalence, where σ and τ are in the same equivalence class for w if they are indistinguishable for w. We process one wire at a time, in arbitrary order. We first record, for one representative σ of each equivalence class of values for w, the outputs $C^*(e_\pi | w = \sigma)$ for all the distinguishing paths π in T_w. Given a setting of the inputs to w (in C), we can tell which equivalence class of values of w it should map to as follows. For each distinguishing path π in T_w, we record the output of C^* for the experiment equal to e_π with the inputs of w set to the given fixed values and $w = *$. The value of σ with recorded outputs that match these outputs for all π is written in w's gate table as the output for this setting of the inputs. Repeating this for every setting of w's inputs completes w's gate table, and we continue to the next gate.

Lemma 10. *Given the graph G and distinguishing tables as constructed in the Distinguishing Paths Algorithm, the procedure Construct-Circuit constructs a circuit C behaviorally equivalent to C^*.*

We analyze the total number of value injection queries used by the Distinguishing Paths Algorithm; the running time is polynomial in the number of queries. To construct the distinguishing tables, each 1 entry in a distinguishing table is processed once. The total number of possible 1 entries in all the tables is bounded by ns^2. The processing for each 1 entry is to take the corresponding distinguishing path π and construct the set E_π of input experiments, each of which consists of choosing up to $2k$ wires and setting them to arbitrary values from Σ, for a

total of $O(n^{2k}s^{2k})$ queries to construct E_π. Thus, a total of $O(n^{2k+1}s^{2k+2})$ value injection queries are used to construct the distinguishing tables.

To build the gate tables, for each of n wires, we try at most s^2 distinguishing path experiments for at most s values of the wire, which takes at most s^3 queries. We then run the same experiments for each possible setting of the inputs to the wire, which takes at most $s^k s^2$ experiments. Thus Construct-Circuit requires a total of $O(n(s^3 + s^{k+2}))$ experiments, which are already among the ones made in constructing the distinguishing tables. Note that every experiment fixes at most $O(kd)$ wires, where d is the depth of C^*. This concludes the proof of Theorem 2.

3.4 The Shortcuts Algorithm

In this section we sketch the Shortcuts Algorithm, which generalizes the Distinguishing Paths Algorithm to circuits with bounded shortcut width.

Theorem 3. *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 a number of value injection queries bounded by $(ns)^{O(k+b)}$ and time polynomial in the number of queries.*

When C^* is not transitively reduced, there may be edges of its graph that are important to the behavior of the circuit, but are not completely determined by the behavior of the circuit. For example, three circuits given in [5] are behaviorally equivalent, but have different topologies; a behaviorally correct circuit cannot be constructed with just the edges that are common to the three circuit graphs. Thus, the Shortcuts Algorithm focuses on finding a **sufficient** set of experiments for C^*, and uses Circuit Builder [5] to build the output circuit C.

On the positive side, we can learn quite a bit about the topology of a circuit C^* from its behavior. An edge (v, w) of the graph of C^* is **discoverable** if it is the initial edge on some minimal distinguishing experiment e for v and some values σ_1 and σ_2. This is a behaviorally determined property; all circuits behaviorally equivalent to C^* must contain all the discoverable edges of C^*.

We generalize the definition of a distinguishing path to a **distinguishing path with shortcuts**, which has an additional set of **cut wires** K, which is disjoint from the path wires and the side wires, and is such that every wire in K is an input to some wire beyond w on the path (where w is the initial wire.) Moreover, $\{w\} \cup K$ is functionally determining for the path.

Like the Distinguishing Paths Algorithm, the Shortcuts Algorithm maintains a directed graph G containing known edges (v, w) of the graph of C^*, and a set of distinguishing tables T_w indexed by triples (B, a_1, a_2), where B is a set of at most b wires not containing w, and a_1 and a_2 are assignments of fixed values to the wires in $\{w\} \cup B$. If there is an entry for (B, a_1, a_2) in T_w, it is a distinguishing path with shortcuts π such that $K \subseteq B$ and $K \cap S = \emptyset$ and it distinguishes a_1 from a_2. Each entry is marked as processed or unprocessed.

The algorithm processes an entry by using the distinguishing path π for (w, B) to find new edges (v, w) in G, and to find new or updated entries in the tables

T_v such that (v, w) is in G. An entry is updated if a new distinguishing path for the entry is shorter than the current one, which it then replaces. When an entry is created or updated, it is marked as unprocessed. All entries in T_w are also marked as unprocessed when a new edge (v, w) is added to G.

We show that when processing of the tables T_w is complete, G contains every discoverable edge of C^* and for every wire w and the shortcut wires $B(w)$ of w in C^*, if the assignments a_1 and a_2 are distinguishable for $(w, B(w))$, then there is a correct entry for $(B(w), a_1, a_2)$ in T_w. The final tables T_w are used to create experiments for Circuit Builder. To guarantee a sufficient set of experiments, this procedure is iterated for every restriction of C^* obtained by selecting at most k possible input wires and assigning arbitrary fixed values to them.

4 Learning Analog Circuits Via Discretization

We show how to construct a discrete approximation of an analog circuit, assuming its gate functions satisfy a Lipschitz condition with constant L, and apply the large-alphabet learning algorithm of Theorem 3.

4.1 A Lipschitz Condition

An analog function g of arity k satisfies a Lipschitz condition with constant L if for all $x_1, \ldots, x_k$ and $x'_1, \ldots, x'_k$ from $[0, 1]$ we have

$$|g(x_1, \ldots, x_k) - g(x'_1, \ldots, x'_k)| \leq L \max_i |x_i - x'_i|.$$

Let m be a positive integer. We define a discretization function D_m from $[0, 1]$ to the m points $\{1/2m, 3/2m, \ldots, (2m-1)/2m\}$ by mapping x to the closest point in this set (choosing the smaller point if x is equidistant from two of them.) Then $|x - D_m(x)| \leq 1/2m$ for all $x \in [0, 1]$. We extend D_m to discretize analog experiments e by defining $D_m(*) = *$ and applying it componentwise to e.

Lemma 11. *If g is an analog function of arity k, satisfying a Lipschitz condition with constant L and m is a positive integer, then for all $x_1, \ldots, x_k$ in $[0, 1]$, $|g(x_1, \ldots, x_k) - g(D_m(x_1), \ldots, D_m(x_k))| \leq L/2m$.*

4.2 Discretizing Analog Circuits

We describe a discretization of an analog gate function in which the inputs and the output may be discretized differently. Let g be an analog function of arity k and r, s be positive integers. The (r, s)**-discretization** of g is g', defined by

$$g'(x_1, \ldots, x_k) = D_r(g(D_s(x_1), \ldots, D_s(x_k))).$$

Let C be an analog circuit of depth d_{max} and let L and N be positive integers. Define $m_d = N(3L)^d$ for all nonnegative integers d. We construct a particular

discretization C' of C by replacing each gate function g_i by its (m_d, m_{d+1})-discretization, where d is the depth of wire w_i. We also replace the value set $\Sigma = [0,1]$ by the value set Σ' equal to the union of the ranges of D_{m_d} for $0 \leq d \leq d_{max}$. Note that the wires and tuples of inputs remain unchanged. The resulting discrete circuit C' is termed the (L,N)**-discretization** of C.

Lemma 12. *Let L and N be positive integers. Let C be an analog circuit of depth d_{max} whose gate functions all satisfy a Lipschitz condition with constant L. Let C' denote the (L,N)-discretization of C and let $M = N(3L)^{d_{max}}$. Then for any experiment e for C, $|C(e) - C'(D_M(e))| \leq 1/N$.*

This lemma shows that if every gate of C satisfies a Lipschitz condition with constant L, we can approximate C's behavior to within ϵ using a discretization with $O((3L)^d/\epsilon)$ points, where d is the depth of C. For $d = O(\log n)$, this bound is polynomial in n and $1/\epsilon$.

Theorem 4. *There is a polynomial time algorithm that approximately learns any analog circuit of n wires, depth $O(\log n)$, constant fan-in, gate functions satisfying a Lipschitz condition, and shortcut width bounded by a constant.*

5 Learning with Experiments and Counterexamples

In this section, we consider the problem of learning circuits using both value injection queries and counterexamples. In a **counterexample query**, the algorithm proposes a hypothesis C and receives as answer either the fact that C exactly equivalent to the target circuit C^*, or a **counterexample**, that is, an experiment e such that $C(e) \neq C^*(e)$. In [5], polynomial-time algorithms are given that use value injection queries and counterexample queries to learn (1) acyclic circuits of arbitrary depth with arbitrary gates of constant fan-in, and (2) acyclic circuits of arbitrary depth with NOT gates and AND, OR, NAND, and NOR gates of arbitrary fan-in.

The algorithm that we now develop generalizes both previous algorithms by permitting any class of gates that is polynomial time learnable with counterexamples. It also guarantees that the depth of the output circuit is no greater than the depth of the target circuit and the number of additional wires fixed in value injection queries is bounded by $O(kd)$, where k is a bound on the fan-in and d is a bound on the depth of the target circuit.

5.1 The Learning Algorithm

The algorithm proceeds in a cycle of proposing a hypothesis, getting a counterexample, processing the counterexample, and then proposing a new hypothesis. Whenever we receive a counterexample e, we process the counterexample so that we can "blame" at least one gate in C; we find a witness experiment e^* eliminating a gate g in C. In effect, we reduce the problem of learning a circuit to the problem of learning individual gates with counterexamples.

An experiment e^* is a **witness experiment** eliminating g, if and only if e^* fixes all inputs of g but sets g free and $C(e^*|_{w=g(e^*)}) \neq C(e^*)$. It is important that we require e^* fix all inputs of g, because then we know it is g and not its ancestors computing wrong values. The main operation of the procedure that processes counterexamples is to fix wires.

Given a counterexample e, let procedure **minimize** fix wires in e while preserving the property that $C(e) \neq C'(e)$ until it cannot fix any more. Therefore, $e^* = minimize(e)$ is a minimal counterexample for C' under the partial order $\preceq$ defined in Sect. 2.2. The following lemma is a consequence of Lemma 10 in [5].

Lemma 13. *If e^* is a minimal counterexample for C', there exists a gate g in C' such that e^* is a witness experiment for g.*

Now we run a separate counterexample learning algorithm for each individual wire. Whenever C' receives a counterexample, at least one of the learning algorithms will receive one. However, if we run all the learning algorithms simultaneously and let each learning algorithm propose a gate function, the hypothesis circuit may not be acyclic. Instead we will use Algorithm 2 to coordinate them, which can be viewed as a generalization of the circuit building algorithm for learning AND/OR circuits in [5]. Conflicts are defined below.

Algorithm 2. Learning with experiments and counterexamples

```
Run an individual learning algorithm for each wire w. Each learning algorithm takes
as candidate inputs only wires that have fewer conflicts.
Let C be the hypothesis circuit.
while there is a counterexample for C do
    Process the counterexample to obtain a counterexample for a wire w.
    Run the learning algorithm for w with the new counterexample.
    if there is a conflict for w then
        Restart the learning algorithms for w and all wires whose candidate inputs have
        changed.
```

The algorithm builds an acyclic circuit C because each wire has as inputs only wires that have fewer conflicts. At the start, each individual learning algorithm runs with an empty candidate input set since there is yet no conflict. Thus, each of them tries to learn each gate as a constant gate, and some of them will not succeed. A **conflict** for w happens when there is no hypothesis in the hypothesis space that is consistent with the set of counterexamples received by w. For constant gates, there is a conflict when we receive a counterexample for each of the $|\Sigma|$ possible constant functions. We note that there will be no conflict for a wire w if the set of candidate inputs contains the set of true inputs of w in the target circuit C^*, because then the hypothesis space contains the true gate.

Whenever a conflict occurs for a wire, it has a chance of having more wires as candidate inputs. Therefore, our learning algorithm can be seen as repeatedly rebuilding a partial order over wires based on their numbers of conflicts. Another natural partial order on wires is given by the **level** of a wire, defined as the length

of a longest directed path in C^* from a constant gate to the wire. The following lemma shows an interesting connection between levels and numbers of conflicts.

Lemma 14. *The number of conflicts each wire receives is bounded by its level.*

Corollary 1. *The depth of C is at most the depth of C^*.*

In fact, the depth of C is bounded by the minimum depth of any circuit behaviorally equivalent to C^*.

Theorem 5. *Circuits whose gates are polynomial time learnable with counterexamples are learnable in polynomial time with experiments and counterexamples.*

Proof. By the learnability assumption of each gate, Algorithm 2 will receive only polynomially many counterexamples between two conflicts, because the candidate inputs for every wire are unchanged. (A conflict can be detected when the number of counterexamples exceeds the polynomial bound.) Lemma 14 bounds the number of conflicts for each wire by its level, which then bounds the total number of counterexamples of Algorithm 2 by a polynomial. It is clear that we use $O(n)$ experiments to process each counterexample. Thus, the total number of experiments is bounded by a polynomial as well.

5.2 A New Diagnosis Algorithm

A shortcoming of **minimize** is that it fixes many wires, which may be undesirable in the context of gene expression experiments. In this section, we propose a new diagnosis algorithm to find a witness experiment e^* for some gate g in C. If the hypothesis circuit C has depth d and fan-in bound k, the new algorithm fixes only $O(dk)$ more gates than the number fixed in the original counterexample.

Given a counterexample e, we first gather a list of potentially wrong wires. Let $w_C(e)$ be the value of wire w in C under experiment e. We can compute $w_C(e)$ given e because we know C. The **potentially wrong** wires are those w's such that $C^*(e|_{w=w_C(e)}) \neq C^*(e)$. It is not hard to see that a potentially wrong wire must be a free wire in e. We can gather all **potentially wrong** wires by conducting n experiments, each fixing one more wire than e does.

Now, pick an arbitrary potentially wrong wire w and let g be its gate function in C. If g's inputs are fixed in e, then e is a witness experiment for g, and we are done. Otherwise, fix all g's free input wires to their values in C, and let e' be the resulting experiment. There are two cases: either g is wrong or one of g's inputs computes a wrong value.

1. If $C^*(e'|_{w=w_C(e)}) \neq C^*(e')$, then e' is a witness experiment for g.
2. Otherwise, we have $C^*(e'|_{w=w_C(e)}) = C^*(e')$. Because $C^*(e|_{w=w_C(e)}) \neq C^*(e)$, we have either $C^*(e') \neq C^*(e)$ or $C^*(e'|_{w=w_C(e)}) \neq C^*(e|_{w=w_C(e)})$. Note that the only difference between e and e' is that e' fixes free inputs of g to their values in C. So either e or $e|_{w=w_C(e)}$ is an experiment in which fixing all g's free inputs gives us a change in the circuit outputs. We then

start from whichever experiment gives us such a change and fix free inputs of g in C one after another, until the circuit output changes. We will find an experiment e'', for which one of g's inputs is potentially wrong. We then restart the process with e'' and this input of g.

At each iteration, we go to a deeper gate in C. The process will stop within d iterations. If C has fan-in at most k, the whole process will fix at most $d(k-1)+1$ more gates than were fixed in the original experiment e.

References

1. Aho, A.V., Garey, M.R., Ullman, J.D.: The transitive reduction of a directed graph. SIAM J. Comput. 1, 131–137 (1972)
2. Akutsu, T., Kuhara, S., Maruyama, O., Miyano, S.: Identification of gene regulatory networks by strategic gene disruptions and gene overexpressions. In: SODA '98: Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 695–702, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics (1998)
3. Angluin, D., Frazier, M., Pitt, L.: Learning conjunctions of Horn clauses. Machine Learning 9, 147–164 (1992)
4. Angluin, D., Hellerstein, L., Karpinski, M.: Learning read-once formulas with queries. J. ACM 40, 185–210 (1993)
5. Angluin, D., Aspnes, J., Chen, J., Wu, Y.: Learning a circuit by injecting values. In: Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing, pp. 584–593. ACM Press, New York, USA (2006)
6. Angluin, D., Kharitonov, M.: When won't membership queries help? J. Comput. Syst. Sci. 50(2), 336–355 (1995)
7. Bshouty, N.H.: Exact learning boolean functions via the monotone theory. Inf. Comput. 123(1), 146–153 (1995)
8. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
9. Ideker, T., Thorsson, V., Karp, R.: Discovery of regulatory interactions through perturbation: Inference and experimental design. In: Pacific Symposium on Biocomputing 5, 302–313 (2000)
10. Jackson, J.C.: An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. J. Comput. Syst. Sci. 55(3), 414–440 (1997)
11. Jackson, J.C., Klivans, A.R., Servedio, R.A.: Learnability beyond AC0. In: STOC '02: Proceedings of the thirty-fourth annual ACM symposium on Theory of computing, pp. 776–784. ACM Press, New York, USA (2002)
12. Kearns, M., Valiant, L.: Cryptographic limitations on learning boolean formulae and finite automata. J. ACM 41(1), 67–95 (1994)
13. Kharitonov, M.: Cryptographic hardness of distribution-specific learning. In: STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, pp. 372–381. ACM Press, New York, USA (1993)
14. Linial, N., Mansour, Y., Nisan, N.: Constant depth circuits, Fourier transform, and learnability. Journal of the ACM 40(3), 607–620 (1993)
15. Niedermeier, R. (ed.): Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)