Learning Large-Alphabet and Analog Circuits with Value Injection Queries

Dana Angluin[1] James Aspnes[1,*] Jiang Chen[2,†]
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 [7] to handle general classes of gate functions that are polynomial time learnable from counterexamples.

*Supported in part by NSF grant CNS-0435201.

†Supported in part by a research contract from Consolidated Edison.

‡Supported in part by a Yahoo! Research Kern family Scholarship.

§This material is based upon work supported under a National Science Foundation Graduate Research Fellowship.

1 Introduction

We consider learning large-alphabet and analog acyclic circuits in the value injection model introduced in [7]. 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 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 [11] 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 [15] 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 [8], even with respect to the uniform distribution [16]. Furthermore, positive learnability results exist only for fairly limited classes, including propositional Horn formulas [4], general read once Boolean formulas [5], and decision trees [9], and those for specific distributions, including **AC0** circuits [17], DNF formulas [12], and **AC0** circuits with a limited number of majority gates [13].[1]

Thus, Angluin et al. [7] 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

[1] Algorithms in both [17] and [13] for learning **AC0** circuits and their variants run in quasi-polynomial time.

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 [10, 20].

In this paper we show positive learnability results for bounded fan-in, large alphabet, arbitrary depth circuits given some restrictions on the topology of the target circuit. Specifically, we show that transitively reduced circuits and circuits with bounded shortcut width (as defined in section 2) are exactly learnable in polynomial time, and we present evidence that shortcut width is the correct parameter to look at for large alphabet circuits. We also show that analog circuits of bounded fan-in, logarithmic depth, and small shortcut width that satisfy a Lipschitz condition are approximately learnable in polynomial time. Finally, we extend the results of [7] when behavioral equivalence queries are also available, for both binary and large-alphabet circuits.

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.

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. Note that in a transitively reduced circuit, for every output-connected wire w_i, no ancestor of w_i is an input of any descendant of w_i, otherwise there would be a shortcut edge in the graph of the circuit.

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, or vertices, 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, for any experiment 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 6.1, 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 5 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 [7] 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 [7] 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.

Combining these results with the discretization from Section 5, we obtain an algorithm using value-injection queries that learns, up to ϵ-equivalence, analog circuits satisfying a Lipschitz condition with constant bound, depth bounded by $O(\log n)$, having constant fan-in and constant shortcut width in time polynomial in n and $1/\epsilon$.

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 [10, 20]), 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.*

Proof. (Note that the function f need not be a polynomial.) On input (G, k), where G has n vertices, we construct the circuit C as described above, which has alphabet size $s' = \binom{n}{2}$, depth $d' = \binom{k}{2}$ and number of wires $n' = d' + k + 1$. We then evaluate $d(1), d(2), \ldots$ to find the least N such that $d(N) \geq n'$. Such an N may be found because $d(n)$ is a nonconstant computable function; the value of N depends only on k. We run the learning algorithm on the circuit C padded with inessential wires to have N wires, using C to answer the value injection queries. By hypothesis, because $d' \leq d(N)$, the learning algorithm runs in time polynomial in N and s'. Its queries enable us to answer correctly whether G has a clique of size k. The total running time is bounded by $f(k)n^\alpha$ for some function f and some 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 non-constant computable function $d(n)$ would imply fixed-parameter tractability of all the problems in $W[1]$ [10, 20]. However, we show that restricting the circuit to be transitively reduced (Theorem 5), or more generally, of bounded shortcut width (Theorem 13), avoids the necessity of a depth bound at all.[2]

Remark. A natural question is whether a pattern graph less dense than a clique might avoid squaring the parameter k in the reduction. In fact, there is a polynomial-time algorithm to test whether a graph contains a path of length $O(\log n)$ [3]. A reduction similar to the one above can be used to test for the presence of an arbitrary graph H on k vertices $\{1, \ldots, k\}$ as an induced subgraph in G. The gate with inputs x_ℓ and x_m tests for an edge in G (if (ℓ, m) is an edge of H) or tests whether the vertices are distinct and not an edge of G (if (ℓ, m) is not an edge of H.) Note that regardless of the number of edges in H, the all-pairs structure is necessary to verify that the distinctness of the vertices assigned to $x_1, \ldots, x_k$.

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

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

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. The following property is then clear.

Lemma 2. *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 [7].

Lemma 3. *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.*

Proof. We prove the result by induction on the depth of the wire w; it clearly holds when w is the output wire. Suppose the result holds for all wires at depth at most d in C^*, and assume that w is a wire at depth $d+1$ and that e is any minimal experiment that distinguishes σ from τ for w. Every free wire in e must be reachable from w; using the acyclicity of C^*, let w' be a free wire in e whose only free input is w. Let $\sigma' = w'(e|_{w=\sigma})$ and $\tau' = w'(e|_{w=\tau})$. Because e is minimal, we must have $\sigma' \neq \tau'$.

Moreover, the minimality of e also implies that

$$C^*(e|_{w=\sigma,w'=\sigma'}) = C^*(e|_{w=\tau,w'=\sigma'})$$

and

$$C^*(e|_{w=\sigma,w'=\tau'}) = C^*(e|_{w=\tau,w'=\tau'}),$$

so we must have

$$C^*(e|_{w=\sigma,w'=\sigma'}) \neq C^*(e|_{w=\sigma,w'=\tau'}),$$

which means that the experiment $e' = e|_{w=\sigma}$ distinguishes σ' from τ' for w'. The experiment e' is also a minimal experiment distinguishing σ' from τ' for w'; otherwise, e would not be minimal. The depth of w' is at most d, so by induction, 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 π', and e' agrees with π'.

We may extend π' to π as follows. Add w to the start of the directed path in π'. The side wires of π are the side wires of π' with their settings in π', together with any inputs of w' (other than w) that are not already side wires of π', set as in e. The result is clearly an isolating test path for w that distinguishes σ from τ. Also the wires in the directed path of π are precisely the free wires of e, and e agrees with π, which completes the induction. □

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 4. *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.*

Proof. Because π is a distinguishing path, w is functionally determining for π, so e distinguishes σ from τ for w. If e is not minimal, then there is some minimal $e' \preceq e$ such that e' distinguishes σ and τ for w. By Lemma 3, there is a distinguishing path for w and values σ and τ whose path wires are the free wires of e'. This contradicts the assumption that π as a shortest path distinguishing σ from τ for w.

□

3.3 The Distinguishing Paths Algorithm

In this section we develop the Distinguishing Paths Algorithm.

Theorem 5. *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 6. *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.*

Proof. Every side wire u of π is an input to some wire beyond w in the directed path of wires, that is, to some descendant of w. If u were an input to w, then u would be an ancestor of w and an input to a descendant of w, contradicting the assumption that C^* is transitively reduced. □

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

Proof. Suppose $V(\pi)$ is not functionally determining for π. Then there are two experiments e_1 and e_2 that agree with π and assign $*$ to all the wires in $V(\pi)$, and an assignment a of fixed values to the wires in $V(\pi)$ such that the two experiments e_1' and e_2' obtained from e_1 and e_2 by fixing all the wires in $V(\pi)$ as in a have the property that $C^*(e_1') \neq C^*(e_2')$.

Because π is a distinguishing path for w, the set V of all inputs of w is functionally determining for π. Thus, e_1' and e_2' must induce different values

for at least one input of w (that cannot be in $V(\pi)$.) Let e_1'' be e_1' with all of the inputs of w fixed to their induced values in e_1', and similarly for e_2'' with respect to e_2'. Now $C^*(e_1'') = C^*(e_1') \neq C^*(e_2') = C^*(e_2'')$, and both e_1'' and e_2'' fix all the inputs of w. By changing the differing fixed values of the inputs of w one by one from their setting in e_1'' to their setting in e_2'', we can find a single input wire u of w such that changing just its value changes the output of the circuit. The resulting two experiments witness that u is an input of w relevant with respect to π, which contradicts the fact that u is not in $V(\pi)$. □

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

Proof. First, there is at least one set in the intersection, because if V_w is the set of all inputs to w in C^*, then by Lemma 6 and the acyclicity of C^*, no wires in V_w are set in p_π. By Lemma 2, V_w is functionally determining for π and therefore determining, and, by the bound on fan-in, $|V_w| \leq k$, so V_w will be one such set V. Let V^* denote the intersection of all determining sets V.

Clearly, every wire in V^* is an input of w, because $V^* \subseteq V_w$. To see that each $v \in V^*$ is relevant with respect to π, consider the set $V' = V_w - \{v\}$ of inputs of w other than v. This set must not appear in V^*(because $v \in V^*$), so it must be that for some pair (V', Y) there are two experiments e_1 and

e_2 in E_π that give the same fixed assignments to V' and different fixed assignments to Y, and are such that $C^*(e_1) \neq C^*(e_2)$. Then $v(e_1) \neq v(e_2)$, because $V' \cup \{v\}$ is functionally determining for π. Thus, if we take e'_1 to be e_1 with v fixed to $v(e_1)$ and e'_2 to be e_1 with v fixed to $v(e_2)$, we have two experiments that witness that v is relevant with respect to π. Thus $V^* \subseteq V(\pi)$.

Conversely, suppose $v \in V(\pi)$ and that V^* does not include v. Then there is some set V in the intersection that excludes v. Also, there are two experiments e_1 and e_2 that agree with π, set the inputs of w to fixed values and differ only on v, such that $C^*(e_1) \neq C^*(e_2)$. Let Y consist of all the inputs of w that are not in V; clearly $v \in Y$, none of the elements of Y are set in p_π and $|Y| \leq k$. There is an experiment $e'_1 \in E_\pi$ for the pair (V, Y) that sets the inputs of w as in e_1 and the other wires of V arbitrarily, and another experiment $e'_2 \in E_\pi$ for the pair (V, Y) that agrees with e_1 except in setting v to its value in e_2. These two experiments set the inputs of w as in e_1 and e_2 respectively, and the inputs of w are functionally determining for π, so we have $C^*(e'_1) = C^*(e_1) \neq C^*(e_2) = C^*(e'_2)$. This is a contradiction: V would not have been included in the intersection. Thus $V(\pi) \subseteq V^*$, concluding the proof. □

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 9. *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.*

Proof. Because v is an input to w in C^*, prefixing v to the path from π is a path of wires from v to the output wire in C^*. Because v is an input of w, by Lemma 6, v is not among the side wires S for π. The new side wires are those in $V(\pi) - \{v\}$, and because they are inputs of w, by Lemma 6 they are not already on the path for π nor in the set S. Thus, π' is a test path. The

new side wires are fixed to values with the property that changing v between σ and τ produces a difference at the output of C^*. Because by Lemma 7, $V(\pi)$ is functionally determining for π, the test path π' is isolating for v. Thus π' is a distinguishing path for wire v and values σ and 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.

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.
```

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 10. *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.*

Proof. This condition clearly holds after the initialization, because the distinguishing path consisting of just the output wire and no side wires correctly distinguishes every distinct pair of values from Σ. Then, by induction on

the number of new 1 entries in distinguishing path tables, when an existing 1 entry in T_w gives rise to a new one in T_v, then the path π from T_w is a correct distinguishing path for w. Thus, by Lemma 8, the Find-Inputs procedure correctly finds the set $V(\pi)$ of inputs of w relevant with respect to π, and by Lemma 9, the Find-Paths procedure correctly finds extensions of π to distinguishing paths π' for elements of $V(\pi)$. Thus, any new 1 entry in a table T_v will have a correct corresponding distinguishing path. □

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

Proof. Assume to the contrary and look at a wire w at the smallest possible depth such that T_w is incomplete; assume it lacks a 1 entry for the pair (σ, τ), which are distinguishable for w. Note that w cannot be the output wire. Because the depth of w is at least one more than the depth of any descendant of w, all wires on all directed paths from w to the root have complete distinguishing tables. By Lemma 10, all the entries in all distinguishing tables are also correct.

Because σ and τ are distinguishable for w, by Lemma 3 there exists a distinguishing path π for wire w and values σ and τ. On this distinguishing path, w is followed by some wire x. The wires along π starting with x and omitting any side wires that are inputs of x is a distinguishing path for wire x and values σ' and τ', where σ' is the value that x takes when $w = \sigma$ and τ' is the value that x takes when $w = \tau$ in any experiment agreeing with π.

Because x is a descendant of w, its distinguishing table T_x is complete and correct. Thus, there exists in T_x a 1 entry for (σ', τ') and a corresponding distinguishing path π_x. This 1 entry must be processed before the Distinguishing Paths Algorithm terminates. When it is processed, two of the input experiments for π_x will set the inputs of x in agreement with π, and set w to σ and τ respectively. Thus, w will be discovered to be a relevant input of x with respect to π, and a distinguishing experiment for wire w and values σ and τ will be found, contradicting the assumption that T_w never gets a 1 entry for (σ, τ). Thus, no such wire w can exist and all the distinguishing tables are complete. □

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 $w = *$. For this setting of the inputs, we set the output in w's gate table to be the value of σ with recorded outputs matching these outputs for all π. Repeating this for every setting of w's inputs completes w's gate table, and we continue to the next gate.

Lemma 12. *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^*.*

Proof. Assume to the contrary that C is not behaviorally equivalent to C^*, and let e be a minimal experiment (with respect to $\preceq$) such that $C(e) \neq C^*(e)$. Using the acyclicity of C, there exists a wire w that is free in e and its inputs (in C) are fixed in e. Let σ be the value that w takes for experiment e in C, and let τ be the value that w takes for experiment e in C^*. Because e is minimal, $\sigma \neq \tau$.

Now $C(e) = C(e|_{w=\sigma})$ and $C^*(e) = C^*(e|_{w=\tau})$, but because e is minimal, we must have $C(e|_{w=\sigma}) = C^*(e|_{w=\sigma})$, so $C^*(e|_{w=\sigma}) = C(e) \neq C^*(e) = C^*(e|_{w=\tau})$ and e distinguishes σ from τ for w. Thus, because the distinguishing tables used by Construct-Circuit are complete and correct, there must be a distinguishing path π for (σ, τ) in T_w.

Consider the set V of inputs of w in C^*. If in the experiment e_π the wires in V are set to the values they take in e in C^*, then the output of C^* is $C^*(e|_{w=\tau})$. If V' is the set of inputs of w in C, then $V' \subseteq V$, and if in the experiment e_π the wires in V' are set to their fixed values in e, then the output of C^* is $C^*(e|_{w=\sigma})$, where σ is the representative value chosen by Construct-Circuit. Thus, there must be a wire $v \in V - V'$ relevant with respect to π, but then v would have been added to the circuit graph as an input to w when π was processed, a contradiction. Thus, C is 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 5.

4 Circuits with Bounded Shortcut Width

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

Theorem 13. *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, the three circuits given in Figure 1 of [7] are behaviorally equivalent, but have different graphs; 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 [7] to build the output circuit C.

A gate with gate function g and input wires $u_1, \ldots, u_\ell$ is **wrong** for w in C^* if there exists an experiment e in which the wires $u_1, \ldots, u_\ell$ are fixed,

say to values $u_j = \sigma_j$, and w is free, and there is an experiment e such that $C^*(e) \neq C^*(e|_{w=g(\sigma_1,\ldots,\sigma_\ell)})$, and is **correct** otherwise. The experiment e, which we term a **witness experiment** for this gate and wire, shows that no circuit C using this gate for w can be behaviorally equivalent to C^*. A set E of experiments is **sufficient** for C^* if for every wire w and every candidate gate that is wrong for w, E contains a witness experiment for this gate and this wire.

Lemma 14. *[7] If the input E to Circuit Builder is a sufficient set of experiments for C^*, then the circuit C that it outputs is behaviorally equivalent to C^*.*

The need to guarantee witness experiments for all possible wrong gates means that the Shortcuts Algorithm will learn a set of distinguishing tables for the restriction of C^* obtained by fixing $u_1, \ldots, u_\ell$ to values $\sigma_1, \ldots, \sigma_\ell$ for every choice of at most k wires u_j and every choice of assignments of fixed values to them.

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^*.

Because e is minimal, w must take on two different values, say τ_1 and τ_2 in $e|_{v=\sigma_1}$ and $e|_{v=\sigma_2}$ respectively.. Moreover, $e|_{v=\sigma_1}$ must be a minimal experiment distinguishing τ_1 from τ_2 for w; this purely behavioral property is both necessary and sufficient for a pair (v, w) to be a discoverable edge.

Lemma 15. *The pair (v, w) is a discoverable edge of C^* if and only if there is an experiment e and values $\sigma_1, \sigma_2, \tau_1, \tau_2$ such that e is a minimal experiment distinguishing σ_1 from σ_2 for v, and $e|_{v=\sigma_1}$ is a minimal experiment distinguishing τ_1 from τ_2 for w.*

We now generalize the concept of distinguishing paths to leave potential shortcut wires unassigned. Assume that C^* is a circuit, with n wires, an alphabet Σ of s symbols, fan-in bound k, and shortcut width bound b. A **test path with shortcuts** π is a directed path of wires from some wire w to the output, a set S of **side wires** assigned fixed values from Σ, and a set K of **cut wires** such that S and K are disjoint and neither contains w, and each wire in $S \cup K$ is an input to at least one wire beyond w in the directed path of wires. One intuition for this is that the wires in K could

have been set as side wires, but we are treating them as possible shortcut wires, not knowing whether they will end up being shortcut wires or not. As before, we define p_π to be the partial experiment setting all the wires in the directed path to $*$ and all the wires in S to the specified fixed values. Also, e_π is the experiment that extends p_π by setting every unspecified wire to $*$. The **length** of π is the number of edges in its directed path of wires.

Let π be a test path with shortcuts of nonzero length, with directed path $v_1, v_2, \ldots, v_r$, side wires S and cut wires K. The **1-suffix** of π is the test path π' obtained as follows. The directed path is $v_2, \ldots, v_r$, the side wires S' are all elements of S that are inputs to at least one of $v_3, \ldots, v_r$, and the cut wires K' are all elements of $K \cup \{v_1\}$ that are inputs to at least one of $v_3, \ldots, v_r$. If $t < r$, the t-suffix of π is obtained inductively by taking the 1-suffix of the $(t-1)$-suffix of π. A **suffix** of π is the t-suffix of π for some $1 \leq t < r$.

If π is a test path with shortcuts 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 that agrees with π and fixes all the wires in V, the value output by C^* depends only on the values assigned to the wires in V. Then π is **isolating** if the set of wires $\{w\} \cup K$ is functionally determining for π. Note that if we assign fixed values to all the wires in K, we get an isolating test path for w.

Lemma 16. *Let π be an isolating test path with shortcuts. If π' is any suffix of π then π' is isolating.*

Proof. Let π have directed path $v_1, \ldots, v_r$, side wires S and cut wires K. Let π' be the 1-suffix of π, with side wires S' and cut wires K'. The values of v_1 and K determine the output of C^* in any experiment that agrees with π. The only wires in $\{v_1\} \cup K$ that are not in K' are inputs of v_2 that are not also inputs of some $v_3, \ldots, v_r$. Similarly, the only wires in $S - S'$ are inputs of v_2 that are not also inputs of some $v_3, \ldots, v_r$. By setting the value of v_2, we make these input wires irrelevant, so $\{v_2\} \cup K'$ are functionally determining for π'. □

In this setting, what we want to distinguish are pairs of assignments to (w, B), where B is a set of wires not containing w. An **assignment** to (w, B) is just a function with domain $\{w\} \cup B$ and co-domain Σ. If a is an assignment to (w, B) and e is an experiment mapping w and every wire in B to $*$, then by $(e|_a)$ we denote the experiment e' such that $e'(v) = a(v)$ if $v \in \{w\} \cup B$ and $e'(v) = e(v)$ otherwise. If a_1 and a_2 are two assignments to

(w, B), then the experiment e **distinguishes** a_1 from a_2 if e maps $\{w\} \cup B$ to $*$ and $C^*(e|_{a_1}) \neq C^*(e|_{a_2})$.

Let π be a **distinguishing path with shortcuts** with initial path wire w, side wires S and cut wires K. Then π is **distinguishing** for the pair (w, B) and assignments a_1 and a_2 to (w, B) if $K \subseteq B$, $B \cap S = \emptyset$, π is isolating and e_π distinguishes a_1 from a_2. If such a path exists, we say (w, B) is **distinguishable** for a_1 and a_2. Note that this condition requires that π not set any of the wires in B. When $B = \emptyset$, these definitions reduce to the previous ones.

4.1 The Shortcuts Algorithm

Overview of algorithm. We assume that at most k wires $u_1, \ldots, u_\ell$ have been fixed to values $\sigma_1, \ldots, \sigma_\ell$, and denote by C^* the resulting circuit. The process described is repeated for every choice of wires and values. Like the Distinguishing Paths Algorithm, the Shortcuts Algorithm builds a directed graph G whose vertices are the wires of C^*, in which an edge (v, w) is added when v is discovered to be an input to w in C^*; one aim of the algorithm is to find all the discoverable edges of C^*.

Distinguishing tables. The Shortcuts Algorithm maintains a distinguishing table T_w for each wire w. Each entry in T_w is indexed by a triple, (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 to (w, B). If an entry exists for index (B, a_1, a_2), it contains π, a distinguishing path with shortcuts that is distinguishing for (w, B), a_1 and a_2. The entry also contains a bit marking the entry as processed or unprocessed.

Initialization. The distinguishing table T_{w_n} for the output wire is initialized with entries indexed by $(\emptyset, \{w_n = \sigma\}, \{w_n = \tau\})$ for every pair of distinct symbols $\sigma, \tau \in \Sigma$, each containing the distinguishing path of length 0 with no side wires and no cut wires. Each such entry is marked as unprocessed. All other distinguishing tables are initialized to be empty.

While there is an entry in some distinguishing table T_w marked as unprocessed, say with index (B, a_1, a_2) and π the corresponding distinguishing path with shortcuts, the Shortcuts Algorithm processes it and marks it as processed. To process it, the algorithm first uses the entry try to discover any new edges (v, w) to add to the graph G; if a new edge is added, all of the existing entries in the distinguishing table for wire w are marked as unprocessed. Then the algorithm attempts to find new distinguishing paths

with shortcuts obtained by extending π in all possible ways. If an extension is found to a test path with shortcuts π' that is distinguishing for (w', B'), a'_1 and a'_2, if there is not already an entry for (B', a'_1, a'_2), or, if π' is of shorter length that the existing entry for (B', a'_1, a'_2), then its entry is updated to π' and marked as unprocessed. When all possible extensions have been tried, the algorithm marks the entry in T_w for (B, a_1, a_2) as processed.

In contrast to the case of the Distinguishing Paths Algorithm, the Shortcuts Algorithm tries to find a *shortest* distinguishing path with shortcuts for each entry in the table. When no more entries marked as unprocessed remain in any distinguishing table, the algorithm constructs a set of experiments E as described below, calls Circuit Builder on E, outputs the resulting circuit C, and halts.

Processing an entry. Let (B, a_1, a_2) be the index of an unprocessed entry in a distinguishing table T_w, with corresponding distinguishing path with shortcuts, π, where the side wires of π are S and the cut wires are K. Note that $K \subseteq B$ and $S \cap B = \emptyset$. Let the set E_π consist of every experiment that agrees with π, arbitrarily fixes the wires in K, and arbitrarily fixes up to $2k$ additional wires not in K and not set by p_π, and sets the remaining wires free. There are $O((ns)^{2k}s^b)$ experiments in E_π; the algorithm makes a value injection query for each of them.

Finding relevant inputs. For every assignment a of fixed values to K, the resulting path π_a is an isolating test path for w. We use the Find-Inputs procedure (in Section 3.3) to find relevant inputs to w with respect to π_a, and let $V^*(\pi)$ be the union of the sets of wires returned by Find-Inputs over all assignments a to K. For each $v \in V^*(\pi)$, add the edge (v, w) to G if it is not already present, and mark all existing entries in all the distinguishing tables for wire w as unprocessed.

Lemma 17. *The wires in $V^*(\pi)$ are inputs to w and the wires in $V^*(\pi) \cup K$ are functionally determining for π.*

Proof. This follows from Lemma 8, because for each assignment a to K, the resulting π_a is an isolating path for w, and any wires in the set returned by Find-Inputs are indeed inputs to w. Also, for each assignment a to K, the set $V(\pi_a)$ is functionally determining for π_a, and is contained in $V^*(\pi)$. □

Additional input test. The Shortcuts Algorithm makes an additional input test if π distinguishes two assignments a_1 and a_2 such that there is

a wire $w' \in K$ such that a_1 and a_2 agree on every wire other than w. Let π' be the distinguishing path obtained from π by fixing every wire in $K - \{w\}$ to its value in a_1. If there is an experiment e agreeing with π' and setting w to $*$ and fixing every element of $V(\pi')$, and two values σ_1 and σ_2 such that $C^*(e|_{v=\sigma_1}) \neq C^*(e|_{v=\sigma_2})$, and, moreover, for every $\tau \in \Sigma$, $C^*(e|_{w=\tau,v=\sigma_1}) = C^*(e|_{w=\tau,v=\sigma_2})$, then add edge (v, w) to G if it is not already present, and mark all the existing entries in the distinguishing table for wire w as unprocessed.

Lemma 18. *If edge (v, w) is added to G by this additional input test, then v is an input of w in C^*.*

Proof. Note that w must take two different values, say τ_1 and τ_2, in the experiments $e|_{v=\sigma_1}$ and $e|_{v=\sigma_2}$; thus, w must be a descendant of v. Moreover, $C^*(e|_{w=\tau_1,v=\sigma_1}) = C^*(e|_{w=\tau_1,v=\sigma_2})$ and $C^*(e|_{w=\tau_2,v=\sigma_1}) = C^*(e|_{w=\tau_2,v=\sigma_2})$, from which we conclude that $C^*(e|_{w=\tau_1,v=\sigma_1}) \neq C^*(e|_{w=\tau_2,v=\sigma_1})$.

If v is not an input of w, then let U be the set of all inputs of w. In e, if we set $v = \sigma_1$ and $w = *$ and the wires in U as induced by $e|_{v=\sigma_1}$, then $w = \tau_1$ and the output of C^* is $C^*(e|_{w=\tau_1,v=\sigma_1})$. If we then change the values on wires in U one by one to their values in $e|_{v=\sigma_2}$, because the final result have $w = \tau_2$ and output $C^*(e|_{v=\sigma_1,w=\tau_2})$, there must be an input u such that fixing the other inputs to w and changing u's value changes the output with respect to $e|_{v=\sigma_1}$. Thus, u is a relevant input with respect to the distinguishing path $\pi_{\{v=\sigma_1\}}$, and must be in the set $V(\pi)$. This is a contradiction, because wires in $V(\pi)$ are fixed in e, and u must change value from $e|_{v=\sigma_1}$ and $e|_{v=\sigma_2}$. Thus v must be an input of w. □

Extending a distinguishing path. After finding as many inputs of w as possible using π, the Shortcuts Algorithm attempts to extend π as follows. Let $I_G(w)$ be the set of all inputs of w in G. For each pair (w', K') such that $w' \in I_G(w)$ and K' is a set of at most b wires not containing w' such that $K' \subseteq I_G(w) \cup K$ and K' is disjoint from the path wires and side wires of π, we let $S_0 = (K \cup V^*(\pi)) - (\{w'\} \cup K')$. Note that the set of wires in $S_0 \cup K' \cup \{w'\}$ is functionally determining for π.

For each assignment a of fixed values to S_0, the algorithm extends π to π' as follows. It adds w' to the start of the directed path, adds S_0 to the set of side wires (fixed to the values assigned by a) and takes the cut wires to be K'. Note that every wire in K' is an input to some wire beyond w on the path. Because w' is an input of w, and all of the wires in $V^*(\pi) \cup K$ are accounted for among (w, K') and S', and all of the wires in S' are inputs to

w or wires beyond w on the path, the result is an isolating test path with shortcuts for (w', K').

The algorithm then searches through all 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 to (w', B'), to discover whether π' is distinguishing for (w', B'), a'_1 and a'_2. If so, the algorithm checks the distinguishing table $T_{w'}$ and creates or updates the entry for index (B', a'_1, a'_2) as follows. If there is no such entry, one is created with π'. If there already is an entry and π' is shorter than the path in the entry, then the entry is changed to contain π'. If the entry is created or changed by this operation, it is marked as unprocessed. When all possible extensions of π have been tried, the entry in T_w for (B, a_1, a_2) is marked as processed.

Correctness and completeness. We define the distinguishing table T_w to be **correct** if whenever π is an entry in T_w for (B, a_1, a_2), then π is a distinguishing path with shortcuts that is distinguishing for (w, B), a_1 and a_2. For each wire w, let $B(w)$ denote the set of shortcut wires of w in the target circuit C^*. If π is a distinguishing path with shortcuts such that every edge in its directed path is discoverable, we say that π is **discoverable**. The distinguishing T_w table is **complete** if for every pair a_1 and a_2 of assignments to $(w, B(w))$ that are distinguishable by a discoverable path, there is an entry in T_w for index $(B(w), a_1, a_2)$.

Lemma 19. *When Shortcuts Algorithm finishes the processing of the distinguishing tables, every distinguishing table T_w is correct and complete.*

Proof. The correctness follows inductively from the correctness of the initialization of T_{w_n} by the arguments given above. To prove completeness, we prove the following stronger condition about the distinguishing tables when the Shortcuts Algorithm finishes processing them: (1) for every wire w and every pair a_1 and a_2 of assignments to $(w, B(w))$ that are distinguishable by a discoverable path, the entry for $(B(w), a_1, a_2)$ is a shortest discoverable distinguishing path with shortcuts that is distinguishing for $(w, B(w))$, a_1 and a_2.

Condition (1) clearly holds for T_{w_n} after it is initialized, and this table does not change thereafter. Assume to the contrary that condition (1) does not hold and let w be a wire of the smallest possible depth such that T_w does not satisfy condition (1). Note that w is not the output wire.

There must be assignments a_1 and a_2 for $(w, B(w))$ that are distinguishable by a discoverable path such that in T_w, the entry for $(B(w), a_1, a_2)$

is either nonexistent or not as short as possible. Let π be a shortest possible discoverable distinguishing path with shortcuts that is distinguishing for $(w, B(w))$, a_1 and a_2. Let S be the side wires of π, with assignment a, and let K be the cut wires of π. Then we have $K \subseteq B$ and $S \cap B = \emptyset$. Because w is not the output wire, the directed path in π is of length at least 1. Let π' be the 1-suffix of π, with initial vertex w', side wires S' and cut wires K'. Note that (w, w') must be a discoverable edge and that π' is also discoverable. By Lemma 16, π' is isolating.

For any two assignments a'_1 and a'_2 to $(w', B(w'))$ such that $a'_j(u)$ is the value of u in $e_\pi|_{a_j}$ for each $u \in \{w'\} \cup K'$, we have that π' is distinguishing for $(w', B(w'))$, a'_1 and a'_2. To see this, note that $\{w'\} \cup K'$ is functionally determining for π', so $C^*(e_{\pi'}|_{a'_j}) = C^*(e_\pi|_{a_j})$ for $j = 1, 2$, and these latter two values are distinct. Let a'_j denote the assignment to $(w', B(w'))$ induced by the experiment $e_\pi|_{a_j}$ for $j = 1, 2$; these two assignments have the required property.

Because the depth of w' is smaller than the depth of w, condition (1) must hold for $T_{w'}$, and the distinguishing table for $T_{w'}$ must contain an entry for $(B(w'), a'_1, a'_2)$ that is a shortest discoverable distinguishing path with shortcuts π'' that is distinguishing for $B(w')$, a'_1 and a'_2. Note that the length of π'' is at most the length of π minus 1.

We argue that the discoverable edge (w, w') must be added to G by the Shortcuts Algorithm. This edge is the first edge on a minimal experiment e distinguishing σ_1 from σ_2 for w. This corresponds to a distinguishing path ρ with no cut edges distinguishing σ_1 from σ_2 for w, and every edge of this path is also discoverable. There are two cases, depending on whether w is a shortcut of w' on the path or not.

If w is not a shortcut edge of w' on the path, then the 1-suffix of ρ will be a discoverable distinguishing path with no cut edges that is distinguishing for w', τ_1, and τ_2, where these are the values w' takes in $e|_{w=\sigma_j}$ for $j = 1, 2$. Because condition (1) holds for $T_{w'}$, there will be an entry in $T_{w'}$ containing a distinguishing path with shortcuts for $(w', B(w'))$ that distinguishes the two assignments that set $B(w')$ as in e and set w' to τ_1 and τ_2. Because w is a relevant input with respect to ρ, the edge (w, w') will be added to G if it is not already present when ρ is processed.

If w is a shortcut edge of w' on the path, then the 1-suffix of ρ will be a discoverable distinguishing path with cut edges $\{w\}$ that is distinguishing for the assignments $\alpha_1 = \{w = \sigma_1, w' = \tau_1\}$ and $\alpha_2 = \{w = \sigma_2, w' = \tau_2\}$. Because $w \in B(w')$ and $T_{w'}$ satisfies condition (1), there will be an entry ρ in $T_{w'}$ for $(w', B(w'))$ that distinguishes the two assignments to $(w', B(w'))$

that agree with α_1 and α_2 on w' and w, and set every other element of $B(w')$ as in e. When the entry ρ is processed, the additional input test will discover the edge (w, w') and add it to the graph G if it is not already present. In fact, this shows that every discoverable edge (v, w') will eventually be discovered by the algorithm because $T_{w'}$ is complete.

Thus, we can be sure that the entry π'' will be (re)processed when every discoverable edge (v, w') is present in G, including (w, w'). When this happens, the entry π'' will be extended to a distinguishing path with shortcuts that is distinguishing for $(w, B(w))$, a_1 and a_2 and has length at most that of π. To see that this holds, note that if v is a side wire of π'', then it cannot be an ancestor of w' because otherwise it is a shortcut wire of w' and in $B(w')$, which is disjoint from the side wires of π''. Thus, the side wires of π'' cannot include any input of w' or any wire in $B(w)$, because all these wires are ancestors of w'. Moreover, since all the discoverable inputs to w' have been added to G, one of the possible extensions of π'' will set (some of) the inputs of w' in such a way that moving from assignment a_1 to assignment a_2 to $(w, B(w))$ with the other side gate settings of π'' will move from a'_1 to a'_2 for $(w', B(w'))$,

Thus, the entry (B, a_1, a_2) will exist and be of length at most the length of π when the algorithm finishes processing the distinguishing tables. This contradiction shows that all the distinguishing tables must be complete.

□

Building a circuit. When all the entries in all the distinguishing tables are marked as processed, the Shortcuts Algorithm constructs a set E of experiments. For every table T_w and every distinguishing path π for (B, a_1, a_2) in the table such that $a_1(u) = a_2(u)$ for every $u \in B$, and every set V of at most k wires not set by p_π and every assignment a to V, add to E the experiment $e_\pi|_a$, that extends e_π by the assignment a. After iterating the above process over all possible choices of at most k wires $u_1, \ldots, u_\ell$ and assignments to them, the algorithm takes the union of all the resulting sets of experiments E and calls Circuit Builder [7] on this union and outputs the returned circuit C and halts.

Lemma 20. *The circuit C is behaviorally equivalent to the target circuit C^*.*

Proof. We show that the completeness of the distinguishing tables implies that the set E of experiments is sufficient, and apply Lemma 14 to conclude that C is behaviorally equivalent to C^*. Suppose a gate g with inputs

$u_1, \ldots, u_\ell$ is wrong for wire w in C^*. Then there exists a minimal experiment e that witnesses this; e fixes all the wires $u_1, \ldots, u_\ell$, say as $u_j = \sigma_j$ for $j = 1, \ldots, \ell$, sets the wire w free and is such that $C^*(e) \neq C^*(e|_{w=g(\sigma_1,\ldots,\sigma_\ell)})$.

Consider the iteration of the table-building process for the circuit C^* with the restriction $u_j = \sigma_j$ for $j = 1, \ldots, \ell$. In this circuit, e distinguishes between $w = \sigma$ and $w = \tau$, where σ is the value w takes in C^* for e, and $\tau = g(\sigma_1, \ldots, \sigma_\ell)$. Note that the free wires of e form a directed path of discoverable edges. Because the table T_w is complete, there will be a distinguishing path π with shortcuts for $(w, B(w))$ for assignments a_1 and a_2 where $a_1(w) = \sigma$ and $a_2(w) = \tau$, and $a_1(v) = a_2(v)$ for all $v \in B(w)$. For every input v of w in C^* that is not among $u_1, \ldots, u_\ell$, π does not set v, because it only sets wires that are inputs to descendants of w, and any input of w that is an input of a descendant of w is a short cut wire of w and therefore in $B(w)$. However, π does not set any wires in $B(w)$. Thus, among the choices of sets of at most k wires and values to set them to, there will be one that sets just the inputs (in C^*) of w as in e. The corresponding experiment e' in E will be a witness experiment eliminating the gate g with inputs $u_1, \ldots, u_\ell$, so the set of experiments to Circuit Builder is sufficient for C^*. □

Running time. To analyze the running time of the Shortcuts Algorithm, note that there are $O(n^k s^k)$ choices of at most k wires and values from Σ to fix them to; this bounds the number of iterations of the table building process. In each iteration, there are $O(n^{b+1}s^{2b+2})$ total entries in the distinguishing tables. Each entry in a distinguishing table may be processed several times: when it first appears in the table, and each time its distinguishing path is replaced by a shorter one, and each time a new input of w is discovered, for a total of at most $n + k$ times. Thus, the total number of entry-processing events by the algorithm in one iteration is $O((n + k)n^{b+1}s^{2b+2})$. Each such event makes $O((ns)^{2k}s^b)$ value injection queries, so $O((n + k)n^{2k+b+1}s^{2k+3b+2})$ value injection queries are made by the algorithm in each iteration, for a total of $O((n + k)n^{3k+b+1}s^{3k+3b+2})$ value injection queries made by the Shortcuts Algorithm. The number of experiments given as input to Circuit Builder is $O(n^{2k+b+1}s^{2k+2b+2})$, because each final entry may give rise to at most $O(n^k s^k)$ experiments in E in each iteration. This concludes the proof of Theorem 13.

5 Learning Analog Circuits via Discretization

We first give a simple example of an analog circuit. We then 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 13, to get a polynomial-time algorithm for approximately learning an analog circuit with logarithmic depth, bounded fan-in and bounded shortcut width.

5.1 Example of an analog circuit

For example, let $\wedge(x, y) = xy$ for all $x, y \in [0, 1]$ and let $\vee(x, y) = x + y - xy$ for all $x, y \in [0, 1]$. (Note that these are polynomial representations of conjunction and disjunction when restricted to the values 0 and 1.) Then $\wedge$ and $\vee$ are analog functions of arity 2, and we define a circuit with 6 wires as follows. Let g_1 be the constant function 0.1, g_2 be the constant function 0.6 and g_3 be the constant function 0.8. These functions assign default values to the corresponding wires. Let g_4 be the function $\vee$, and let its pair of inputs be w_1, w_2. Let g_5 be the function $\vee$, and let its pair of inputs be w_2, w_3. Finally, let w_6 be the function $\wedge$, and let its pair of inputs be w_4, w_5. If we consider the experiment e_0 that assigns $*$ to every wire, we calculate the values $w_i(e_0)$ as follows. Using their default values,

$$w_1(e_0) = 0.1, w_2(e_0) = 0.6, w_3(e_0) = 0.8.$$

Then, because the inputs to w_4 and w_5 have defined values,

$$w_4(e_0) = \vee(0.1, 0.6) = 0.64, w_5(e_0) = \vee(0.6, 0.8) = 0.92.$$

Because the inputs to w_6 now have defined values,

$$w_6 = \wedge(0.64, 0.92) = 0.5888.$$

If we consider the experiment e_1 that fixes the value of w_5 to 0.2 and assigns $*$ to every other wire, then as before,

$$w_1(e_1) = 0.1, w_2(e_1) = 0.6, w_3(e_1) = 0.8, w_4(e_1) = 0.64.$$

However, because the value of w_5 is fixed to 0.2 in e_1,

$$w_5(e_1) = 0.2, w_6(e_1) = \wedge(0.64, 0.2) = 0.128.$$

5.2 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

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

For example, the function $\wedge(x, y) = xy$ satisfies a Lipschitz condition with constant 2. A Lipschitz condition on an analog function allows us to bound the error of a discrete approximation to the function. For more on Lipschitz conditions, see [14].

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. An easy consequence is the following.

Lemma 21. *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 $[0,1]$, $|g(x_1, \ldots, x_k) - g(D_m(x_1), \ldots, D_m(x_k))| \leq L/2m$.*

5.3 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 22. *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, $|C(e) - C'(D_M(e))| \leq 1/N$.*

Proof. Define $m_d = N(3L)^d$ for all nonnegative integers d; then $M = m_{d_{max}}$. We prove the stronger condition that for every experiment e for C and every wire w_i, if d is the depth of w_i, we have

$$|w_i(e) - w_i'(D_M(e))| \leq 1/m_d,$$

where $w_i(e)$ is the value of wire w_i in C for experiment e and $w_i'(D_M(e))$ is the value of wire w_i in C' for experiment $D_M(e)$. Because the output wire is at depth $d = 0$, this will imply that $C(e)$ and $C'(D_M(e))$ do not differ by more than $1/N$.

Let e be an arbitrary experiment for C. We proceed by downward induction on the depth d of w_i. When this quantity is d_{max}, the wire w_i is at maximum depth and has no inputs. The wire w_i is fixed in e if and only if it is fixed in $D_M(e)$, and in either case, the values assigned to w_i agree to within $1/2M < 1/m_{d_{max}}$. Now consider w_i at depth d, assuming inductively that the condition holds for all wires at greater depth. If w_i is fixed in e then it is fixed in $D_M(e)$ and the values assigned to it differ by at most $1/2M$. If w_i is free in e then it is free in $D_M(e)$. Consider the input wires to w_i, say $w_{j_1}, \ldots, w_{j_s}$; these are all at depth at least $d + 1$, so by the inductive hypothesis

$$|w_{j_r}(e) - w_{j_r}'(D_M(e))| \leq 1/m_{d+1},$$

for $r = 1, \ldots, s$.

Note that

$$w_i(e) = g_i(w_{j_1}(e), \ldots, w_{j_s}(e))$$

and

$$w_i'(D_M(e)) = D_{m_d}(g_i(y_1, \ldots, y_s)),$$

where $y_r = D_{m_{d+1}}(w_{j_r}'(D_M(e)))$ for $r = 1, \ldots, s$. Note that by the properties of the discretization function,

$$|y_r - w_{j_r}'(D_M(e))| \leq 1/(2m_{d+1}).$$

By the Lipschitz condition on the gate function g_i we have

$$|g_i(w_{j_1}(e), \ldots, w_{j_s}(e)) - g_i(y_1, \ldots, y_s)| \leq L(3/2)(1/m_{d+1}) = 1/(2m_d),$$

because

$$|w_{j_r}(e) - y_r| \leq 1/m_{d+1} + 1/(2m_{d+1}).$$

Discretizing the output of g_i by D_{m_d} adds at most $1/(2m_d)$ to the difference, so

$$|g_i(w_{j_1}(e), \ldots, w_{j_s}(e)) - D_{m_d}(g_i(y_1, \ldots, y_x))| \leq 1/m_d,$$

that is,

$$|w_i(e) - w'_i(D_M(e))| \leq 1/m_d,$$

which completes the induction. □

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 23. *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 with a constant bound, and short-cut width bounded by a constant.*

6 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 [7], 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 AND, OR, NOT, 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 that 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.

An advantage of learning with counterexamples is its flexibility. As remarked in [7], if the counterexample queries return counterexamples that fix only the input wires of the circuit, learning algorithms output a circuit equivalent to the target circuit with respect to input/output behaviors. In general, the algorithms only output a circuit equivalent to the target with respect to the set of counterexamples presented to them. Moreover, the

algorithms presented in this section can be naturally generalized to work when more than one gate is observable. In this case, an experiment e is a counterexample if C and C^* compute one of the observable gates differently and the learning algorithm outputs a circuit that behaves the same way on all observable gates with respect to the given set of counterexamples.

6.1 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; we find a witness experiment e^* eliminating a candidate gate function g. 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 to specific values.

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 [7].

Lemma 24. *If e^* is a minimal counterexample for C, there exists a gate g in C such that e^* is a witness experiment for g.*

Proof. Because C is acyclic, there exists a gate g that is free in e^* such that all the inputs of g are fixed in e^*. Then e^* is a witness experiment for g, because otherwise we have $C^*(e^*|_{w=g(e^*)}) = C^*(e^*) \neq C(e^*) = C(e^*|_{w=g(e^*)})$, which contradicts the minimality of e^*. □

Although it does the job, the procedure **minimize** may fix many more wires than necessary. (In Section 6.2 we will describe a different algorithm that will fix many fewer wires for certain classes of circuits.)

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 [7]. 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 $s = |\Sigma|$ 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 from a constant gate to the wire in the target circuit C^*. The following lemma shows an interesting connection between levels and numbers of conflicts.

Lemma 25. *The number of conflicts each wire receives is bounded above by its level.*

Proof. A conflict happens to a wire w only when the candidate input wires do not contain all input wires of the true gate of w. Therefore, constant gates, whose levels are zero, have no conflict. Assuming the lemma is true

for all wires with level no higher than i, for a level i wire w, at the point w has i conflicts, all the input wires of w's true gates have fewer conflicts than w and thus are considered as candidate input wires for w by our algorithm. Therefore, w can not have more than i conflicts. □

Corollary 26. *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 27. *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 25 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. □

We make the learnability assumption that *each gate is polynomial time learnable with counterexamples*. The model of learning with counterexamples are also known as the mistake-bound model [18]. Circuits with constant fan-in gates are learnable in this model, even with large alphabets. To see this, note that there are at most $\binom{n}{k}$ many choices of input wires. For each combination of k input wires, the gate function is fully specified by the s possible outputs associated with each of the s^k possible settings to the k inputs. Each counterexample will eliminate one of the s possible outputs for one of the s^k settings to the inputs.

More efficient algorithms exist when we have prior knowledge of hypothesis space. For example, the "halving" algorithm and the "optimal" algorithm are able to cut the hypothesis space in half whenever they receive a mistake [18]. Although computationally expensive in general, these algorithms can be efficient when the hypothesis space is small which can be true in a practical application of our circuit learning algorithms. In fact, there exists a large body of work in the counterexample or mistake-bound learning literature and many efficient learning algorithms exist. For example, AND/OR functions are certainly learnable in this model. Furthermore,

halfspaces and boxes with finite domain and threshold functions [19] are also learnable in this model.

6.2 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 and other applications. 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.

7 Discussion

In this paper, we extended the results of Angluin et al. [7] to the large-alphabet setting under the value injection query model. We showed topological conditions under which large-alphabet circuits are efficiently learnable and gave evidence that the conditions for shortcut width that we consider are necessary. We also showed that analog circuits can be approximated by large alphabet circuits, and that they can be approximately learned given a restriction on their depth. We improved on the results of [7] for the case when counterexamples are added, and we extended some of the results to the large-alphabet case.

Now that small-alphabet, large-alphabet, and analog circuits have been studied under the value-injection model, we plan to consider Bayesian circuits, where probabilities are attached to the gates. Another interesting direction to explore is possible implications of this work to complexity theory. For example, does the class of circuits that are efficiently learnable with value injection queries represent a natural class of problems?

8 Acknowledgments

A preliminary version of this paper was presented at COLT 2007 [6]. We would like to thank the anonymous reviewers of that version and of the present paper for their helpful suggestions and revisions.

References

[1] Alfred V. Aho, M. R. Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM J. Comput.*, 1:131–137, 1972.

[2] Tatsuya Akutsu, Satoru Kuhara, Osamu Maruyama, and Satoru Miyano. Identification of genetic networks by strategic gene disruptions and gene overexpressions under a boolean model. *Theor. Comput. Sci.*, 298(1):235–251, 2003.

[3] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[4] D. Angluin, M. Frazier, and L. Pitt. Learning conjunctions of Horn clauses. *Machine Learning*, 9:147–164, 1992.

[5] D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. *J. ACM*, 40:185–210, 1993.

[6] Dana Angluin, James Aspnes, Jiang Chen, and Lev Reyzin. Learning large-alphabet and analog circuits with value injection queries. In *Twentieth Annual Conference on Learning Theory*, pages 51–65, June 2007.

[7] Dana Angluin, James Aspnes, Jiang Chen, and Yinghua Wu. Learning a circuit by injecting values. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, pages 584–593, New York, NY, USA, 2006. ACM Press.

[8] Dana Angluin and Michael Kharitonov. When won't membership queries help? *J. Comput. Syst. Sci.*, 50(2):336–355, 1995.

[9] Nader H. Bshouty. Exact learning boolean functions via the monotone theory. *Inf. Comput.*, 123(1):146–153, 1995.

[10] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[11] T. Ideker, V. Thorsson, and R Karp. Discovery of regulatory interactions through perturbation: Inference and experimental design. In *Pacific Symposium on Biocomputing 5*, pages 302–313, 2000.

[12] Jeffrey C. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *J. Comput. Syst. Sci.*, 55(3):414–440, 1997.

[13] Jeffrey C. Jackson, Adam R. Klivans, and Rocco A. Servedio. Learnability beyond AC0. In *STOC '02: Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 776–784, New York, NY, USA, 2002. ACM Press.

[14] H. Jeffreys and B. Jeffreys. *Methods of Mathematical Physics*. Cambridge University Press, Cambridge, England, 3rd. edition, 1988.

[15] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *J. ACM*, 41(1):67–95, 1994.

[16] Michael Kharitonov. Cryptographic hardness of distribution-specific learning. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 372–381, New York, NY, USA, 1993. ACM Press.

[17] Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, Fourier transform, and learnability. *Journal of the ACM*, 40(3):607–620, 1993.

[18] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Mach. Learn.*, 2(4):285–318, 1988.

[19] W. Maass and G. Turan. On the complexity of learning from counterexamples. In *the 30th Annual Symposium on Foundations of Computer Science*, pages 262–267, 1989.

[20] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.