Algorithms for Circuits and Circuits for Algorithms: Connecting the Tractable and Intractable[*]

Ryan Williams[†]

Abstract. The title of this paper highlights an emerging duality between two basic topics in algorithms and complexity theory.

Algorithms for circuits refers to the design of algorithms which can analyze finite logical circuits or Boolean functions as input, checking a simple property about the complexity of the underlying function. For instance, an algorithm determining if a given logical circuit C has an input that makes C output *true* would solve the NP-complete Circuit-SAT problem. Such an algorithm is unlikely to run in polynomial time, but could possibly be more efficient than exhaustively trying all possible inputs to the circuit.

Circuits for algorithms refers to the modeling of "complex" uniform algorithms with "simple" Boolean circuit families, or proving that such modeling is impossible. For example, can every exponential-time algorithm be simulated using Boolean circuit families of only polynomial size? It is widely conjectured that the answer is *no*, but the present mathematical tools available are still too crude to resolve this kind of separation problem.

This paper surveys these two generic subjects and the connections that have been developed between them, focusing on connections between non-trivial circuit-analysis algorithms and proofs of circuit complexity lower bounds.

Mathematics Subject Classification (2010). Primary 68Q17; Secondary 68Q25.

Keywords. circuit complexity; algorithm analysis; satisfiability; lower bounds; derandomization; learning; exact algorithms; parameterized algorithms

1. Introduction

Budding theoretical computer scientists are generally taught several dictums at an early age. One such dictum is that the algorithm designers and the complexity theorists (whoever they may be) are charged with opposing tasks. The algorithm designer discovers interesting methods for solving certain problems; along the way, she may also propose new notions of what is interesting, to better understand the

[*]This article is an adaptation of an invited paper in the proceedings of the 2014 IEEE Conference on Computational Complexity.

[†]The author thanks Boaz Barak, Lance Fortnow, Valentine Kabanets, and Rahul Santhanam for comments on an earlier version. The author was supported by the Alfred P. Sloan foundation, Microsoft Research, and the NSF under grant CCF-1212372.

scope and power of algorithms. The complexity theorist is supposed to prove *lower bounds*, showing that sufficiently interesting methods for solving certain problems do not exist. Barring that, he develops a structural framework that explains the consequences of such impossibility results, as well as consequences of possessing such interesting methods.

Another dictum is that algorithm design and analysis is, on the whole, an easier venture than proving lower bounds. In algorithm design, one only has to find a single efficient algorithm that will solve the problem at hand, but a lower bound must reason about *all* possible efficient algorithms, including bizarrely behaving ones, and argue that none solve the problem at hand. This dictum is also reflected in the literature: every year, many interesting algorithms are discovered, analyzed, and published, compared to the tiny number of lower bounds proved.[1] Furthermore, there are rigorously mathematical reasons for believing that lower bounds are hard to prove. The most compelling of these are the three "barriers" of Relativization [BGS75], Natural Proofs [RR97], and Algebrization [AW09]. These "no-go" theorems demonstrate that the known lower bound proof methods are simply too coarse to prove even weak lower bounds, much weaker than $\mathsf{P} \neq \mathsf{NP}$. Subsequently, complexity theory has been clouded with great pessimism about resolving some of its central open problems.

While the problems of algorithm design and proving lower bounds may arise from looking at opposing tasks, the two tasks do have deep *similarities* when viewed in the appropriate way.[2] This survey will concentrate on some of the most counterintuitive similarities: from the design of certain algorithms (the supposedly "easier" task), one can derive new lower bounds (the supposedly "harder" task). That is, there are senses in which algorithm design is *at least as hard* as proving lower bounds, contrary to dictums. These connections present an excellent mathematical "arbitrage" opportunity for complexity theorists: to potentially prove hard lower bounds via supposedly easier algorithm design. (Moreover, there is money to be made: this approach *has* recently led to new lower bounds.)

Several connections take the following form:

The *existence* of an "efficient" algorithm T that can analyze *all* structured circuits C implies the *existence* of an "efficient" function f that is not computable by *all* structured circuit families.

Therefore, while algorithms and lower bounds are opposites by definition, there are situations where algorithm design for a problem X can be translated into "lower bound design" for another problem Y. The key is that there are two computational models under consideration here: the *algorithm model* or the usual "Turing" style model of algorithms, and the *circuit model* or the non-uniform circuit family model, which we shall define shortly. Careful design of algorithms for analyzing instances of the circuit model are used to construct functions computable (in one sense) in

[1]Of course, there can be other reasons for this disparity, such as funding.

[2]Similarities are already present in the proof(s) that the Halting Problem is undecidable: such results rely on the construction of a *universal Turing machine* that can run arbitrary Turing machine code given as input. This is a textbook application of how an algorithm can be used to prove an impossibility theorem.

the algorithm model that are uncomputable (in another sense) in the circuit model. There is a kind of duality lurking beneath which is not well-understood.

The focus of this article is on two generic topics in algorithms and complexity, and connections between them:

- *Circuits for Algorithms* refers to the modeling of powerful uniform algorithms with non-uniform circuit families, or proving that such modeling is impossible. For instance, the celebrated EXP versus P/poly question asks if exponential-time algorithms can be simulated using non-uniform circuit families of polynomial size. Complexity theorists believe that the answer is *no*, but they presently have no idea how to prove such a circuit lower bound.
- *Algorithms for Circuits* refers to the design of algorithms which can analyze finite logical circuits or Boolean functions as input, checking some property about the complexity of the underlying function. To illustrate, the problem Circuit-SAT asks if a given logical circuit has an input that forces the circuit to output *true*. Circuit-SAT is NP-complete and believed to be intractable; nevertheless, even "mildly intractable" algorithms for this problem would be useful in both theory and practice. It is an outstanding open question whether one can asymptotically improve over the "brute force" algorithm for Circuit-SAT which simply evaluates the circuit on all possible inputs. Recent surprising developments have shown that even tiny improvements over exhaustive search would significantly impact Circuits for Algorithms—in fact, new circuit lower bounds have been deduced from such algorithms.

The rest of the paper is organized as follows. The next section provides a bit of relevant background. Section 3 surveys *circuits for algorithms*, and Section 4 surveys *algorithms for circuits*. Section 5 discusses known connections between the two, and prospects for future progress. Section 6 briefly concludes.

2. Preliminaries

Recall $\{0,1\}^n$ is the set of all n-bit binary strings, and $\{0,1\}^\star = \bigcup_{n\in\mathbb{N}}\{0,1\}^n$.

A quick recollection of machine-based complexity Any reasonable algorithmic model with a coherent method for counting steps (such as Turing machines and their transition functions) will suffice for our discussion. For an algorithm A, we let $A(x)$ denote the output of A on the input x. A *language* L is a subset of $\{0,1\}^\star$; in the following, the variable L always denotes a language. We typically think of L as an indicator function from $\{0,1\}^\star$ to $\{0,1\}$, in the natural way.

Let $t : \mathbb{N} \to \mathbb{N}$. An algorithm A *runs in time* $t(n)$ if, on all $x \in \{0,1\}^n$, $A(x)$ halts within $t(|x|)$ steps. *Decidability of* L *in time* $t(n)$ means that there is an algorithm A running in time $t(n)$ such that $A(x) = L(x)$ for all x.

L is *verifiable in time* $t(n)$ if there exists an algorithm A such that, on all $x \in \{0,1\}^n$, $x \in L$ if and only if there is a $y_x \in \{0,1\}^{t(|x|)}$ such that $A(x, y_x)$ runs in time $O(t(|x|))$ and $A(x, y_x) = 1$. Intuitively, the string y_x serves as a *proof* that $x \in L$, and this proof can be verified in time $O(t(|x|))$.

An algorithm A *runs in space* $t(n)$ if, on all $x \in \{0,1\}^n$, the total workspace used by $A(x)$ is at most $t(|x|)$ cells (or registers, or bits, depending on the model). Decidability of a language in space $t(n)$ is defined in the obvious way.

Some complexity classes relevant to our discussion are:

- P: the class of languages decidable in $O(p(n))$ time for some $p \in \mathbb{Z}[x]$.
- NP: languages verifiable in $O(p(n))$ steps for some $p \in \mathbb{Z}[x]$.
- PSPACE: languages decidable in space $O(p(n))$ for some $p \in \mathbb{Z}[x]$.
- EXP: languages decidable in $O(2^{p(n)})$ time for some $p \in \mathbb{Z}[x]$.
- NEXP: languages verifiable in $O(2^{p(n)})$ time for some $p \in \mathbb{Z}[x]$.
- $\mathsf{EXPSPACE}$: languages decidable in space $O(2^{p(n)})$ for some $p \in \mathbb{Z}[x]$.

Let C be one of the above classes. An *algorithm A with oracle access to C* has a powerful extra instruction: there is a language $L \in C$ such that A can call $L(y)$ in one time step, on any input y of its choosing. (Intuitively, A can efficiently "consult the oracle" in class C for answers.) This is an interesting notion when C is a hard complexity class, say in NP or in PSPACE, and L is chosen to be a hard language in C.

Circuit complexity A function $f : \{0,1\}^n \to \{0,1\}$ is called *Boolean.* We let $x_1, \ldots, x_n$ denote the n variables to a Boolean function f. Circuit complexity is chiefly concerned with the difficulty of building up Boolean functions out of "simpler" functions, such as those of the form $g : \{0,1\}^2 \to \{0,1\}$. Examples of interesting Boolean functions include:

- $\mathrm{OR}_k(x_1, \ldots, x_k)$, $\mathrm{AND}_k(x_1, \ldots, x_k)$, with their usual logical meanings,
- $\mathrm{MODm}_k(x_1, \ldots, x_k)$ for a fixed integer $m > 1$, which outputs 1 if and only if $\sum_i x_i$ is divisible by m.
- $\mathrm{MAJ}_k(x_1, \ldots, x_k) = 1$ if and only if $\sum_i x_i \geq \lceil k/2 \rceil$.

A *basis set* $\mathcal{B}$ is a set of Boolean functions. Two popular choices for $\mathcal{B}$ are B_2, the set of all functions $g : \{0,1\}^2 \to \{0,1\}$, and U_2, the set B_2 without MOD2 and the negation of MOD2. A *Boolean circuit* of *size* s with n inputs $x_1, \ldots, x_n$ over basis $\mathcal{B}$ is a sequence of $n+s$ functions $C = (f_1, \ldots, f_{n+s})$, with $f_i : \{0,1\}^n \to \{0,1\}$ for all i, such that:

- for all $i = 1, \ldots, n$, $f_i(x_1, \ldots, x_n) = x_i$,
- for all $j = n+1, \ldots, n+s$, there is a function $g : \{0,1\}^k \to \{0,1\}$ from $\mathcal{B}$ and indices $i_1, \ldots, i_k < j$ such that

$$f_j(x_1, \ldots, x_n) = g(f_{i_1}(x_1, \ldots, x_n), \ldots, f_{i_k}(x_1, \ldots, x_n)).$$

The f_i are the *gates* of the circuit; $f_1, \ldots, f_n$ are the *input gates*, $f_{n+1}, \ldots, f_{n+s-1}$ are the *internal gates*, and f_{n+s} is the *output gate.* The circuit C can naturally be thought of as a function as well: on an input string $x = (x_1, \ldots, x_n) \in \{0,1\}^n$, $C(x)$ denotes $f_{n+s}(x)$.

Thinking of the connections between the gates as a directed acyclic graph in the natural way, with the input gates as n source nodes $1, \ldots, n$, and the jth gate with indices $i_1, \ldots, i_k < j$ as a node j with incoming arcs from nodes $i_1, \ldots, i_k$, the *depth* of C is the longest path from an input gate to the output gate. As a convention, gates with fan-in 1 are not counted in the depth measure. That is, gates of the form $g(x) = x$ or $g(x) = \neg x$ are not counted towards the length of a path from input to output.

Given a basis set $\mathcal{B}$ and $f : \{0,1\}^n \to \{0,1\}$, what is the minimal size s of a Boolean circuit over $\mathcal{B}$ with output gate $f_{n+s} = f$? This quantity is the *$\mathcal{B}$-circuit complexity of f*, and is denoted by $C_{\mathcal{B}}(f)$. The minimal depth of a circuit computing f is also of interest for parallel computing, and is denoted by $D_{\mathcal{B}}(f)$.

3. Circuits for Algorithms

The circuit model is excellent for understanding the difficulty and efficiency of computing *finite* functions. For every $f : \{0,1\}^n \to \{0,1\}$ and basis set, the circuit complexity of f is a fixed integer which could be high or low, relative to n.

Boolean circuits should be contrasted with the typical *uniform algorithm* models used in computability and complexity theory, based on finite objects such as Turing machines. In that setting, one is presented with functions (languages) defined over infinitely many strings, i.e., of the form

$$L : \{0,1\}^\star \to \{0,1\}, \tag{1}$$

and a primary goal is to find a fixed program or machine M such that, for every input $x \in \{0,1\}^\star$, running M on input x always produces the output $L(x)$ in some finite (or efficient) number of steps. This sort of computational model can trivially compute all finite functions (outputting 1 on only finitely many inputs) in *constant* time, by hard-coding the answers to each of the finitely many inputs in the program's code.

There is a logical way to extend the Boolean circuit model to also compute functions of type (1): we simply provide infinitely many circuits.

Definition 3.1. Let $s : \mathbb{N} \to \mathbb{N}$, $d : \mathbb{N} \to \mathbb{N}$, and $L : \{0,1\}^\star \to \{0,1\}$. *$L$ has size-$s(n)$ depth-$d(n)$ circuits* if there is an infinite family $\{C_n \mid n \in \mathbb{N}\}$ of Boolean circuits over B_2 such that, for every n, C_n has n inputs, size at most $s(n)$, depth at most $d(n)$, and for all $x \in \{0,1\}^n$, $C_n(x) = L(x)$.

This is an *infinite* (so-called *non-uniform*) computational model: for each input length n, there is a different "program" C_n for computing the 2^n inputs of that length, and the size of this program can grow with n.

Note that *every* language L has circuits of size $O(n2^n)$, following the observation that every $f : \{0,1\}^n \to \{0,1\}$ is specified by a 2^n-bit vector, called the *truth table* of f. This construction can be improved to $2^n/n + o(2^n/n)$ size [Sha49, Lup59], and a simple counting argument shows that this improved size bound is tight for general functions. The class of functions of type (1) computable with "feasibly-sized" circuits is often called P/poly:

Definition 3.2. Let $s : \mathbb{N} \to \mathbb{N}$ and $L : \{0,1\}^\star \to \{0,1\}$. Define $\mathsf{SIZE}(s(n))$ to be the class of functions L such that L has size-$s(n)$ circuits, and P/poly to be the class of functions L such that there is a $k \geq 1$ satisfying $L \in \mathsf{SIZE}(n^k + k)$.

Studying P/poly requires us to contemplate explicit trade-offs between the *sizes* of programs for computing functions and the *sizes* of inputs to those programs. Proving that a language is *not* in P/poly is a very strong result, implying that even finite segments of the language require "large" computations, relative to the sizes of inputs in the segment. From such results one can, in principle, derive concrete numerical statements about the limits of solving a problem. A proof that $L \notin \mathsf{P}/\text{poly}$ could potentially be used to establish that solving L on 1000-bit inputs requires 10^{100} size computations. This would be a true claim concerning the intractability of L in the known physical universe.[3]

Immediately one wonders how the two computational models of algorithms and circuits relate. The basic *Circuits for Algorithms* question is:

What "normal" algorithms (efficient or not) can be simulated in P/poly?

More precisely, take a complexity class $\mathcal{C}$ defined with respect to the usual uniform algorithm model (P, NP, PSPACE, EXP, NEXP, and so on). Which of these classes are contained in P/poly? For example, if EXP were contained in P/poly, then all uniform algorithms running in exponential time can be simulated by polynomial-size computations in the non-uniform circuit model. It is believed that in general, circuit families cannot really solve NP-hard problems significantly more efficiently than algorithms can, and that $\mathsf{NP} \not\subset \mathsf{P}/\text{poly}$. Complexity theory is *very* far from proving this; for one, it would imply $\mathsf{P} \neq \mathsf{NP}$.

To gain a little insight into the difficulty, we may first ask if P/poly is contained in any of the above classes. The answer to that question is no. Let $\{M_1, M_2, M_3, \ldots\}$ be a computable enumeration of Turing machines. Consider the function $L(x)$ defined to output 1 if and only if $M_{|x|}$ halts on $1^{|x|}$. For every n, either L outputs 1 on all n-bit strings, or L outputs 0 on all such strings. It is easy to infer from this that $L \in \mathsf{P}/\text{poly}$. However, L is also *undecidable*, as there is an easy reduction from the Halting Problem to L. The class P/poly, defined in terms of an infinite computational model, has unexpected power.

In general, the tools of computability theory are essentially powerless for understanding P/poly, and complexity theory has not yet discovered enough new tools. Indeed, this provides another reason to study circuit complexity: we're forced to develop new lower bound proof methods that go beyond old methods like diagonalization, which is known not to be sufficient by itself due to the Relativization barrier [BGS75]. These new methods may be useful in the long run for resolving other problems such as P vs NP. While nontrivial results are known (which we now survey), they are meager in comparison to what is conjectured.

[3]In fact, statements of this form have been extracted from circuit complexity lower bounds. See Stockmeyer-Meyer [SM02].

3.1. Classes with efficient circuits.

It is relatively easy to see that $\mathsf{P} \subset \mathsf{P}/\text{poly}$: polynomial-time algorithms can be "unrolled" for polynomially many steps, and simulated step-by-step using polynomial-size circuits. Furthermore, randomized polynomial-time algorithms have polynomial-size circuit families, i.e., $\mathsf{BPP} \subset \mathsf{P}/\text{poly}$ [Adl78], by judiciously hard-coding good random seeds in polynomial-size circuits.

Besides what we have already sketched, there are few other nontrivial results known. Kolmogorov made an intriguing conjecture:

Conjecture 3.3 (A. N. Kolmogorov, according to Levin [Lip94]). *For every $L \in \mathsf{P}$, there is a k such that L has kn size circuits.*[4]

The conjecture would be surprising, if true. For languages in P requiring $n^{100^{100}}$ time, it appears unlikely that the complexity of such problems would magically shrink to $O(n)$ size, merely because a different circuit can be designed for each input length. Kolmogorov's conjecture implies $\mathsf{P} \neq \mathsf{NP}$ [Lip94].

While it is generally believed that Conjecture 3.3 isn't true, a resolution looks very difficult. To see why, we sketch here the lack of progress on circuit lower bounds for languages in P. For a language $L : \{0,1\}^\star \to \{0,1\}$, define $L_n : \{0,1\}^n \to \{0,1\}$ to be *the n-bit restriction of L*: L_n agrees with L on all $x \in \{0,1\}^n$. The best known circuit lower bounds for functions in P are only small linear bounds:

Theorem 3.4 ([Blu84]). *There is an $L \in \mathsf{P}$ with $C_{B_2}(L_n) \geq 3n - o(n)$ for all n.*

Theorem 3.5 ([LR01, IM02]). *There is an $L \in \mathsf{P}$ with $C_{U_2}(L_n) \geq 5n - o(n)$ for all n.*

Hence it is possible that every $L \in \mathsf{P}$ has circuits of size $5.1n$. Even if the L is allowed to be in NP, no better circuit lower bounds are known. It is open whether every $L \in \mathsf{TIME}[2^{O(n)}]^{\mathsf{NP}}$ (functions in $2^{O(n)}$ time with access to an NP oracle) has $5.1n$ size circuits. In Section 5 we will see a possible approach to this question.

It was recently shown that, if Kolmogorov's conjecture is true, then such $O(n)$-size circuits must be intractable to construct algorithmically [SW13].[5]

3.2. Classes without efficient circuits.

Let us now survey which functions *are* known to not be in P/poly.

Ehrenfeucht [Ehr75] studied the decision problem for sentences in the first order theory of $\mathbb{N}$ with addition, multiplication, and exponentiation, where all quantified variables are bounded by constants. (The problem is clearly decidable since all variables are bounded.) He showed that this problem requires $(1+\delta)^n$-size circuits for some $\delta > 0$, assuming a reasonable encoding of sentences as binary strings. Meyer (1972, cf. [SM02]) and Sholomov [Sho75] proved that the same problem

[4]Apparently the conjecture was based on the affirmative answer by Kolmogorov and Arnol'd of Hilbert's 13th problem [Kol56, Arn57], which asks if every continuous function on three variables can be expressed as a composition of finitely many continuous functions on two variables.

[5]More formally, there is a language L computable in n^c time for some $c \geq 1$, such that for *every* $d \geq 1$ and every algorithm A running in n^d time, there are infinitely many n such that $A(1^n)$ does not output an $O(n)$ size circuit C_n computing L on n-bit inputs.

is decidable by a Turing machine using exponential ($2^{O(n)}$) space—in complexity notation, $\mathsf{EXPSPACE} \not\subset \mathsf{SIZE}((1+\delta)^n)$. This result can be scaled down to show the same circuit size lower bound for a language in $\Sigma_3\mathsf{EXP}$.[6]

Kannan [Kan82] proved that $\mathsf{NEXP}^{\mathsf{NP}} \not\subset \mathsf{P}/\text{poly}$. In fact his proof shows that $\mathsf{NEXP}^{\mathsf{NP}} \not\subset \mathsf{SIZE}(f(n))$, for every $f : \mathbb{N} \to \mathbb{N}$ satisfying $f(f(n)) \leq 2^n$ (these are the *half-exponential* functions). It is open whether $\mathsf{NEXP}^{\mathsf{NP}} \subset \mathsf{SIZE}(2^{\varepsilon n})$ for all $\varepsilon > 0$.

The P/poly lower bound of Kannan has been mildly improved over the years, to the presumably smaller (but still gigantic) complexity class MAEXP [BFT98]. However, it is open whether NEXP (or even $\mathsf{EXP}^{\mathsf{NP}}$) is contained in P/poly. It looks impossible that all problems verifiable in *exponential time* could be computed using only polynomial-size circuits, but the infinite nature of the circuit model has confounded all proof attempts. Section 5 outlines a new approach to this problem.

3.3. Restricted circuits.

There are several natural ways to restrict the circuit model beyond just circuit size, and still allow for complex circuit computations. In particular, restricting the depth leads to an array of possibilities.

Let A be the basis of *unbounded fan-in* AND and OR gates with NOT, i.e.,

$$A = \{\text{NOT}\} \cup \bigcup_{n\in\mathbb{N}} \{\text{OR}_n, \text{AND}_n\}.$$

For an integer $m \geq 2$, let M_m be the basis of unbounded fan-in MODm, AND, and OR gates with NOT:

$$M_m = \{\text{NOT}\} \cup \bigcup_{n\in\mathbb{N}} \{\text{OR}_n, \text{AND}_n, \text{MODm}_n\}.$$

Let T be the basis of unbounded fan-in MAJ gates with NOT:

$$T = \{\text{NOT}\} \cup \bigcup_{n\in\mathbb{N}} \{\text{MAJ}_n\}.$$

The following complexity classes are all subclasses of P/poly that have been widely studied. Let $k \geq 0$ be an integer.

- NCk: Languages computable with polynomial size, $O(\log^k n)$ depth circuits over the basis U_2.[7]
- ACk: Languages computable with a polynomial size and $O(\log^k n)$ depth circuit family $\{C_n\}$ over A. That is, there is a fixed integer $d \geq 1$ such that every C_n has depth $d \log^k n$.[8]
- $\mathsf{ACk[m]}$: Languages computable with polynomial size, $O(\log^k n)$ depth circuits over M_m.

[6]$\Sigma_3\mathsf{EXP} = \mathsf{NEXP}^{\mathsf{NP}^{\mathsf{NP}}}$ is nondeterministic exponential time with oracle access to $\mathsf{NP}^{\mathsf{NP}}$ (and $\mathsf{NP}^{\mathsf{NP}}$ equals nondeterministic polynomial time with oracle access to NP). This class is contained in $\mathsf{EXPSPACE}$, and the containment is probably proper.

[7]The acronym NC stands for "Nick's Class," named after Nick Pippenger.

[8]AC stands for "Alternating Circuits," alternating between AND and OR. As a reminder, NOT gates are not counted in the depth bounds of AC, ACC, and TC circuits.

- ACCk: The union over all $m \geq 2$ of ACk[m].[9]
- TCk: Languages computable with polynomial size, $O(\log^k n)$ depth circuits over the basis T.[10]

A thorough survey of these classes cannot be provided here; instead, let us focus attention on the most relevant aspects for the present story. The most well-studied of these classes are AC0, ACC0, TC0, and NC1, and it is known that

$$\mathsf{AC0} \subsetneq \mathsf{AC0[p]} \subsetneq \mathsf{ACC0} \subseteq \mathsf{TC0} \subseteq \mathsf{NC1} \subseteq \mathsf{P}/\text{poly},$$

when p is a prime power.

NC1 is well-motivated in several ways: for instance, it is also the class of languages computable with infinite families of polynomial-size Boolean *formulas*, or circuits where all internal gates have outdegree one. For formulas, interesting lower bounds are known: the best known formula size lower bound for a function in P is $n^{3-o(1)}$ over U_2, by Håstad [Hås98]. TC0 is well-motivated from the study of neural networks: the MAJ function is a primitive model of a neuron, and the constant depth criterion reflects the massive parallelism of the human brain. Less primitive models of the neuron, such as *linear threshold functions*, end up defining the same class TC0. (A linear threshold function is a Boolean function f defined by a linear form $\sum_{i=1}^{n} w_i x_i$ for some $w_i \in \mathbb{Z}$, and a threshold value $t \in \mathbb{Z}$. For all $(x_1, \ldots, x_n) \in \{0,1\}^n$, $f(x_1, \ldots, x_n) = 1$ if and only if $\sum_i w_i x_i \geq t$.)

The MODm operations may look strange, but they arose naturally out of a specific program to develop circuit complexity in a "bottom up" way, starting with very restricted circuits and a hope of gradually relaxing the restrictions over time. First, AC0 was studied as a "maximally parallel" but still non-trivial class, and it was shown that MOD2 $\notin$ AC0 [Ajt83, FSS81]. This made it reasonable to ask what is computable when the MOD2 function is provided among the basis functions in AC0, leading to the definition of AC0[2]. Then it was proved that for distinct primes p and q, MODq $\notin$ AC0[q] [Raz87, Smo87], hence MOD3 $\notin$ AC0[2]. One then wonders what is computable when MOD3 and MOD2 are both allowed in the basis. It is not hard to see that including MOD6 in the basis functions is equivalent to including MOD3 and MOD2. Attention turned to AC0[6]. (There were many other separate threads of research, such as lower bounds on fixed-depth versions of TC0 [HMP$^+$93], which space prevents us from covering here.)

At this point, the trail was lost. It is still open whether every language in P/poly (and in EXP) has depth-three circuit families over M_6. It has been shown only recently that NEXP is not contained in ACC0, via a generic connection between algorithms-for-circuits and circuits-for-algorithms [Wil10, Wil11] (see Section 5). Yet it is open whether NEXP is contained in TC0, even for TC0 circuits of depth *three*.

[9] ACC stands for "Alternating Circuits with Counting."
[10] TC stands for "Threshold Circuits."

4. Algorithms For Circuits

In the most common form of circuit analysis problem, one takes a circuit as input, and decides a property of the function computed by the circuit. Let a property P be a function from the set of all Boolean functions $\{f : \{0,1\}^n \to \{0,1\} \mid n \geq 0\}$ to the set $\{0,1\}$.

Generic Circuit Analysis
Input: A logical circuit C
Output: A property $P(f)$ of the function f computed by C

The canonical example of such a problem is the Circuit Satisfiability problem (a.k.a. Circuit-SAT), which we shall survey in detail.

Circuit-SAT
Input: A logical circuit C
Output: Does the function f computed by C output 1 on some input?

This is basically equivalent to checking if C implements a trivial function that is constant on all inputs—a function of *minimum* circuit complexity. Hence the Circuit-SAT problem may viewed as providing nontrivial insight into the circuit complexity of the function implemented by a given circuit.

As Circuit-SAT is NP-complete, it is unlikely that there is an polynomial-time algorithm for it. An algorithm which exhaustively searches over all possible inputs to C requires $\Omega(2^n \cdot |C|)$ time steps, where n is the number of inputs to C, and $|C|$ is the size of the circuit. Is there a *slightly* faster algorithm, running in (for example) $1.99^n \cdot |C|^2$ time? Presently, there is no known algorithm for solving the problem on generic circuits of size s and n inputs that is asymptotically faster than the time cost of exhaustive search. Fine-grained questions of this variety are basic to two emerging areas of research: parameterized algorithms [DF99, FG06] and exact algorithms [FK10]. For many NP-hard problems, asymptotically faster algorithms over exhaustive search do exist, and researchers actively study the extent to which exhaustive search can be beaten. (We shall see in Section 5 that even slightly faster Circuit-SAT algorithms can sometimes have a major impact.)

4.1. Restrictions of Circuit-SAT. As seen in Section 3, many circuit restrictions have been studied; here we survey the known algorithms for the satisfiability problem under these different restrictions. In this section, we think of AC0, ACC, TC0, NC1, and P/poly not as classes of languages, but as *classes of circuit families*: collections of infinite circuit families satisfying the appropriate restrictions. For each class $\mathcal{C}$, a satisfiability problem can be defined:

$\mathcal{C}$-SAT
Input: A circuit C from a family in class $\mathcal{C}$
Output: Is there an input on which C evaluates to true?

Just as with general Circuit-SAT, the $\mathcal{C}$-SAT problem remains NP-complete even for AC0-SAT [Coo71], yet for simple enough $\mathcal{C}$, $\mathcal{C}$-SAT algorithms running faster than exhaustive search *are* known.

k-SAT. The k-SAT problem is to determine satisfiability of a very simple circuit type: an AND of ORs of k literals (which can be input variables and/or their negations). This is also called *conjunctive normal form* (CNF). Without loss of generality, the AND gate may be assumed to have $O(n^k)$ fan-in, as there are only $O(n^k)$ possible ORs of k literals. The k-SAT problem is also NP-complete [Coo71] for all $k \geq 3$. Nevertheless, 3-SAT can be solved in 1.331^n time using a deterministic algorithm [MTY11], or 1.308^n time [Her11] using a randomized algorithm. These running times form the tail end of a long line of published algorithms, with each subsequent algorithm decreasing the base of the exponent by a little bit. (See the survey of Dantsin and Hirsch [DH09].)

How much faster can 3-SAT be solved? The *Exponential Time Hypothesis* of Impagliazzo and Paturi [IP01] asserts that this line of work must "converge" to some base of exponent greater than 1:

Exponential Time Hypothesis (ETH): There is a $\delta > 0$ such that 3-SAT on n variables cannot be solved in $O((1+\delta)^n)$ time.

Impagliazzo, Paturi, and Zane [IPZ01] showed that ETH is not just a hypothesis about one NP-complete problem: by using clever *subexponential time reductions*, ETH implies that many other NP-hard problems require $(1+\delta)^n$ time to solve for some $\delta > 0$. Many other consequences of ETH have been found [LMS11].

The k-SAT problem for arbitrary k has also been extensively studied. The best known k-SAT algorithms all run in $2^{n-n/(ck)}$ time, for a fixed constant c [PPZ97, Sch02, PPSZ98, DH09]. So for $k > 3$, the savings in running time over 2^n slowly disappears as k increases. The *Strong Exponential Time Hypothesis* [IP01, CIP09] asserts that this phenomenon is inherent in all SAT algorithms:

Strong Exponential Time Hypothesis (SETH): For every $\delta < 1$ there is a k such that k-SAT on n variables cannot be solved in $2^{\delta n}$ time.

For example, SETH implies that even $2^{.99999n}$ is not enough time for solving k-SAT over all constants k. (It is known that SETH implies ETH.)

AC0-SAT There has been less work on this problem, but recent years have seen progress [CIP09, BIS12, IMP12]. The fastest known AC0-SAT algorithm is that of Impagliazzo, Matthews, and Paturi [IMP12], who give an $O(2^{n-\Omega(n/(\log s)^{d-1})})$ time algorithm on circuits with n inputs, s gates, and depth d.

ACC0-SAT The author [Wil11] gave an algorithm running in $O(2^{n-n^{\varepsilon}})$ time for $\mathsf{ACC0}$ circuits of $2^{n^{\varepsilon}}$ size, for some $\varepsilon \in (0,1)$ which is a function of the depth d of the given circuit and the modulus m used in the MODm gates. This algorithm was recently extended to handle the larger circuit class $\mathsf{ACC0} \circ \mathsf{THR}$, which is $\mathsf{ACC0}$ augmented with an additional layer of arbitrary linear threshold gates near the inputs [Wil14].

TC0-SAT For depth-two $\mathsf{TC0}$ circuits, Impagliazzo, Paturi, and Schneider [IPS13] showed that satisfiability with n inputs and cn wires (i.e., edges) can be determined in $2^{\delta n}$ time for some $\delta < 1$ that depends on c. No nontrivial algorithms are known for satisfiability of depth-three $\mathsf{TC0}$ (and circuit lower bounds aren't known, either).

Formula-SAT Santhanam [San10] proved that satisfiability of cn size formulas over U_2 can be determined in $2^{\delta n}$, for some $\delta < 1$ depending on c. His algorithm was extended to the basis B_2 by Seto and Tamaki [ST12], and to larger size formulas over U_2 by Chen et al. [CKK+14]. Applying recent concentration results of Komargodski, Raz and Tal [KRT13], the algorithm of Chen et al. can solve SAT for formulas over U_2 of size $n^{3-o(1)}$ in randomized $2^{n-n^{\Omega(1)}}$ time with zero error. (Recall that the best known formula *lower bound* is $n^{3-o(1)}$ size as well; these Formula-SAT algorithms exploit similar ideas as in the lower bound methods.)

4.2. Approximate Circuit Analysis.

A different form of circuit analysis is that of *additive approximate counting*; that is, approximating the *fraction of satisfying assignments* to a given circuit:

Circuit Approximation Probability Problem (CAPP)
Input: A circuit C
Output: The quantity $\Pr_x[C(x) = 1]$, to within $\pm$ 1/10.

The constant 1/10 is somewhat arbitrary, and could be any constant in $(0, 1/2)$ (usually this constant is a parameter in the algorithm). As with $\mathcal{C}$-SAT, the problem $\mathcal{C}$-CAPP can be defined for any circuit class $\mathcal{C}$. Approximate counting has been extensively studied due to its connections to *derandomization.* CAPP is easily computable with *randomness* by sampling (for instance) 100 x's uniformly at randomn, and evaluating C on them. We want to know purely *deterministic* algorithms. The structure of this subsection will parallel that of the coverage of $\mathcal{C}$-SAT. We cannot hope to cover all work in this article, and can only provide highlights.[11]

Several algorithms we shall mention give a stronger property than just approximately counting. Prior to viewing the circuit, these algorithms efficiently construct a small collection $\mathcal{A}$ of strings (assignments), such that for all circuits C of the appropriate size and depth from a circuit class $\mathcal{C}$, the fraction of satisfying

[11]We should also note that many algorithms from the previous subsection not only solve $\mathcal{C}$-SAT, but can *exactly* count the number of satisfying assignments (or can be modified to do so), implying a $\mathcal{C}$-CAPP algorithm.

assignments of C over $\mathcal{A}$ is a close approximation to the total fraction of satisfying assignments of C. Such algorithms are called *pseudorandom generators* and are inherently tied to lower bounds. Indeed, lower bounds against a circuit class $\mathcal{C}$ are generally a prerequisite for pseudorandom generators for $\mathcal{C}$, because the efficient process which produces such a collection $\mathcal{A}$ cannot be modeled within $\mathcal{C}$.

The case of depth-two $\mathsf{AC0}$ (i.e., of an AND of ORs of literals, or an OR of AND of literals) is especially interesting. Luby and Velickovic [LV96] showed that this case of CAPP is computable in $n^{\exp(O(\sqrt{\log\log n}))}$ time. Gopalan, Meka, and Reingold [GMR13] improved this to about $n^{O(\log\log n)}$ time. It appears that here, a deterministic polynomial-time algorithm for CAPP may be within reach.

Ajtai and Wigderson [AW85] showed that $\mathsf{AC0}$-CAPP is solvable in $2^{n^{\varepsilon}}$ time for every $\varepsilon > 0$, providing a pseudorandom generator. A pseudorandom generator of Nisan [Nis91] yields an $\mathsf{AC0}$-CAPP algorithm running in $n^{\log^{O(d)} s}$ time, where s is the size and d is the depth. There has been much work since then; most recently, Trevisan and Xue [TX13] construct tighter pseudorandom generators for $\mathsf{AC0}$, showing that $\mathsf{AC0}$-CAPP can be computed in $n^{\tilde{O}(\log^{d-1} s)}$ time.

For the class $\mathsf{ACC0}$, *exact* counting of satisfying assignments can be done in about the same (best known) running time as computing satisfiability [Wil14].

To our knowledge, no nontrivial CAPP algorithm for depth-two $\mathsf{TC0}$ circuits is known. However, here is a good place to mention two other threads of work relating to low-depth circuits. The problem of approximately counting the number of zeroes in $\{0,1\}^n$ of a low-degree polynomial over a finite field is equivalent to computing CAPP on a MODp of AND gates of fan-in d. This problem can be solved essentially optimally for fixed d, in deterministic time $O_d(n^d)$ [LVW93, BV10, Lov09, Vio09]. A *polynomial threshold function of degree d* (PTF) has the form $f : \{-1,1\}^n \to \{-1,1\}$ and is representable by the sign of a multivariate degree-d polynomial over the integers. (Such functions can be construed as Boolean; the convention is that $-1, 1$ correspond to *true* and *false*, respectively.) Approximating the number of zeroes to a degree-d PTF can be modeled by solving CAPP on a *linear threshold gate* of MOD2 gates of fan-in d. It is known that for every fixed d, approximate counting for degree-d PTFs can be done in polynomial time [MZ13].

For Boolean formulas, Impagliazzo, Meka, Zuckerman [IMZ12] give a pseudorandom generator yielding a $2^{s^{1/3+o(1)}}$ time algorithm for Formula-CAPP on size-s formulas over U_2. For formulas of size s over B_2 and branching programs of size s, their generator can be used to approximately count in $2^{s^{1/2+o(1)}}$ time.

No nontrivial results for CAPP are known for unrestricted Boolean circuits.

4.3. Truth Table Analysis. So far, we have only considered circuit analysis problems where the input to be analyzed is a circuit. Another class of circuit analysis problems take a *Boolean function on n variables* as input, specified as a 2^n-bit string, and the goal is to compute some property of "good" circuits which compute the function f.

Generic Truth Table Analysis
Input: A function $f : \{0,1\}^n \to \{0,1\}$
Output: Property $P(f)$ of circuits computing f

A natural example is that of minimizing a circuit given its truth table:

Circuit-Min [Yab59, KC00]
Input: A function $f : \{0,1\}^n \to \{0,1\}$ and $k \in \mathbb{Z}^+$
Output: Is $C_{B_2}(f) \leq k$?

In other words, we want to decide if the circuit complexity of f is at most k. As with Circuit-SAT and CAPP, we can also define the $\mathcal{C}$-Min problem for restricted circuit classes $\mathcal{C}$. The problem is easily seen to be in NP. It is strongly believed that Circuit-Min is intractable: if it were in P, then there would be no *pseudorandom functions*, contradicting conventional wisdom in cryptography. Informally, a pseudorandom function is a function f implementable with polynomial-size circuits that "behaves like" a random function, to all efficient processes with input/output access to f. Since a random function g has high circuit complexity with high probability, and f has low circuit complexity, an efficient algorithm for Circuit-Min could be used to tell f and g apart with non-negligible success probability, after querying them at $n^{O(1)}$ points. As a result, restricted versions of Circuit-Min such as $\mathsf{NC1}$-Min and $\mathsf{TC0}$-Min are also intractable under cryptographic assumptions, as those classes are believed to support such functions.[12]

Perhaps Circuit-Min is NP-hard. Proving that is a difficult open problem. To obtain a polynomial-time reduction from (say) 3-SAT to Circuit-Min, unsatisfiable formulas have to be efficiently mapped into functions without small circuits; however, recall that we do not *know* explicit functions with high circuit complexity. Kabanets and Cai [KC00] show that if the NP-hardness of Circuit-Min could be proved under a natural notion of reduction, then long-open circuit lower bounds like $\mathsf{EXP} \not\subset \mathsf{P}/\text{poly}$ would follow.

One version of Circuit-Min is known to be NP-complete: *DNF-Min*, the problem of minimizing a DNF formula (an OR of ANDs of literals) given its truth table [Mas79, AHM$^+$08]. (Intuitively, DNF-Min can be proved hard because strong lower bounds are known for computing Boolean functions with DNFs.) However, one can efficiently find an *approximately* minimum-sized DNF [AHM$^+$08].

A newly-introduced and related analysis problem is that of *compression*:

[12]Here is a good point to briefly mention a connection between Circuit-Min and complexity barriers. Razborov and Rudich [RR97] showed that practically all known circuit lower bound proof techniques (i.e., proving there are no efficient circuits-for-algorithms) yield weak *efficient* algorithms for Circuit-Min, weak enough to break any candidate pseudorandom function. Hence it's likely that such "natural proofs" cannot prove even $\mathsf{TC0}$ lower bounds. In summary, every "natural proof" that there are no efficient circuits for some algorithms also yields an interesting algorithm for efficient circuits!

Compression of $\mathcal{C}$ [CKK+14]
Input: A function $f : \{0,1\}^n \to \{0,1\}$ computable with a circuit from $\mathcal{C}$
Output: A (possibly unrestricted) circuit C computing f with size $\ll 2^n/n$

Chen et al. [CKK+14] show that the techniques used in existing circuit lower bound proofs can be "mined" to obtain somewhat efficient compression algorithms for $\mathsf{AC0}$, small Boolean formulas, and small branching programs. They pose as an open problem whether $\mathsf{ACC0}$ admits such a compression algorithm.

Learning circuits There is one more important form of circuit analysis that can be viewed as restricted access to the truth table of a function: that of *learning* a function $f : \{0,1\}^n \to \{0,1\}$ which is initially hidden, but is known or assumed to be implementable in some restricted circuit class $\mathcal{C}$. In this survey we focus on the problem of *exact learning of $\mathcal{C}$ with membership and equivalence queries* [Ang87], where a learning algorithm does not see f in its entirety, but has the ability to:

- query f on an arbitrary $x \in \{0,1\}^n$ (a *membership query*), and
- pose a hypothesis circuit H on n bits, asking if H and f compute the same function (an *equivalence query*). If $H \neq f$, the algorithm is provided with a counterexample point x on which $H(x) \neq f(x)$.

Pseudorandom functions, mentioned earlier, naturally connect with learning. A pseudorandom function has small circuits yet "looks like a random function" when it is queried a small number of times—this kind of function is naturally difficult to learn. Hence learning of Boolean functions computable in $\mathsf{TC0}$ and $\mathsf{NC1}$ is believed to be intractable. Other examples can be found in the references [Val84, KV94].

5. Connections

In the *Circuits for Algorithms* space, one designs simple circuits to simulate complex algorithms, or proves that no simple circuits exist for this task. In *Algorithms for Circuits*, the goal is to design faster circuit-analysis algorithms. It is reasonable to hypothesize that these tasks may inform each other. A provably nontrivial algorithm for analyzing all circuits from a class should exhibit, at its core, nontrivial understanding about the limitations of that circuit class. Conversely, if a simple function cannot be computed by small circuits, then algorithms may be able to use this function to analyze small circuits faster than exhaustive search.

For restricted classes of circuits, one can sometimes adapt known techniques for proving lower bounds to derive faster SAT algorithms (or CAPP algorithms) for those circuits. For instance, the progress on Formula-SAT algorithms and on pseudorandom generators for Boolean formulas, both mentioned in Section 4, came out of tighter analyses of the *random restriction* method originally used for proving formula lower bounds [Sub61, Hås98].

In the following, we restrict attention to more generic connections (i.e., formal implications) between efficient circuit-analysis algorithms and circuit lower bounds.

5.1. Circuit lower bounds and derandomization/CAPP. Perhaps the earliest explicit study of how algorithms and lower bounds connect can be found in the formal theory of cryptographic pseudorandomness, initiated by Blum and Micali [BM84] and Yao [Yao82]. The existence of cryptographic pseudorandom generators were shown to imply subexponential time deterministic simulations of randomized polynomial time algorithms. Nisan and Wigderson [NW94] defined a relaxed notion of pseudorandom generator explicitly for the purposes of derandomizing randomized algorithms (instead of for cryptography) and proved connections between circuit lower bounds and the existence of pseudorandom generators. Subsequent work [BFNW93, IW97, KvM02, IKW02] improved these connections. These papers give an effective *equivalence* between (for example) functions in $2^{O(n)}$ time requiring "high" circuit complexity, and the existence of pseudorandom generators computable in $2^{O(n)}$ time that are effective against "low complexity" circuits.

For an example, Babai et al. [BFNW93] showed that $\mathsf{EXP} \not\subset \mathsf{P}/\text{poly}$ implies that randomized polynomial-time algorithms can be simulated deterministically in subexponential time, on infinitely many input lengths. Formally speaking:

Theorem 5.1 ([BFNW93]). $\mathsf{EXP} \not\subset \mathsf{P}/poly$ *implies* $\mathsf{BPP} \subseteq \mathsf{ioSUBEXP}$.

This connection was sharpened by Impagliazzo and Wigderson:

Theorem 5.2 ([IW97]). *If there is a $\delta > 0$ and a function computable in $2^{O(n)}$ time requiring circuits of size at least $(1+\delta)^n$ for almost all input lengths n, then* $\mathsf{P} = \mathsf{BPP}$.

That is, from exponential-size lower bounds, one can simulate every randomized polynomial-time algorithm in deterministic polynomial time. Impagliazzo, Kabanets, and Wigderson [IKW02] showed that even a seemingly weak lower bound like $\mathsf{NEXP} \not\subset \mathsf{P}/\text{poly}$ would imply a derandomization result: namely, there is a simulation of Merlin-Arthur games (a probabilistic version of NP) computable in nondeterministic subexponential time. In the opposite direction, they showed how a subexponential time algorithm for CAPP implies lower bounds:

Theorem 5.3 ([IKW02]). *If CAPP can be computed in $2^{n^{o(1)}}$ time for all circuits of size n, then* $\mathsf{NEXP} \not\subset \mathsf{P}/poly$.

Recall the best known algorithm for CAPP is exhaustive search, taking $\Omega(2^n)$ time; an improvement to $2^{n^{\varepsilon}}$ for every $\varepsilon > 0$ would be an incredible achievement. However, the hypothesis of Theorem 5.3 can be weakened significantly: essentially any nontrivial improvement over 2^n time for CAPP implies the lower bound.

Theorem 5.4 ([Wil10]). *Suppose for every k, CAPP on circuits of size n^k and n inputs can be computed in $O(2^n/n^k)$ time. Then* $\mathsf{NEXP} \not\subset \mathsf{P}/poly$.

Furthermore, computing CAPP for a restricted circuit class $\mathcal{C}$ faster than exhaustive search would imply that $\mathsf{NEXP} \not\subset \mathcal{C}$ [Wil10, SW13]. Theorem 5.4 requires that $\mathcal{C}$ satisfy certain closure properties (all classes covered in this survey satisfy them). Ben-Sasson and Viola [BSV14] have recently sharpened the connection between CAPP algorithms and circuit lower bounds, by carefully modifying a known construction of probabilistically checkable proofs.

5.2. Circuit lower bounds from SAT algorithms.

We now survey the impact of Circuit-SAT algorithms on the topic of Circuits for Algorithms. First, if we have "perfect" circuit analysis, i.e., Circuit-SAT is solvable in *polynomial time*, then there is a function in EXP that does not have small circuits. This result is quite old in complexity-theory years:

Theorem 5.5 (Meyer [KL82]). *If* $\mathsf{P} = \mathsf{NP}$ *then* $\mathsf{EXP} \not\subset \mathsf{P}/poly$.

This is an interesting implication, but it may be of limited utility since we do not believe the hypothesis. Nevertheless, Theorem 5.5 is a good starting point for thinking about how circuit analysis can relate to circuit lower bounds. A proof can be quickly sketched: assuming $\mathsf{P} = \mathsf{NP}$, we obtain many other equalities between complexity classes, including $\mathsf{NP}^{\mathsf{NP}^{\mathsf{NP}}} = \mathsf{P}$ and $\Sigma_3\mathsf{EXP} = \mathsf{NEXP}^{\mathsf{NP}^{\mathsf{NP}}} = \mathsf{EXP}$. As stated in Section 3, $\Sigma_3\mathsf{EXP}$ contains a language requiring circuits of maximum complexity (by directly "diagonalizing" against all circuits up to the maximum size). Therefore EXP now contains such a language as well.

This simple argument shows how a feasibility hypothesis like $\mathsf{P} = \mathsf{NP}$ implies a reduction in the algorithmic complexity of hard functions. It is tantalizing to wonder if a lower bound could proved by contradiction, in this way: from a feasibility hypothesis, deduce that the complexity of another *provably hard* function reduces so drastically that it becomes contradictorily *easy*. Sure enough, recent progress by the author on $\mathsf{ACC0}$ lower bounds (described below) takes this approach.

Studying the proof more carefully, Theorem 5.5 can be improved in a few ways. Considering the contrapositive of the proof sketch, we find that if every function in $2^{O(n)}$ time has less than the *maximum possible* circuit complexity $(1 + o(1))2^n/n$, then $\mathsf{P} \neq \mathsf{NP}$. In other words, if non-uniform circuits can gain even a small advantage over exponential-time algorithms in simulation, then $\mathsf{P} \neq \mathsf{NP}$ would follow. Another improvement of Theorem 5.5 comes from observing we do not exactly need *polynomial time* Circuit-SAT algorithms: weaker guarantees such as $n^{(\log n)^k}$ time would suffice to conclude $\mathsf{EXP} \not\subset \mathsf{P}/\text{poly}$. Assuming ETH, this sort of running time is still beyond what is expected.

Combining these results with our earlier remarks on derandomization, we see that either EXP doesn't have large circuits and hence $\mathsf{P} \neq \mathsf{NP}$, or EXP requires large circuits and every randomized algorithm would have an interesting deterministic simulation, by Theorem 5.2. No matter how EXP vs P/poly is resolved, the consequences will be very interesting.

Modern times Theorem 5.5 and its offshoots only work for Circuit-SAT algorithms running in subexponential time. An indication that techniques for weak SAT algorithms may still be useful for circuit lower bounds appears in the work of Paturi, Pudlak, and Zane [PPZ97]. They gave a structure lemma on k-SAT instances, and applied it to prove not only that k-SAT has an $2^{n-n/k}$ time algorithm, but also lower bounds for depth-three $\mathsf{AC0}$ circuits.

In recent years, the author showed that very weak improvements over exhaustive search for $\mathcal{C}$-SAT would imply circuit lower bounds for NEXP:

Theorem 5.6 ([Wil10, Wil11]). *There is a $c > 0$ such that, if $\mathcal{C}$-SAT can be solved on circuits with n inputs and n^k size in $O(2^n/n^c)$ time for every k, then* $\mathsf{NEXP} \not\subset \mathcal{C}$.

While the conclusion is weaker than Theorem 5.5, the hypothesis (for all classes $\mathcal{C}$ we have considered) is *extremely* weak compared to $\mathsf{P} = \mathsf{NP}$; indeed, it even looks plausible. The above theorem was combined with the ACC0-SAT algorithm mentioned in Section 4.1 to conclude:

Theorem 5.7 ([Wil11]). $\mathsf{NEXP} \not\subset \mathsf{ACC0}$.

Since Theorem 5.7 was proved, it has been concretely extended twice. The first extension slightly lowers the complexity of NEXP, down to complexity classes such as $\mathsf{NEXP}/1 \cap \mathsf{coNEXP}/1$ [Wil13]. (In fact a generic connection is proved between $\mathcal{C}$-SAT algorithms and $\mathcal{C}$ circuit lower bounds for $\mathsf{NEXP}/1 \cap \mathsf{coNEXP}/1$, with a slightly stronger hypothesis: we have to assume SAT algorithms for $n^{\log^k n}$ size circuits.) The second extension strengthens ACC0 up to the class $\mathsf{ACC0} \circ \mathsf{THR}$, or ACC0 circuits augmented with a layer of linear threshold gates near the inputs [Wil14].

Theorem 5.6 holds for all circuit classes $\mathcal{C}$ of Section 2, but one may need (for example) a SAT algorithm for $2d$-depth circuits to obtain a d-depth circuit lower bound. The project of tightening parameters to make $\mathcal{C}$-SAT algorithms directly correspond to the same $\mathcal{C}$ circuit lower bounds has seen much progress [SW13, JMV13, Oli13, BSV14]. Now (for example) it is known that SAT algorithms for depth $d + 1$ or $d + 2$ (depending on the gate basis) imply depth-d lower bounds.

Perhaps Circuit-SAT looks too daunting to improve upon. Are there other connections between SAT algorithms and circuit lower bounds? Yes. From faster 3-SAT algorithms, superlinear size lower bounds follow:

Theorem 5.8 ([Wil10]). *Suppose the Exponential Time Hypothesis (ETH) is false: that is, 3-SAT is in $2^{\varepsilon n}$ time for every $\varepsilon > 0$. Then there is a language $L \in \mathsf{TIME}[2^{O(n)}]^{\mathsf{NP}}$ such that, for every $c \geq 1$, L does not have cn-size circuits.*

ETH was discussed in Section 4.1, and the conclusion of Theorem 5.8 was discussed as open in Section 3.1. Refuting the Strong Exponential Time Hypothesis (SETH) from Section 4.1 also implies (weaker) circuit lower bounds:

Theorem 5.9 ([JMV13]). *Suppose SETH is false: that is, there is a $\delta < 1$ such that k-SAT is in $O(2^{\delta n})$ time for all k. Then there is a language $L \in \mathsf{TIME}[2^{O(n)}]^{\mathsf{NP}}$ such that, for every $c \geq 1$, L does not have cn-size Valiant-series-parallel circuits.*

Intuition for the connections One intuition is that a faster circuit-analysis algorithm (say, for $\mathcal{C}$-SAT) demonstrates a specific *weakness* in representing computations with circuits from $\mathcal{C}$. A circuit family from $\mathcal{C}$ is *not* like a collection of black boxes which can easily hide satisfying inputs. (If we could only query the circuit as a black box, viewing only its input/output behavior, we could not solve $\mathcal{C}$-SAT in $o(2^n)$ time.) Another intuition is that the existence of a faster circuit-analysis algorithm for $\mathcal{C}$ demonstrates a *strength* of algorithms that run in less-than-2^n time: they can analyze nontrivial properties of a given circuit. Hence from assuming a less-than-2^n time $\mathcal{C}$-SAT algorithm, we should be capable of inferring that "less-than-2^n time algorithms are strong" and "$\mathcal{C}$-circuits are weak."

These observations hint at a proof that, assuming a $\mathcal{C}$-SAT algorithm, there is a language in NEXP without polynomial-size $\mathcal{C}$ circuits. The actual proof does not resemble these hints; it is a proof by contradiction. We assert that both a faster algorithm for analyzing $\mathcal{C}$ exists, and that $\mathsf{NEXP} \subset \mathcal{C}$. Together these two assumptions imply a too-good-to-be-true algorithm: a way to simulate every language solvable in nondeterministic $O(2^n)$ time with only $o(2^n)$ time. This simulation contradicts the *nondeterministic time hierarchy theorem* [Žák83], which implies that there are problems solvable in 2^n time nondeterministically which cannot be solved in $O(2^n/n)$ time nondeterministically. Informally, the faster nondeterministic simulation works by using $\mathsf{NEXP} \subset \mathcal{C}$ to nondeterministically guess $\mathcal{C}$ circuits that help perform an arbitrary $2^{O(n)}$ time computation, and using the faster circuit-analysis algorithm to verify that these $\mathcal{C}$ circuits do the job correctly.

5.3. Other connections.

Circuit lower bounds from learning. Intuitively, an efficient algorithm for learning circuits would have to harness some deep properties about the circuit class under consideration; perhaps these properties would also be enough to prove circuit lower bounds. Fortnow and Klivans proved a theorem modeling this intuition. Let $\mathcal{C}$ be a restricted circuit class, such as those defined in Section 3.3. In the following, say that $\mathcal{C}$ is *exactly learnable* if there is an algorithm for learning every hidden function from $\mathcal{C}$ using membership and equivalence queries (cf. Section 4.3).

Theorem 5.10 ([FK09]). *If all n-bit functions from $\mathcal{C}$ are exactly learnable in deterministic $2^{n^{o(1)}}$ time, then $\mathsf{EXP}^{\mathsf{NP}} \not\subset \mathcal{C}$.*

Theorem 5.11 ([FK09]). *If all n-bit functions from $\mathcal{C}$ are exactly learnable in randomized polynomial time, then randomized exponential time (BPEXP) is not contained in $\mathcal{C}$.*

Recently, these connections between learning circuits and circuit lower bounds have been somewhat strengthened:

Theorem 5.12 ([KKO13]). *If $\mathcal{C}$ is exactly learnable in $2^{n^{o(1)}}$ time, then there is a language in $\mathsf{TIME}[2^{n^{o(1)}}]$ that is not in $\mathcal{C}$.*

Theorem 5.13 ([KKO13]). *If $\mathcal{C}$ is exactly learnable in polynomial time, then there is a language in $\mathsf{TIME}[n^{\omega(1)}]$ that is not in $\mathcal{C}$.*

These proofs use a clever diagonalization argument, where the learning algorithm is used to construct an efficiently computable function f that plays the role of a *contrarian teacher* for the learning algorithm. When the learner asks a membership query x, f tells the learner *true* if f has not already committed to a value for x (otherwise, f reports $f(x)$). When an equivalence query is asked, f tells the learner "not equivalent" and outputs the first string y for which it has not already committed to an output value (thereby committing to a value for y). As f is constructed to never be equivalent to any hypothesis proposed by the learning algorithm, f cannot have circuits in $\mathcal{C}$.

Equivalences between circuit analysis and circuit lower bounds Earlier it was mentioned that there are rough equivalences between pseudorandom generators and circuit lower bounds. Pseudorandom generators can be viewed as "circuit analysis" algorithms, in the context of computing CAPP. Impagliazzo, Kabanets, and Wigderson [IKW02] proved an explicit equivalence:

Theorem 5.14 ([IKW02]). *$\mathsf{NEXP} \not\subset \mathsf{P}/poly$ if and only if for all $\varepsilon > 0$, CAPP is in $\mathsf{ioNTIME}[2^{n^\varepsilon}]/n^\varepsilon$.*

Without going into the notation, this theorem states that NEXP circuit lower bounds are equivalent to the existence of "non-trivial" subexponential time algorithms for CAPP. The author recently proved a related equivalence between the $\mathsf{NEXP} \not\subset \mathcal{C}$ problem (for various circuit classes $\mathcal{C}$) and circuit-analysis algorithms. Call an algorithm A *non-trivial for $\mathcal{C}$-Min* if

- $A(f)$ runs in $2^{O(n)}$ time on a given $f : \{0,1\}^n \to \{0,1\}$, and
- for all constants k and for infinitely many input lengths n, there is a $f : \{0,1\}^n \to \{0,1\}$ such that $A(f)$ outputs 1, and for all $f : \{0,1\}^n \to \{0,1\}$ computable with an $(n^k + k)$-size circuit from $\mathcal{C}$, $A(f)$ outputs 0.

That is, for infinitely many n, algorithm A outputs 1 on at least one Boolean function on n bits, and 0 on all functions with small circuit complexity.

Theorem 5.15 ([Wil13]). *$\mathsf{NEXP} \not\subset \mathcal{C}$ if and only if there is an algorithm A which is non-trivial for $\mathcal{C}$-Min.*

Connections in an algebraic setting In this survey, we considered Boolean functions and circuits computing them. However, connections between circuit-analysis algorithms and circuit lower bounds also hold in an *algebraic* framework, where Boolean functions are replaced by *polynomials over a ring R*, and Boolean circuits are replaced by *algebraic circuits*, which defined analogously to Boolean circuits, but we allow side constants from the ring as extra inputs to an algebraic circuit, and the gates are either *additions* or *multiplications* over the ring. Typically, R is taken to be a finite field, or $\mathbb{Z}$. Each algebraic circuit $C(x_1, \ldots, x_n)$ computes some polynomial $p(x_1, \ldots, x_n)$ over R.

The canonical circuit-analysis problem in this setting is:

Polynomial Identity Testing (PIT): Given an algebraic circuit C, does C compute the identically zero polynomial?

Using subtraction, it is easy to see this problem is equivalent to determining if two algebraic circuits C and C' compute the *same* polynomial.

It's natural to think of PIT as a type of satisfiability problem. However, PIT is probably *not* NP-hard: the problem is easily solvable in *randomized polynomial* *time* by substituting random elements (possibly over an extension field) [DL78, Zip79, Sch80]. A very interesting open problem is to determine whether randomness is necessary for efficiently solving PIT. Kabanets and Impagliazzo [KI04] proved that an efficient *deterministic* algorithm for PIT would imply *algebraic circuit lower bounds*: either $\mathsf{NEXP} \not\subset \mathsf{P}/\mathrm{poly}$, or the permanent of a matrix requires superpolynomial-size algebraic circuits.

6. Conclusion

This article has shown how a host of open problems in algorithms have direct bearing on some of the central problems in complexity theory. It is quite likely that there exist deeper interactions between Algorithms for Circuits and Circuits for Algorithms which await our discovery. Hopefully, the reader has been persuaded to think a little more about how algorithms and lower bounds relate to each other.

References

[Adl78] Leonard Adleman. Two theorems on random polynomial time. In *FOCS*, pages 75–83, 1978.

[AHM$^+$08] Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael Saks. Minimizing DNF formulas and AC0 circuits given a truth table. *SIAM J. Comput.*, 38(1):63–84, 2008.

[Ajt83] Miklos Ajtai. Σ^1_1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24:1–48, 1983.

[Ang87] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987.

[Arn57] V. I. Arnol'd. On functions of three variables. *Dokl. Akad. Nauk SSSR*, 114:679–681, 1957.

[AW85] Miklós Ajtai and Avi Wigderson. Deterministic simulation of probabilistic constant depth circuits (preliminary version). In *FOCS*, pages 11–19, 1985.

[AW09] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM TOCT*, 1, 2009.

[BFNW93] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

[BFT98] Harry Buhrman, Lance Fortnow, and Thomas Thierauf. Nonrelativizing separations. In *CCC*, pages 8–12, 1998.

[BGS75] Theodore Baker, John Gill, and Robert Solovay. Relativizations of the P =? NP question. *SIAM J. Comput.*, 4(4):431–442, 1975.

[BIS12] Paul Beame, Russell Impagliazzo, and Srikanth Srinivasan. Approximating AC0 by small height decision trees and a deterministic algorithm for # ac0sat. In *CCC*, pages 117–125, 2012.

[Blu84] Norbert Blum. A boolean function requiring 3n network size. *Theoretical Computer Science*, 28:337–345, 1984.

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequence of pseudo-random bits. *SIAM J. Comput.*, 13:850–864, 1984.

[BSV14] Eli Ben-Sasson and Emanuele Viola. Short pcps with projection queries. In *ICALP*, page to appear, 2014.

[BV10] Andrej Bogdanov and Emanuele Viola. Pseudorandom bits for polynomials. *SIAM J. Comput.*, 39(6):2464–2486, 2010.

[CIP09] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Parameterized and Exact Complexity (IWPEC)*, pages 75–85, 2009.

[CKK$^+$14] Ruiwen Chen, Valentine Kabanets, Antonina Kolokolova, Ronen Shaltiel, and David Zuckerman. Mining circuit lower bound proofs for meta-algorithms. In *CCC*, page to appear, 2014.

[Coo71] Stephen Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, 1971.

[DF99] Rodney G. Downey and Michael R. Fellows. Springer-Verlag, 1999.

[DH09] Evgeny Dantsin and Edward A. Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications*, volume 185, pages 403–424. IOS Press, 2009.

[DL78] Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):192–195, 1978.

[Ehr75] Andrzej Ehrenfeucht. Practical decidability. *J. Comput. Syst. Sci.*, 11:392–396, 1975.

[FG06] Jörg Flum and Martin Grohe. *Parameterized complexity theory.* Springer Heidelberg, 2006.

[FK09] Lance Fortnow and Adam R. Klivans. Efficient learning algorithms yield circuit lower bounds. *J. Comput. Syst. Sci.*, 75(1), 2009.

[FK10] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms.* Springer, 2010.

[FSS81] Merrick Furst, James Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, April 1984. See also FOCS'81.

[GMR13] Parikshit Gopalan, Raghu Meka, and Omer Reingold. Dnf sparsification and a faster deterministic counting algorithm. *Computational Complexity*, 22(2):275–310, 2013.

[Hås98] Johan Håstad. The shrinkage exponent of de morgan formulae is 2. *SIAM J. Comput.*, 27:48–64, 1998.

[Her11] Timon Hertli. 3-SAT faster and simpler - Unique-SAT bounds for PPSZ hold in general. In *FOCS*, pages 277–284, 2011.

[HMP$^+$93] András Hajnal, Wolfgang Maass, Pavel Pudlák, Mario Szegedy, and György Turán. Threshold circuits of bounded depth. *J. Comput. Syst. Sci.*, 46(2):129–154, 1993.

[IKW02] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. *J. Comput. Syst. Sci.*, 65(4):672–694, 2002.

[IM02] Kazuo Iwama and Hiroki Morizumi. An explicit lower bound of 5n - o(n) for boolean circuits. In *MFCS*, pages 353–364, 2002.

[IMP12] Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for AC^0. In *SODA*, pages 961–972, 2012.

[IMZ12] Russell Impagliazzo, Raghu Meka, and David Zuckerman. Pseudorandomness from shrinkage. In *FOCS*, pages 111–119, 2012.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[IPS13] Russell Impagliazzo, Ramamohan Paturi, and Stefan Schneider. A satisfiability algorithm for sparse depth two threshold circuits. In *FOCS*, pages 479–488, 2013.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[IW97] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *STOC*, pages 220–229, 1997.

[JMV13] Hamidreza Jahanjou, Eric Miles, and Emanuele Viola. Local reductions. Technical Report TR13-099, Electronic Colloquium on Computational Complexity, July 2013.

[Kan82] Ravi Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55(1):40–56, 1982.

[KC00] Valentine Kabanets and Jin-Yi Cai. Circuit minimization problem. In *STOC*, pages 73–79, 2000.

[KI04] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

[KKO13] Adam Klivans, Pravesh Kothari, and Igor C. Oliveira. Constructing hard functions using learning algorithms. In *CCC*, pages 86–97, 2013.

[KL82] Richard Karp and Richard Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28(2):191–209, 1982.

[Kol56] A. N. Kolmogorov. On the representation of continuous functions of several variables by superposition of continuous functions of a smaller number of variables. *Dokl. Akad. Nauk SSSR*, 108:179–182, 1956.

[KRT13] Ilan Komargodski, Ran Raz, and Avishay Tal. Improved average-case lower bounds for DeMorgan formulas. In *FOCS*, pages 588–597, 2013.

[KV94] Michael J. Kearns and Leslie G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *JACM*, 41(1):67–95, 1994.

[KvM02] Adam Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial hierarchy collapses. *SIAM J. Comput.*, 31(5):1501–1526, 2002.

[Lip94] Richard Lipton. Some consequences of our failure to prove non-linear lower bounds on explicit functions. In *Structure in Complexity Theory Conference*, pages 79–87, 1994.

[LMS11] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.

[Lov09] Shachar Lovett. Unconditional pseudorandom generators for low degree polynomials. *Theory of Computing*, 5(1):69–82, 2009.

[LR01] Oded Lachish and Ran Raz. Explicit lower bound of 4.5n - o(n) for boolena circuits. In *STOC*, pages 399–408, 2001.

[Lup59] O. B. Lupanov. A method of circuit synthesis. *Izvestiya VUZ, Radiofizika*, 1(1):120–140, 1959.

[LV96] Michael Luby and Boban Velickovic. On deterministic approximation of DNF. *Algorithmica*, 16(4/5):415–433, 1996.

[LVW93] Michael Luby, Boban Velickovic, and Avi Wigderson. Deterministic approximate counting of depth-2 circuits. In *Proceedings of the 2nd ISTCS*, pages 18–24, 1993.

[Mas79] W. J. Masek. Some NP-complete set covering problems. *Manuscript*, 1979.

[MTY11] Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto. Derandomizing hssw algorithm for 3-sat. In *COCOON*, pages 1–12. 2011.

[MZ13] Raghu Meka and David Zuckerman. Pseudorandom generators for polynomial threshold functions. *SIAM J. Comput.*, 42(3):1275–1301, 2013.

[Nis91] Noam Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, 11(1):63–70, 1991.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

[Oli13] Igor Oliveira. Algorithms versus circuit lower bounds. Technical Report TR13-117, ECCC, September 2013.

[PPSZ98] Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-sat. *JACM*, 52(3):337–364, 2005. (See also FOCS'98).

[PPZ97] Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. *Chicago J. Theor. Comput. Sci.*, 1999, 1999. See also FOCS'97.

[Raz87] Alexander A. Razborov. Lower bounds on the size of bounded-depth networks over the complete basis with logical addition. *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

[RR97] Alexander Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997.

[San10] Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and qbf satisfiability. In *FOCS*, pages 183–192, 2010.

[Sch80] Jacob Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *JACM*, 27(4):701–717, 1980.

[Sch02] Uwe Schöning. A probabilistic algorithm for k-SAT based on limited local search and restart. *Algorithmica*, 32(4):615–623, 2002.

[Sha49] Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell Syst. Techn. J.*, 28:59–98, 1949.

[Sho75] L. A. Sholomov. On one sequence of functions which is hard to compute. *Mat. Zametki*, 17:957–966, 1975.

[SM02] Larry J. Stockmeyer and Albert R. Meyer. Cosmological lower bound on the circuit complexity of a small problem in logic. *JACM*, 49(6):753–784, 2002.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *STOC*, pages 77–82, 1987.

[ST12] Kazuhisa Seto and Suguru Tamaki. A satisfiability algorithm and average-case hardness for formulas over the full binary basis. *Computational Complexity*, 22(2):245–274, 2013. See also CCC'12.

[Sub61] B. A. Subbotovskaya. Realizations of linear functions by formulas using +,*,-. *Soviet Mathematics Doklady*, 2:110–112, 1961.

[SW13] Rahul Santhanam and Ryan Williams. On medium-uniformity and circuit lower bounds. In *CCC*, pages 15–23, 2013.

[TX13] Luca Trevisan and TongKe Xue. A derandomized switching lemma and an improved derandomization of AC0. In *CCC*, pages 242–247, 2013.

[Val84] Leslie G. Valiant. A theory of the learnable. In *STOC*, pages 436–445, 1984.

[Vio09] Emanuele Viola. The sum of d small-bias generators fools polynomials of degree d. *Computational Complexity*, 18(2):209–217, 2009.

[Wil13] Ryan Williams. Natural proofs versus derandomization. In *STOC*, pages 21–30, 2013.

[Wil14] Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. In *STOC*, page to appear, 2014.

[Wil11] Ryan Williams. Nonuniform ACC circuit lower bounds. *JACM*, 61(1):2, 2014. See also CCC'11.

[Wil10] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013. See also STOC'10.

[Yab59] S. V. Yablonksi. The algorithmic difficulties of synthesizing minimal switching circuits. *Dokl. Akad. Nauk SSSR*, 124(1):44–47, 1959.

[Yao82] Andrew Yao. Theory and application of trapdoor functions. In *FOCS*, pages 80–91, 1982.

[Ž́83] Stanislav Žák. A Turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327–333, 1983.

[Zip79] R. E. Zippel. Probabilistic algorithms for sparse polynomials. In *International Symposium on Symbolic and Algebraic Manipulation*, pages 216–226, 1979.

Computer Science Department, Stanford University, Stanford, CA, USA
E-mail: rrw@cs.stanford.edu