Algorithms for Circuits and Circuits for Algorithms

(Invited Paper)

Ryan Williams
Computer Science Department
Stanford University
Stanford, CA, USA
Email: rrw@cs.stanford.edu

***Abstract*—The title of this paper is meant to highlight an emerging duality between two fundamental topics in algorithms and complexity theory.**

***Algorithms for circuits* refers to the design of interesting algorithms which can perform non-trivial circuit analysis of some kind, on either a circuit or a Boolean function given as a truth table. For instance, an algorithm determining whether a given circuit has an input that forces a true output would solve the NP-complete Circuit-SAT problem. Such an algorithm is of course 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 uniform algorithms with non-uniform circuit families (or proving such modeling is impossible). For instance, the NEXP versus P/poly question asks whether nondeterministic exponential-time algorithms can be simulated using non-uniform circuit families of polynomial size. It is widely believed that the answer is *no*, however the present mathematical tools available are still too crude to prove this kind of separation.**

This paper surveys these two generic subjects, the ways in which they arise, and connections that have been developed between them, focusing on the connections between non-trivial circuit-analysis algorithms and proofs of circuit size lower bounds. To give one example, if there is a nontrivial algorithm (running slightly faster than exhaustive search) that can determine if a given circuit computes a constant function, then it can be concluded that NEXP is not contained in P/poly. Informally, this connection can be interpreted as saying "some good algorithms for circuits imply there are no good circuits for some algorithms."

***Keywords*-satisfiability; derandomization; exact algorithms; learning; circuit complexity; parameterized algorithms**

I. 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 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 of consequences of such impossibility results, as well as consequences of possessing such interesting methods (in the hopes of an eventual proof by contradiction).

Another dictum is that algorithm design and analysis is, on the whole, an easier venture than proving lower bounds. Conventional wisdom states that 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 theoretical 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 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. Such implications present an excellent mathematical "arbitrage" opportunity for complexity theorists, to potentially prove hard lower bounds via supposedly easier algorithm design. (Moreover, this approach *has* recently led to new lower bounds.)

Some connections take the following form. Suppose there

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

[2]Similarities can already be found in the proof(s) that the Halting Problem is undecidable: the workhorse behind such results is the construction of a *universal Turing machine* that can run arbitrary Turing machine code given as input. This is a canonical example of a positive algorithmic result applied to prove an impossibility result.

is a Turing machine T which receives on its input tape a description of a finite logical circuit C, and on all "structured" circuits C, T is guaranteed to perform some nontrivial *analysis* of the function computed by C. (For example, T could determine whether C outputs the same value on all possible inputs to C, provided C is a "shallow" circuit with a few layers of gates.) Such a T can then be used to construct a function f that is computable "somewhat efficiently" by a Turing machine but is *not* computable efficiently by *non-uniform* circuit families possessing that structure. That is, an interesting circuit-analysis algorithm can be applied to prove an interesting circuit complexity lower bound.

It is worth emphasizing the quantifiers in the above implication schema:

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

That is, there are situations in which designing algorithms for some 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. Careful design of algorithms for analyzing given instances of the circuit model are used to construct functions computable (in one sense) in 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.

This article will survey 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 EXP versus P/poly question asks whether exponential-time algorithms can be simulated using non-uniform circuit families of polynomial size.
- *Algorithms for circuits* refers to designing interesting algorithms which can perform some interesting circuit analysis. The input may be a circuit or it may be a Boolean function (given as a truth table), and the algorithm checks whether the circuit (or truth table) satisfies a simple property related to the circuit complexity of the underlying function. To illustrate, an algorithm determining if a given circuit has an input that forces a true output solves the NP-complete Circuit-SAT problem. It is an outstanding open question whether one can asymptotically improve over the "brute force" algorithm that tries all possible inputs to the circuit.

The rest of the paper is organized as follows. The next section provides a bit of background from complexity theory and circuit complexity. Section III surveys the topic of *circuits for algorithms*, modeling algorithms with non-uniform circuit families. Section IV surveys existing knowledge of circuit-analysis algorithms, which we call *algorithms for circuits*. Section V discusses known connections between the two, and prospects for future progress. Section VI briefly concludes.

II. Preliminaries

We assume familiarity with machine-based complexity theory [AB09] but not necessarily circuit complexity.

Circuit complexity is concerned with how to construct Boolean functions out of "simpler" functions, such as those of the form $g : \{0,1\}^2 \to \{0,1\}$. Examples of 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 $\sum_i x_i \geq \lceil k/2 \rceil$.

Circuit complexity: 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 called 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, we will not count gates with fan-in 1 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 a function $f : \{0,1\}^n \to \{0,1\}$, what is the minimal size s of a Boolean circuit over $\mathcal{B}$ with output gate $f_{n+s} = f$? We call this quantity the $\mathcal{B}$-circuit complexity of f, and it is typically denoted by $C_{\mathcal{B}}(f)$. The minimal depth of a circuit computing f is also of interest for parallel computing; the minimal depth is denoted by $D_{\mathcal{B}}(f)$.

III. Circuits for Algorithms

The circuit model is excellent for understanding the difficulty and efficiency of computing *finite* functions. Boolean circuits should be contrasted with the typical *uniform algorithm* models used in computability and complexity theory, based on objects such as Turing machines deciding and/or recognizing infinite languages of the form

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

All finite functions are trivially computable by Turing machines in *constant* time.

There is a natural way to extend the Boolean circuit model to also compute functions of type (1): 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 provides an *infinite* (so-called *non-uniform*) computational model. If an L can be shown to have $s(n)$-size circuits for small $s(n)$, it means that the circuit complexities of all finite segments of L scale well with the input length.

Every Boolean function has circuits of size $2^n/n + o(2^n/n)$ size [Sha49], [Lup59], and this upper bound is tight by a simple counting argument. The class of functions of type (1) which are computable with polynomial-size 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 $L \in \mathsf{SIZE}(n^k + k)$.

Proving that a function is *not* in P/poly is a very strong result, showing that even computing finite segments of the problem requires quite "large" computations, relative to the size of the segment we wish to compute.

Immediately one wonders how the two computational models (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 which uniform algorithms (efficient or inefficient) do efficient circuit families exist? It is believed that in general, circuit families cannot really solve NP-hard problems significantly more efficiently than algorithms can, and $\mathsf{NP} \not\subset \mathsf{P}/\text{poly}$. This would imply $\mathsf{P} \neq \mathsf{NP}$, and complexity theory is *very* far from proving this.

In general, because P/poly contains undecidable problems and permits an "infinitely long" computational model, the familiar tools of computability theory are essentially powerless for understanding P/poly, and complexity theory has not yet discovered enough new tools. While nontrivial results (which we now survey) are known, they are meager in comparison to what is conjectured.

A. Classes with efficient circuits

It is well-known that $\mathsf{P} \subset \mathsf{P}/\text{poly}$ and $\mathsf{BPP} \subset \mathsf{P}/\text{poly}$, so both classes have efficient circuits. Aside from the occasional undecidable problem, there are few known surprises in terms of efficient circuit simulations. One surprise might be:

Conjecture 3.1 (A. N. Kolmogorov [Lip94]): For every $L \in \mathsf{P}$, there is a k such that L has kn size circuits.[3]

The truth of the conjecture would be surprising, because for languages that require (for example) $n^{100^{100}}$ time but are contained in P, it appears unlikely that the complexity of such problems would magically shrink to $O(n)$ size, merely because one can design a different circuit for each input length. Kolmogorov's conjecture would imply $\mathsf{P} \neq \mathsf{NP}$ [Kan82], [Lip94].

While it is generally believed that Conjecture 3.1 isn't true, a resolution looks very difficult. To see why, we sketch here the lack of progress on circuit lower bounds for P. For $L : \{0,1\}^\star \to \{0,1\}$, 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 3.1 ([Blu84]): There is an $L \in \mathsf{P}$ with $C_{B_2}(L_n) \geq 3n - o(n)$ for all n.

Theorem 3.2 ([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 V 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].[4]

[3]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.

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

B. Classes without efficient circuits

It's believed that NP is not in P/poly, but the results stated above show that a proof is probably distant. Let us survey which functions *are* known to not be in P/poly.

Ehrenfeucht [Ehr75] showed that the problem of deciding sentences in the first order theory of $\mathbb{N}$ with addition, multiplication, and exponentiation, where all quantified variables are bounded by constants, 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] showed that the same problem is decidable by a Turing machine using exponential ($2^{O(n)}$) space. That is, $\mathsf{EXPSPACE} \not\subset \mathsf{SIZE}((1+\delta)^n)$. This result can be scaled down to the presumably smaller complexity class $\Sigma_3\mathsf{EXP}$.

Kannan [Kan82] proved that $\Sigma_2\mathsf{EXP} \not\subset \mathsf{P}/\mathsf{poly}$. (With regards to Kolmogorov's conjecture, his work also proves that $\Sigma_2\mathsf{P} \not\subset \mathsf{SIZE}(O(n))$.) In fact his proof shows that $\Sigma_2\mathsf{EXP} \not\subset \mathsf{SIZE}(f(n))$, for every $f : \mathbb{N} \to \mathbb{N}$ satisfying $f(f(n)) \leq 2^n$ (these are the *half-exponential* functions). It is a longstanding open problem to show that $\Sigma_2\mathsf{EXP} \not\subset \mathsf{SIZE}(2^{\varepsilon n})$ 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] of exponential-time Merlin-Arthur games. However, it is open whether $\mathsf{NEXP} \subset \mathsf{P}/\mathsf{poly}$. It looks extremely unlikely that problems verifiable with *exponentially-long witnesses* could be computed using only polynomial-size circuits, but the infinite nature of the circuit complexity model has confounded all proof attempts. Recently a new approach has been found to attack this problem, which is a major subject of Section V.

C. Restricted circuits

The circuit model is extremely general. There are several natural ways to restrict the model beyond just a polynomial size measure, 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 subsets 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.[5]
- 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 $d \log^k n$.[6]
- $\mathsf{ACk[m]}$: Languages computable with polynomial size, $O(\log^k n)$ depth circuits over M_m.
- ACCk: The union over all $m \geq 2$ of $\mathsf{ACk[m]}$.[7]
- TCk: Languages computable with polynomial size, $O(\log^k n)$ depth circuits over the basis T.[8]

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

$$\mathsf{AC0} \subsetneq \mathsf{AC0[p]} \subsetneq \mathsf{ACC0} \subseteq \mathsf{TC0} \subseteq \mathsf{NC1},$$

where p is prime.

$\mathsf{NC1}$ 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. The best known formula size lower bound for a function in P is $n^{3-o(1)}$, by Håstad [Hås98]. $\mathsf{TC0}$ is also 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 $\mathsf{TC0}$. (Recall 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 appearance of MODm operations may look a bit strange, but they arose out of a specific program to understand circuit complexity in a "bottom up" way, by starting with very restricted circuits and attempting to gradually relax the restrictions. First, it was shown that MOD2 $\notin \mathsf{AC0}$ [Ajt83], [FSS81]. This made it natural to ask what is computable when the MOD2 function is provided among the basis functions in constant-depth circuits, leading to the definition of $\mathsf{AC0[2]}$. Then it was proved that for distinct primes p and q, MODq $\notin \mathsf{AC0[q]}$ [Raz87], [Smo87], hence MOD3 $\notin \mathsf{AC0[2]}$. Naturally one then wonders what

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

[6]AC stands for "Alternating Circuits," alternating between AND and OR. We remind the reader that NOT gates are not counted in the depth bounds of AC, ACC, and TC circuits.

[7]ACC stands for "Alternating Circuits with Counting."

[8]TC stands for "Threshold Circuits."

is computable when MOD3 and MOD2 are both allowed in the basis. It is not hard to see that including MOD6 as a basis function is equivalent to including MOD3 and MOD2. Attention turned to $\mathsf{AC0}[6]$. (There were many other separate threads of research, such as lower bounds on further restricted versions of $\mathsf{TC0}$ [HMP$^+$93], which we do not have space to cover here.)

At this point, the trail was lost. It is still open whether every language in $\mathsf{P/poly}$ has depth-three circuit families over M_6. It's also open whether $\mathsf{AC0}[6]$ can compute every problem in EXP. It has been shown only recently that $\mathsf{NEXP} \not\subset \mathsf{ACC0}$, via a generic connection between algorithms-for-circuits and circuits-for-algorithms [Wil10], [Wil11]. Yet it is open whether NEXP is contained in $\mathsf{TC0}$, even for $\mathsf{TC0}$ circuits of depth three.

IV. Algorithms For Circuits

Circuit analysis problems can take multiple forms. The most common form takes a circuit as input, and decides some property of the function computed by the circuit. Let 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?

As Circuit-SAT is NP-complete, it is unlikely that there is an efficient algorithm for it, and the brute force algorithm takes $\Omega(2^n \cdot |C|)$ 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? Fine-grained questions of this variety are basic to two emerging areas of research: parameterized algorithms [DF99], [FG06] and exact algorithms [Woe03], [FK10]. For many NP-hard problems, asymptotically improved algorithms over exhaustive search do exist, and researchers actively study the extent to which exhaustive search can be avoided.

Could Circuit-SAT be solved in $(1+\varepsilon)^n|C|^2$ time, for *every* constant $\varepsilon > 0$? A central conjecture in the area, known as the *Exponential Time Hypothesis* (ETH for short), would imply that such algorithms do *not* exist. We will speak more about ETH shortly.

A. Flavors of Circuit-SAT

As seen in Section III, there are many interesting restrictions one can place on circuits. For each restriction, a new satisfiability problem emerges. Here we survey the known algorithms for satisfiability for restricted circuit classes.

In this section, we construe the classes $\mathsf{AC0}$, ACC, $\mathsf{TC0}$, $\mathsf{NC1}$, and $\mathsf{P/poly}$ as not as classes of languages,but as *classes of circuit families*: collections of infinite circuit families satisfying the appropriate restrictions. For each circuit 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?

Of course, $\mathcal{C}$-SAT is NP-complete for all interesting circuit classes. But for simple enough $\mathcal{C}$, $\mathcal{C}$-SAT algorithms running significantly faster than exhaustive search *are* known, although $\mathcal{C}$-SAT is NP-complete.

k-SAT algorithms: The k-SAT problem is perhaps the simplest NP-hard satisfiability problem, the circuit being an AND of ORs of k literals, in conjunctive normal form (CNF). 3-SAT can be solved in 1.331^n time deterministically [MTY11], or in 1.308^n time [Her11] with a randomized algorithm. These running times form the tail end of a long line of successive improvements, 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 implies that many other NP-hard problems require essentially exponential time to solve, using clever *subexponential time reductions*. Therefore, this is not just a hypothesis about one NP-complete problem: it says that many problems will not admit subexponential time algorithms (running in $(1+\delta)^n$ for any prescribed $\delta > 0$). Many other consequences of ETH have been found [LMS11].

The k-SAT problem for general k has also been extensively studied. The best known algorithms for k-SAT all run in $2^{n-n/(ck)}$ time, for a fixed constant c [PPZ97], [Sch02], [PPSZ98], [DH09]. That is, 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.)[9]

AC0-SAT: The next more expressive class of circuits is AC0, with arbitrary constant depth. There has been less work on this case, 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: Further more expressive is the circuit class ACC0. The author [Wil11] gave an algorithm running in $O(2^{n-n^{\varepsilon}})$ time for ACC0 circuits of $2^{n^{\varepsilon}}$ size, where $\varepsilon \in (0,1)$ 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 presumably larger circuit class ACC0 $\circ$ THR; that is, ACC0 augmented with an additional layer of arbitrary linear threshold gates near the inputs [Wil14].

TC0-SAT: For depth-two TC0, Impagliazzo, Paturi, and Schneider [IPS13] showed that satisfiability of circuits on n inputs and cn wires 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 TC0 (and nothing is known in terms of circuit lower bounds for this class, 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.)

Circuit-SAT: For Circuit-SAT, there is no known algorithm for solving the problem on generic circuits of size s and n inputs that is asymptotically faster than the cost of exhaustive search, i.e., $O(2^n \cdot s)$ time.

B. 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 $\pm 1/10$.

Here, the constant $1/10$ is arbitrary, and could be any sufficiently small constant in $(0, 1/2)$.

As with $\mathcal{C}$-SAT, the problem $\mathcal{C}$-CAPP can be defined for a restricted circuit class $\mathcal{C}$. Approximate counting has been extensively studied in complexity theory, due to its connections to *derandomization*. We cannot hope to cover all work on this topic in the space of this article, and can only provide highlights. The structure of this subsection will parallel that of the previous coverage of $\mathcal{C}$-SAT.[10]

Several algorithms we shall mention give a stronger property than the ability to approximately count satisfying assignments: they are *pseudorandom generators* which output a fixed collection of assignments to evaluate the circuit on, prior to viewing the circuit. Pseudorandom generators are inherently tied to lower bounds. Indeed, lower bounds against a circuit class $\mathcal{C}$ are a prerequisite for constructing efficient pseudorandom generators fooling $\mathcal{C}$.

Ajtai and Wigderson [AW85] showed that AC0-CAPP is solvable in $2^{n^{\varepsilon}}$ time for every $\varepsilon > 0$, by providing a pseudorandom generator. Nisan [Nis91] gave a pseudorandom generator which yields an approximate counting 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 AC0, showing that AC0-CAPP can be computed in $n^{\tilde{O}(\log^{d-1} s)}$ time.

The depth-two case is especially interesting. Luby and Velickovic [LV96] showed in this case that CAPP is solvable 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 a deterministic polynomial-time algorithm for approximate counting satisfying assignments to CNFs may be within reach.

[9]The author can't help but confess his belief here that SETH is false. Many of his papers were conceived by finding an approach to refute SETH which ultimately failed, but was applicable to another problem instead (Max-2-SAT, ACC-SAT, All-Pairs Shortest Paths, etc.)

[10]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), yielding a $\mathcal{C}$-CAPP algorithm.

For the class 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 approximate counting algorithm for depth-two TC0 circuits is known. However, here is a good place to mention two other threads of work in this area that relate. 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 the problem of computing CAPP on a MODp of AND gates of fan-in d. This problem is now known to be solvable essentially optimally for fixed d, in deterministic time $O_d(n^d)$ [NN93], [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 corresponds to *true* and 1 corresponds to *false*.) 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 of degree-d PTFs can be done in polynomial time [MZ13].

For Boolean formulas, Impagliazzo, Meka, Zuckerman [IMZ12] give a pseudorandom generator yielding an $2^{O(s^{1/3+o(1)})}$ 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 pseudorandom generator can be used to approximately count in $2^{O(s^{1/2+o(1)})}$ time.

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

C. 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 and notoriously hard example of such a problem 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 $C_{B_2}(f) \le k$?

That is, the task is 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 widely conjectured that Circuit-Min is intractable: for one, if the problem 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 random functions have high circuit complexity with high probability, and f has low circuit complexity, an efficient algorithm for Circuit-Min be used to could tell the two apart with non-negligible success probability, after querying them at poly(n) points.

Indeed, putting Circuit-Min in P would consequently provide a "P-natural property useful against exponential-size circuits" in the sense of Razborov and Rudich [RR97]. As a result, some restricted versions of Circuit-Min, such as NC1-Min and TC0-Min are also intractable under cryptographic assumptions.

However, proving that Circuit-Min is NP-hard is a difficult open problem. To obtain a polynomial-time reduction from (say) 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 restricted case 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 [Qui52], [McC56], [Mas79], [AHM+08]. However, one can efficiently find an *approximately* minimum-sized DNF [AHM+08].

Another truth table analysis problem has recently been introduced is that of *compression*:

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 many existing circuit lower bound proofs can be "mined" to obtain interesting (faster than exhaustive search) compression algorithms for AC0, small Boolean formulas, and small branching programs. They pose as an open problem whether 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 $H(x) \neq f(x)$.

It's well known that pseudorandomness and cryptography naturally connect with computational learning: a pseudorandom function is one implemented with small circuits yet "looks like a random function" when it's queried a small number of times—the kind of function which isn't easily learnable. 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].

V. Connections

In the *Circuits for Algorithms* space, we wish to design small circuits to simulate complex algorithms, or prove that no small circuits exist for this task. In *Algorithms for Circuits*, the goal is to design faster circuit-analysis algorithms. It is perhaps natural to hypothesize that these tasks may be used to inform the other. A provably nontrivial algorithm for analyzing all possible circuits from a class must exhibit, at its core, some nontrivial understanding about the limitations of that circuit class. In the converse direction, if a simple function cannot be computed by small circuits, then algorithms may be able to use this function to exploit this limitation, and analyze small circuits faster than exhaustive search would take.

For sufficiently restricted classes of circuits, one can adapt the known techniques for proving lower bounds to derive faster satisfiability algorithms (or pseudorandom generators) for those circuits. For instance, the progress on Formula-SAT algorithms and on pseudorandom generators for Boolean formulas, both mentioned in Section IV, 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 the existence of efficient circuit-analysis algorithms and circuit lower bounds.

A. 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]. There, the existence of cryptographic pseudorandom generators 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 gave some 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 all "low complexity" circuits.

Babai, Fortnow, Nisan, and Wigderson [BFNW93] showed that $\mathsf{EXP} \not\subset \mathsf{P}/\mathrm{poly}$ implies the existence of pseudorandom generators that can deterministically simulate any randomized polynomial-time algorithm in subexponential time. Formally speaking:

Theorem 5.1 ([BFNW93]): $\mathsf{EXP} \not\subset \mathsf{P}/\mathrm{poly}$ 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 some function computable in $2^{O(n)}$ time that needs circuits of size at least $(1+\delta)^n$ for almost all input lengths n, then $\mathsf{P} = \mathsf{BPP}$.

That is, from exponential-size circuit lower bounds, every randomized polynomial-time algorithm can be simulated in deterministic polynomial time. Impagliazzo, Kabanets, and Wigderson [IKW02] showed that even a seemingly weak lower bound like $\mathsf{NEXP} \not\subset \mathsf{P}/\mathrm{poly}$ would imply a derandomization result: namely, that 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 approximating the acceptance probability of a circuit 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 $\mathsf{NEXP} \not\subset \mathsf{P}/\mathrm{poly}$.

The fastest known algorithm for CAPP is exhaustive search, requiring $\Omega(2^n)$ time. An improvement in the running time from 2^n time to 2^{n^ε} time for every $\varepsilon > 0$ would be an incredible accomplishment, and a remote possibility for the near future. However, the hypothesis of Theorem 5.3 can be significantly weakened. Essentially any nontrivial improvement over exhaustive search for CAPP would already yield the desired lower bounds:

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

Furthermore, it is also known that 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.

B. 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 $\mathsf{EXP} \not\subset \mathsf{P}/\mathrm{poly}$.

This is an interesting conditional statement, 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 collapses of existing complexity classes, including $\Sigma_3\mathsf{EXP} = \mathsf{EXP}$. However, as stated in Section III, $\Sigma_3\mathsf{EXP}$ 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 the feasibility offered by a 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 be obtained via proof by contradiction, in this way: assert a feasibility hypothesis strong enough that the complexity of another *provably hard* function reduces drastically, to the point where it becomes contradictorily *easy*. Sure enough, the recent progress of the author on $\mathsf{ACC0}$ circuit 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 $\mathsf{EXP} \not\subset \mathsf{P}/\mathrm{poly}$. 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 has less than maximum size circuits, and hence $\mathsf{P} \neq \mathsf{NP}$, or EXP requires maximum size circuits, and every randomized algorithm could then be efficiently simulated deterministically, by Theorem 5.2! So no matter how the EXP vs P/poly problem is resolved, the consequences will be very interesting.

Modern times: Theorem 5.5 and its offshoots only work for Circuit-SAT algorithms that run in subexponential time. An indication that techniques for weak SAT algorithms may still be useful for circuit lower bounds can be found in work of Paturi, Pudlak, and Zane [PPZ97]. They gave a structure lemma on k-SAT instances, and used it to not only prove that k-SAT can be solved in about $2^{n-n/k}$ time, but also to prove 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 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 IV-A 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 $n^{\mathrm{poly}(\log n)}$ size circuits.) The second extension strengthens $\mathsf{ACC0}$ up to the class $\mathsf{ACC0} \circ \mathsf{THR}$, or $\mathsf{ACC0}$ circuits augmented with a layer of linear threshold gates near the inputs [Wil14].

Theorem 5.6 holds for all circuit classes $\mathcal{C}$ mentioned in the survey, but one may need (for example) a satisfiability 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? Indeed, from faster 3-SAT algorithms we can conclude superlinear size lower bounds:

Theorem 5.8 ([Wil10]): Suppose the Exponential Time Hypothesis 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.

The hypothesis was discussed in Section IV-A, and the conclusion was discussed as open in Section III-A. Refuting the Strong Exponential Time Hypothesis (SETH) from Section IV-A 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 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 solve problems like satisfiability. 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 by contradiction. We assert that both a faster algorithm for analyzing $\mathcal{C}$ exists, and that $\mathsf{NEXP} \subset \mathcal{C}$. Then, it is shown that these two assumptions imply an algorithm which is too good to be true: a way to simulate every language solvable in nondeterministic $O(2^n)$ time, in only $o(2^n)$ time. This simulation contradicts the nondeterministic time hierarchy theorem [Žák83]. The faster nondeterministic simulation works by using $\mathsf{NEXP} \subset \mathcal{C}$ to nondeterministically guess $\mathcal{C}$ circuits that help perform a arbitrary $2^{O(n)}$ time computation, and using the faster circuit-analysis algorithm to verify that these $\mathcal{C}$ circuits do the job correctly.

C. Circuit lower bounds from learning

Intuitively, circuit lower bounds require understanding something deep about the circuit class under consideration, which would seem to be necessary for any efficient learning algorithm for circuits. Fortnow and Klivans proved a theorem modeling this intuition. Let $\mathcal{C}$ be a restricted circuit class, such as those defined in Section III-C. In the following, we 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 IV-C).

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 follow 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 $f(x)$). For every equivalence query, 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, it cannot have circuits in $\mathcal{C}$.

D. Circuit analysis as a complexity barrier

The Algorithms for Circuits problem connects with circuit lower bounds in a *negative* way as well. The Natural Proofs Barrier [RR97] has the following setup. Suppose one establishes a superpolynomial circuit lower bound against a language L, by giving an algorithm A that, given the truth table of a function $f : \{0,1\}^n \to \{0,1\}$:

- *[A is constructive]* A runs in time $2^{O(n)}$, polynomial in the input length.
- *[A is large]* A accepts at least a $1/2^{O(n)}$ fraction of all 2^n-bit inputs.
- *[A is useful]* For all k, there are infinitely many n such that A accepts the truth table of L_n and A rejects all strings which are truth tables for circuits of size n^k.[11]

Then A is said to be *natural*. One can see that the algorithm A is basically a weak algorithm for the Circuit-Min problem (cf. Section IV-C): the algorithm can distinguish many functions that are "hard" from all functions which are sufficiently "easy." Practically all known circuit lower bound proofs (with rare exception) effectively construct such natural algorithms A for distinguishing the function they want to prove hard from a given circuit class.

Razborov and Rudich proved that when an algorithm A exists which is *useful against a circuit class* $\mathcal{C}$, then that

[11]This is the definition of usefulness for polynomial size circuits. For restricted circuit classes, one would add other conditions in a natural way.

circuit class $\mathcal{C}$ is too weak to support modern cryptography! (In particular, $\mathcal{C}$ cannot support pseudorandom functions.) Therefore, if we believe that it is possible to prove circuit lower bounds which are strong enough for doing unconditionally secure cryptography with circuit class $\mathcal{C}$, then we must also believe that such "natural" approaches to lower bounds cannot establish results like $\mathsf{P} \neq \mathsf{NP}$.

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 solving CAPP. Impagliazzo, Kabanets, and Wigderson [IKW02] proved an explicit form of such an equivalence:

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

That is, NEXP circuit lower bounds are *equivalent* to the existence of nontrivial nondeterministic subexponential time algorithms for approximately counting SAT assignments to circuits.

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

That is, on infinitely many n, algorithm A outputs 1 on at least one Boolean function on n bits, and 0 on all functions with polynomial circuit complexity. This is a very weak algorithmic guarantee.

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.

Using the language of Natural Proofs, this theorem says: $\mathsf{NEXP} \not\subset \mathcal{C}$ if and only if there is a *constructive property useful against $\mathcal{C}$ circuits*. Recall that the Natural Proofs barrier teaches us that we must find lower bound proofs which avoid constructivity, largeness, and usefulness. Hence one may interpret the theorem as saying that largeness can always be "avoided": from *any* proof of an NEXP circuit lower bound we can extract a constructive and useful property.

Connections in an algebraic setting: In this survey, we have restricted our attention to 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?

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 $\mathsf{NEXP} \not\subset \mathsf{P}/\text{poly}$, or computing the permanent of a matrix requires superpolynomial-size algebraic circuits.

VI. Conclusion

This survey 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.

Acknowledgment

The author acknowledges support by the Alfred P. Sloan foundation, Microsoft Research, and the NSF under grant CCF-1212372.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[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.

[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.

[McC56] E. L. McCluskey Jr. Minimization of boolean functions. *Bell System Tech. J.*, 35:1417–1444, 1956.

[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.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993.

[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.

[Qui52] W. V. Quine. The problem of simplifying truth functions. *Amer. Math. Monthly*, 59:521–531, 1952.

[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 *ACM 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.

[Woe03] Gerhard J Woeginger. Exact algorithms for NP-hard problems: A survey. In *Combinatorial Optimization—Eureka, You Shrink!*, pages 185–207. Springer, 2003.

[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.