The Computational Complexity Column

by

V. Arvind

Institute of Mathematical Sciences, CIT Campus, Taramani
Chennai 600113, India
arvind@imsc.res.in
http://www.imsc.res.in/~arvind

The 1980's was a golden period for Boolean circuit complexity lower bounds. There were major breakthroughs. For example, Razborov's exponential size lower bound for monotone Boolean circuits computing the Clique function and the Razborov-Smolensky superpolynomial size lower bounds for constant-depth circuits with MOD_p gates for prime p. These results made researchers optimistic of progress on big lower bound questions and complexity class separations. However, in the last two decades, this optimism gradually turned into despair. We still do not know how to prove superpolynomial lower bounds for constant-depth circuits with MOD_6 gates for a function computable in exponential time.

Ryan Williams' exciting lower bound result of 2011, that nondeterministic exponential time does not have polynomial-size unbounded fanin constant-depth circuits with MOD_m gates for any composite m, has renewed optimism in the area. The best part is that his approach is potentially applicable to other lower bound questions.

In this wonderful article, Rahul Santhanam explores this theme of connections between improved SAT algorithms and circuit lower bounds.

Ironic Complicity: Satisfiability Algorithms and Circuit Lower Bounds

Rahul Santhanam
University of Edinburgh
rsanthan@inf.ed.ac.uk

Abstract

I discuss recent progress in developing and exploiting connections between SAT algorithms and circuit lower bounds. The centrepiece of the article is Williams' proof that $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$, which proceeds via a new algorithm for ACC^0-SAT beating brute-force search. His result exploits a formal connection *from* non-trivial SAT algorithms *to* circuit lower bounds. I also discuss various connections in the reverse direction, which have led to improved algorithms for k-SAT, Formula-SAT and AC^0-SAT, among other problems.

1 Introduction

Theoretical computer science suffers from a dichotomy between the *algorithmic* endeavour and the *complexity-theoretic* endeavour. Algorithmists strive to design the most efficient algorithms for problems of interest, while complexity theorists investigate which problems are hard to solve, and why. Algorithmists focus on concrete problems, while complexity theorists often work in a more abstract framework, proving general theorems about computation. Algorithmists use constructive methods, while the enterprise of proving complexity lower bounds seems an inherently non-constructive one.

But is this dichotomy fundamental? At some level, algorithmists and complexity theorists are studying two sides of the same question: which is the most efficient solution for a problem? A priori, one would imagine that a deep understanding of the structure of a computational problem would assist both in designing the most efficient solution possible, as well as proving that no more efficient solution exists. In part because the theory of computation is still at a fairly early stage in its development, and in part because the basic questions seem to be very difficult, this has not often been the case so far. The algorithms community and the complexity theory community have pursued their research programs more or less independently.

Recent developments have the potential to change this, opening the possibility of greater interaction and accelerated progress in both areas. These developments hint at a *complicity* between algorithms and lower bounds, which is ironic in that these endeavours seem superficially to be in opposition.

The most significant such development is the recent work of Williams [37, 38] proving that $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$. This work has attracted a great deal of interest, since lower bound breakthroughs are rare. Though the result is interesting in itself, what is more interesting is the conceptual message of Williams' work, which is that algorithms for Satisfiability (SAT) can be used to prove lower bounds, and that there are strong connections between the two endeavours.

In this article, I give a sampler of work in the past couple of decades which shares this message. I make no claim that this is an exhaustive survey of the connections between SAT algorithms and lower bounds. Rather, I aim to give illustrations of the various connections that exist, and an indication of what the most promising research directions might be. This is a very actively growing area, and my hope is that this article could serve as a rough "road-map" for researchers wishing to work in this area, or else as a quick digest for those who are curious about the recent developments.

1.1 Historical Context

The connection between lower bounds and algorithms can be traced back to the pioneering work of Yao [39] and Blum & Micali [8] on pseudo-random generators. They showed how to construct cryptographic pseudo-random generators based on strong average-case circuit lower bounds. Cryptographic pseudo-random generators can be used to define sub-exponential time algorithms for problems in BPP, beating the trivial brute-force bound. Indeed, this implication was explicitly noted in Yao's paper [39].

Yao's connection is in a sense a byproduct of a conceptual machinery designed for cryptographic problems. In an influential paper, Nisan & Wigderson [28] adapted the notion of a pseudo-random generator to the context of complexity theory, and gave tighter implications from circuit lower bounds to pseudo-random generators, and vice versa. Since then, a sequence of papers [23, 26, 20], have established progressively tighter and more refined versions of these implications, and it is now known that circuit lower bounds for E (linear exponential time) against a class C of circuits are more or less equivalent to pseudo-random generators which are resilient to statistical tests from C, for essentially any natural class C of circuits. While pseudo-random generators imply improved deterministic simulations for problems in BPP, the converse is not the case. However, Kabanets & Impagliazzo [24] have shown that sub-exponential time algorithms for the Polynomial Identity Testing (PIT) problem actually imply circuit lower bounds against arithmetic circuits. A weak converse of this result is known as well, showing a deep connection between algorithms and circuit lower bounds in this setting.

Though these results in the theory of pseudo-randomness are fairly strong, the connections haven't led to much progress either on lower bounds or on algorithms. The reason is that the known algorithmic ideas for solving PIT fall well short of having implications for pseudo-random generators, and hence for lower bounds. We won't discuss the pseudo-randomness literature further in this survey, but we note that it heavily influenced the formation of the connections we *will* discuss both historically, as well as methodologically.

There are other areas of theoretical computer science where progress on hardness results has gone hand-in-hand with new algorithms. This is the case, for example, with the recent work on semi-definite programming algorithms and the Unique Games conjecture [31], with the caveat that the notion of hardness there is conditional, i.e., based on reductions from presumed hard problems rather than on proven lower bounds. There is also the sophisticated and ambitious Geometric Complexity Theory (GCT) approach of Mulmuley & Sohoni [27] towards proving complexity lower bounds, which relies ultimately on algorithmic conjectures. We do not discuss these other examples of complicity between algorithms and lower bounds, but they do add to the evidence that there is something fundamental about this phenomenon.

1.2 Plan of the Article

Following on a short section establishing relevant notation, there are three main sections to this article discussing recent work, and a final section speculating on future research directions. The first section discusses a series of papers by Paturi, Zane and others proving structural theorems about CNF formulas which were then exploited both in an algorithmic context and to prove lower bounds. These were the earliest papers showing connections between exact algorithms for Satisfiability and circuit lower bounds. The middle section discusses the breakthroughs of Williams, which demonstrate and use a formal connection *from* SAT algorithms to lower bounds. The final section discusses various subsequent works which exploit connections in the reverse direction to give new and improved algorithms for variants of SAT such as Formula-SAT and AC^0-SAT.

Throughout this article, I will favour heuristic arguments over precise ones in cases where the former are more helpful in establishing intuition.

2 Preliminaries

I assume knowledge of the basic concepts of complexity theory. The book by Arora and Barak [1] and the Complexity Zoo (which can be found at `http://qwiki.caltech.edu/wiki/ComplexityZoo`) are good references.

I will be dealing with several variants of Satisfiability. For a positive integer k, k-SAT is the satisfiability problem for k-CNFs. CNF-SAT is the satisfiability problem for CNFs without any restriction on clause size. Formula-SAT is the satisfiability problem for formulas over the De Morgan basis. Circuit-SAT is the satisfiability problem for Boolean circuits. In general, given a class C of circuits, C-SAT is the satisfiability problem for circuits in C. I will refer simply to "SAT" when I wish to speak of the Satisfiability problem generally rather than of a specific variant.

Definition 1. *A parametric problem p-L consists of a language $L \subseteq \{0,1\}^*$ together with a parameter function $n : \{0,1\}^* \rightarrow \mathbb{N}$. Given a function $t : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, we say that p-$L$ is solvable (resp. probabilistically solvable) in time t if there is a deterministic (resp. probabilistic) algorithm which decides L correctly and runs in time $t(|x|, n(x))$ on all inputs x.*

I will only be considering parametric versions of SAT variants, and for these problems there is a very natural notion of parameter: the number of variables in the formula. For any SAT variant L, p-L is the parametric problem corresponding to L.

The notion of "non-trivial" solvability of SAT can now be defined.

Definition 2. *A SAT variant L is said to have a non-trivial algorithm if p-L is solvable in time t, where $t(m,n) = O(\mathsf{poly}(m)2^{n-\omega(\log(n))})$.*

There is a natural notion of the "savings" an algorithm for SAT achieves over brute-force search. Note that the brute-force search algorithm operates in time $2^n\mathsf{poly}(m)$.

Definition 3. *Given a function $c : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, a SAT variant L is said to have savings (resp. probabilistic savings) c if p-L is solvable (resp. probabilistically solvable) in time t, where $t(m,n) = O(\mathsf{poly}(m)2^{n-c(m,n)})$.*

Thus a non-trivial algorithm achieves savings $\omega(\log(n))$, and $\mathsf{NP} = \mathsf{P}$ iff 3-SAT has savings $n - O(\log(n))$.

For information on the best known upper bounds for variants of SAT, refer to the survey by Dantsin and Hirsch [11]. Here I only discuss upper bound techniques and results which connect in some way to lower bounds.

However, it might be useful to say something about the common algorithmic paradigms for SAT. There are essentially two commonly used paradigms: the *DLL* paradigm and the *local search* paradigm. Algorithms belonging to the DLL paradigm operate as follows. At each stage in the algorithm, a fixed rule is used to select a variable in the formula and a value to assign to it. With the variable set accordingly, the formula is simplified according to standard simplification rules, and the algorithm proceeds to the next stage. If at any stage, the formula simplifies to "true", the algorithm halts, since a satisfying assignment has been found. If it simplifies to "false", the algorithm "backtracks" by re-setting the most recently set variable to the other possible value and recursing. Intuitively, a DLL procedure explores a tree of candidate satisfying assignments, where nodes correspond to variables and edges to values which can be assigned to a given variable, with leaves being labelled "true" or "false". The procedure aims to construct and explore this tree in the most efficient possible manner, and the number of leaves of the tree gives a bound on the running time.

Algorithms belonging to the local search paradigm operate as follows. An initial assignment is chosen, and if this assignment is not already satisfying, the algorithm explores the space of assignments by changing the value of one variable at a time, with the variable whose value is to be changed determined by using some local measure of "progress". This exploration continues for a fixed number of steps, unless a satisfying assignment is found in the process. "Re-starts" are also allowed, with the algorithm choosing a new assignment and starting its exploration from scratch.

It seems as though other kinds of algorithmic ideas could potentially be useful as well, but there has been little rigorous analysis of alternatives to DLL and local search. One exception, jumping ahead, is Williams' algorithm for ACC^0-SAT [38], which uses dynamic programming.

3 Algorithms for k-SAT and Lower Bounds for Depth-3 Circuits

To the best of my knowledge, the first instance in the literature where a connection is explicitly drawn between upper bounds for SAT and circuit lower bounds is a paper by Paturi, Pudlak and Zane [30] giving probabilistic savings n/k for k-SAT. They also derandomize their algorithm to achieve savings $n/2k$. The inspiration for their algorithm and analysis is a lemma which they call the "Satisfiability Coding Lemma". This lemma is then used by them to give tight bounds for the circuit size of unbounded fan-in depth-3 circuits computing Parity.

Before describing their ideas, it might be good to step back a bit and give some general intuition for why there are connections between non-trivial SAT algorithms and circuit lower bounds. Suppose we wish to design a non-trivial algorithm for C-SAT, where C is some natural class of circuits. For example, k-SAT corresponds to C being the class of depth-2 circuits with bottom fan-in bounded by k, and CNF-SAT corresponds to C being the class of depth-2 circuits. Intuitively, in order to design and analyze a non-trivial algorithm, we require some understanding of the structure of instances. Suppose we are able to isolate some special property that the instances to our problem share, eg., some property common to all k-CNFs, then we might be able to exploit this to achieve savings over brute-force search. The point is that the same property also indicates some *limitation* of the circuit class C under consideration, and by identifying a Boolean function f which does not have this property, we can prove a lower bound against C. Thus, it is fundamental to this connection between upper bounds and lower bounds that SAT is a *meta-algorithmic* problem - the instances to the problem are themselves computational objects, such as formulas or circuits.

Of course, the key to achieving good upper bounds as well as tight lower bounds is identifying the right property. The Satisfiability Coding Lemma shows that *isolated* solutions to k-CNFs have short descriptions on average, and hence that there can't be too many of them. Here an isolated solution is a satisfying assignment such that none of its neighbours in the Hamming cube are satisfying assignments to the same formula. Note that the property identified in the Satisfiability Coding Lemma is rather specialized. Parity, for example, has 2^{n-1} isolated solutions. Indeed Parity is in a sense the function that violates the property in the Satisfiability Coding Lemma most drastically, and intuitively this is why the Lemma is also useful in proving tight circuit size lower bounds for Parity.

To describe the Lemma more precisely, we need some notation. Given a formula ϕ on n variables and an integer $j, 0 \leqslant j \leqslant n$, call a satisfying assignment w to the variables of ϕ j-isolated if exactly j neighbours of y in the Hamming cube are not satisfying assignments to ϕ. An isolated solution is one that is n-isolated.

Lemma 4. *[30] There are polynomial-time computable functions Enc and Dec such that the following holds. Let ϕ be a k-CNF formula on n variables, and w be a j-isolated solution to the variables, where $0 \leqslant j \leqslant n$. Then*

$Dec(Enc(\phi, \pi, w)) = w$ for any permutation π on $[n]$, and moreover, on average over uniformly random choice of π, $|Enc(\phi, \pi, w)| \leqslant n - n/k$.

The intuition behind the proof of Lemma 4 is that isolated solutions lead to many *critical* clauses. Given a solution w, a critical clause is one for which exactly one of the literals is true. An isolated solution w has at least n critical clauses, one for each assignment to a variable in w. If there were a variable without a critical clause corresponding to it, then flipping the value of that variable would result in a satisfying assignment, contradicting the fact that w is isolated.

Critical clauses can be used to save on variables when searching the space of solutions. Let w be an isolated solution. Imagine a process where variables are chosen in a random order and set in ϕ to their value in w, excepting when there's a unit clause containing that variable. If there's a unit clause, the variable is set to satisfy that clause. The point is that if variables are chosen in random order, then for a critical clause of length k, there is a probability at least $1/k$ that the variable (say x) corresponding to the true literal in that clause is chosen last. In this case, the clause has already been reduced to a unit clause by the time x is set, and therefore x is *forced* rather than having to be set by w. So we don't need to store the value of x in w - in some sense, it can be recovered from the formula itself. Since there at least n critical clauses, on average at least n/k variables are forced in this process, and hence an isolated solution can be compressed to only store values of variables that are not forced, which saves n/k bits. In general, for a j-isolated solution, j/k bits are saved, using the same argument. This essentially gives the proof of Lemma 4.

It is easy to imagine how Lemma 4 can be used to achieve savings for Unique-k-SAT, the version of k-SAT where there's a promise that the input formula has either zero or one satisfying assignments. Clearly, any satisfying assignment in such a case is isolated, and hence it can be compressed on average. Intuitively, one just needs to search the compressed representations to find a solution if one exists, and this reduces the size of the search space to $2^{n-n/k}$ from 2^n.

A variation of this argument actually gives the same upper bound for k-SAT without any restriction on number of satisfying assignments. Consider a k-CNF ϕ. If there is a solution w which is j-isolated for large j, then it can be compressed by Lemma 4 and hence can be found much more quickly than brute-force search. If on the other hand, if all solutions are only j-isolated for small j, then intuitively there are *many* solutions, which means that a random solution is likely to work. In the paper by Paturi, Pudlak and Zane, this tradeoff idea is exploited nicely to prove the following result.

Theorem 5. *[30] k-SAT has probabilistic savings n/k.*

This was a huge improvement over the previous best known result for general k, which only gave savings $n/g(k)$ for some exponential function g. Because I wished to highlight how the algorithmic result takes advantage of the Satisfiability Coding Lemma, I focussed on the ideas in the analysis, and wasn't specific about the actual algorithm used. In fact, the algorithm designed by Paturi, Pudlak and Zane is a very natural and simple DLL algorithm. The algorithm repeatedly does the following: set the variables in ϕ in a random order to random values, except when there is a unit clause and the current variable is forced. It is no coincidence that this algorithm is similar to the encoding process used to prove Lemma 4!

Lemma 4 implies that there are at most $2^{n-n/k}$ isolated solutions to a k-CNF, and this can be used to give depth-3 circuit size lower bounds for Parity, where the circuits have bottom fan-in bounded by k. The argument is very simple: a depth-3 circuit with bottom fan-in bounded by k is an OR of k-CNFs (the circuit can be assumed to have top gate OR without loss of generality). Since Parity has 2^{n-1} isolated solutions but each k-CNF can only have $2^{n-n/k}$ isolated solutions, the circuit needs to have at least $2^{n/k-1}$ gates. This bound is tight up to a constant factor. By a slightly more involved argument, Paturi, Pudlak and Zane show the following for general depth-3 circuits computing Parity.

Theorem 6. *[30] The depth-3 circuit size of Parity is $\theta(n^{1/4}2^{\sqrt{n}})$.*

The upper bound in Theorem 6 is given by a very natural divide-and-conquer strategy: break the variables up into blocks of size $\sqrt{n} - \log(n)/4$, compute the parity within each block, and then compute the parity of the resulting values.

Paturi, Pudlak, Saks and Zane [29] showed an improvement to Theorem 5 by using Resolution in a pre-processing step before applying the Paturi-Pudlak-Zane algorithm. Essentially, they try to increase the number of critical clauses in a formula. Note that if some variable in an isolated solution occurs in more than one critical clause, then in a random permutation of variables, the probability that it occurs last in *some* critical clause is larger than $1/k$, and so better compression of isolated solutions can be achieved than in Lemma 4. They prove that the repeated use of Resolution to derive all possible clauses of some bounded width (where the bound is $o(\log(n))$) from the original formula actually does yield benefits.

Theorem 7. *[29] For each $k \geqslant 3$, there is a constant $\mu_k > 1$ such that k-SAT has probabilistic savings $\mu_k n/(k-1)$.*

As with the Paturi-Pudlak-Zane result, the proof of this theorem gives a structural characterization of k-CNFs in terms of the maximum possible number of sufficiently isolated solutions. Here a sufficiently isolated solution is one such that there is no other solution within a given distance of it. This characterization was used to give the first depth-3 circuit size lower bound of the form $2^{c\sqrt{n}}$ for an explicit function, where $c > 1$.

Theorem 8. *[29] There is an explicit Boolean function f in* P *such that f does not have depth-3 circuits of size* $2^{\pi\sqrt{n}/\sqrt{6}-\sqrt{n}/\log(\log(n))}$.

A further example of a structural property of CNFs which is relevant both to algorithmic questions and to lower bounds is the Sparsification Lemma of Impagliazzo, Paturi and Zane [22] which says that every k-CNF can be written as the disjunction of $2^{\epsilon n}$ linear-sized k-CNFs, for arbitrarily small $\epsilon > 0$. I do not discuss this further here because the Sparsification Lemma does not directly give an improved algorithm for a natural variant of SAT. However, it has been quite influential in the structural theory of SAT, specifically with regard to the robustness of the Exponential Time Hypothesis (ETH), which states that 3-SAT is not solvable in time $2^{o(n)}$. It is also useful in proving certain kinds of depth-3 circuit lower bounds.

4 From Algorithms for Circuit-SAT to Circuit Lower Bounds

In the previous section, I described an informal connection between SAT algorithms and lower bounds - the Satisfiability Coding Lemma can be used both to analyze a natural algorithm for k-SAT and to prove tight lower bounds on the size of depth-3 circuits solving Parity. In this section, the spotlight is on the recent breakthroughs of Ryan Williams [37, 38]. Williams made two major contributions. First, he proved that non-trivial algorithms for C-SAT imply that $\mathsf{NEXP} \not\subseteq \mathsf{C}$ for a wide range of natural circuit classes C. This makes the connection between algorithms and circuit lower bounds formal, and also generic, in the sense that it opens up the possibility of using the algorithmic approach to prove a variety of new circuit lower bounds. Second, he gave a "proof-of-concept" for this novel approach by using it to show that $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$, a brand-new circuit lower bound. This involved designing and analyzing a non-trivial algorithm for ACC^0-SAT.

To give intuition for the formal connection from SAT algorithms to circuit lower bounds, I first describe a simpler version of the result, which has an easy proof. Williams' connection is best understood as a refinement of this simpler result.

Suppose we have a polynomial-time algorithm for SAT. Then it is easy to see that EXP does not have polynomial-size circuits. If $\mathsf{EXP} \subseteq \mathsf{SIZE}(poly)$, then by the classical Karp-Lipton-Meyer theorem [25] relating non-uniform inclusions of EXP to uniform collapses, $\mathsf{EXP} \subseteq \Sigma_2^p$. Now, by our assumption that SAT is in P, we have that $\mathsf{NP} = \mathsf{P}$, and hence that $\Sigma_2^p = \mathsf{P}$. But these collapses together imply that $\mathsf{EXP} = \mathsf{P}$, which is a contradiction to the deterministic time hierarchy theorem [18, 19]. Hence the assumption that $\mathsf{EXP} \subseteq \mathsf{SIZE}(poly)$ must be false.

This is an example of an *indirect diagonalization* argument. An implication is proved by showing that its negation implies a contradiction to a hierarchy theorem. Such arguments have proven very useful in various contexts in structural complexity theory, including uniform lower bounds for the permanent [2], time-space tradeoffs [13, 12], a Karp-Lipton style result for NEXP [20] and separations against advice [6].

How far can this argument be stretched? If we try and use it to show that EXP does not have subexpontial-size circuits, we run into the issue that subexponential functions are not closed under composition. Indeed, if SAT is in SUBEXP, we have that $\mathsf{NP} \subseteq \mathsf{SUBEXP}$, but this does not imply that $\Sigma_2^p \subseteq \mathsf{SUBEXP}$. The best we can say is that $\Sigma_2^P \subseteq \mathsf{NSUBEXP}$, by replacing the inner co-nondeterministic polynomial-time part of a Σ_2^p computation with a deterministic subexponential-time computation. But this is not enough to derive a contradiction to a hierarchy theorem, as all we get using the additional assumption that $\mathsf{EXP} \subseteq \mathsf{SIZE}(poly)$ is that $\mathsf{EXP} \subseteq \mathsf{NSUBEXP}$.

Perhaps we can salvage a superpolynomial size circuit lower bound for NEXP instead? Indeed this is the case. As hinted before, the analogue of the Karp-Lipton-Meyer theorem for NEXP is known - it was proved by Impagliazzo, Kabanets and Wigderson [20]. Their argument is a clever indirect one using pseudo-randomness in a critical way (though the statement of the result itself does not mention randomness!). At this point, we just need the result, not the proof technique. However, as we shall see, the Impagliazzo-Kabanets-Wigderson proof technique plays an important role in the derivation of Williams' connection.

Let us now re-do the old argument to establish a circuit lower bound from the weaker assumption that there is an algorithm for SAT running in time $2^{n^{o(1)}}$. The circuit lower bound we get from this assumption is that $\mathsf{NEXP} \not\subseteq \mathsf{SIZE}(poly)$. Assume, to the contrary, that $\mathsf{NEXP} \subseteq \mathsf{SIZE}(poly)$. Then, by the Impagliazzo-Kabanets-Wigderson result, we have that $\mathsf{NEXP} = \Sigma_2^p$. Now, SAT in time $2^{n^{o(1)}}$ implies that $\mathsf{NP} \subseteq \mathsf{SUBEXP}$, and therefore that $\Sigma_2^p \subseteq \mathsf{NSUBEXP}$. Combining this with the collapse for NEXP, we have that $\mathsf{NEXP} \subseteq \mathsf{NSUBEXP}$, which is a contradiction to the non-deterministic time hierarchy theorem [10, 35, 41, 14].

The implication we have just proved is folklore. It wasn't given much significance because it does not represent a viable route to proving circuit lower bounds - few believe that SAT can be solved in sub-exponential time. Indeed, the Exponential-Time Hypothesis of Impagliazzo, Paturi and Zane [22] stating that 3-SAT cannot be solved in time $2^{o(n)}$ is widely believed.

On the surface, it doesn't look like there is much hope for getting an implication for circuit lower bounds from a much weaker algorithmic assumption for SAT, such as solvability in time $2^{n/2}$. Such a simulation seems "fragile" in that it doesn't compose with polynomial-time reductions to give a non-trivial simulation for all of NP, so it seems unlikely that the method of indirect diagonalization can be used.

However, it turns out that is is still possible to use the method, and a key factor in getting things to work is the parametric view of SAT, i.e., making a distinction between the size of the instance and the number of variables. Williams [37] proved the following theorem.

Theorem 9. *[37] If there is a non-trivial algorithm for Circuit-SAT, then* $\mathsf{NEXP} \not\subseteq \mathsf{SIZE}(poly)$.

It is somewhat surprising that such a weak algorithmic assumption already yields lower bounds, and just the implication is interesting in itself. But what makes it more interesting is the possibility of actually proving circuit lower bounds this way. As per the current state of knowledge, there is no indication that Circuit-SAT is unlikely to have a non-trivial algorithm. After all, we are only asking to save over brute-force search by a superpolynomial factor in the running time. Indeed, as it later turned out, a more general version of Theorem 9 yielded new lower bounds against ACC^0.

The proof of Theorem 9 combines several known facts and ideas in a clever way, including the completeness of the Succinct-3SAT problem for NEXP, local checkability and the easy witness method [20].

The high-level idea is still to use indirect diagonalization. Consider an arbitrary language $L \in \mathsf{NTIME}(2^n)$, and assume that $\mathsf{NEXP} \subseteq \mathsf{SIZE}(poly)$. We use the presumed non-trivial algorithm for Circuit-SAT to solve L non-deterministically in time $2^n/\omega(1)$. This contradicts the non-deterministic time hierarchy theorem, which has as a consequence the existence of a language L in $\mathsf{NTIME}(2^n)$ but not in $\mathsf{NTIME}(2^n/\omega(1))$.

Let x be an instance for the language L such that $|x| = n$. We first use the NEXP-completeness of the Succinct-3SAT problem to reduce x in polynomial time to a circuit C of size $\mathsf{poly}(n)$ with $n + O(\log(n))$ input bits. C implicitly encodes a 3CNF formula ϕ_C of size $2^n\mathsf{poly}(n)$ such that ϕ_C is satisfiable iff $x \in L$. By an implicit encoding here, we mean that given an index i into the binary representation of the formula ϕ_C, C outputs the i'th bit of the representation of ϕ_C.

We can't apply the presumed Circuit-SAT algorithm directly to ϕ_C since it is too large. Instead, we will work with the implicit encoding. The easy witness method [20] shows that if $\mathsf{NEXP} \subseteq \mathsf{SIZE}(poly)$, then every positive Succinct-SAT instance has a succinct witness, meaning that there is a circuit C' of size $\mathsf{poly}(n)$ and with $n + O(\log(n))$ inputs such that C' is the implicit encoding of a satisfying assignment to the formula encoded by the instance. Applying this to our context, we have that there is a circuit C' of size $\mathsf{poly}(n)$ which implicitly encodes a satisfying assignment to ϕ_C.

Now we can apply the guess-and-check paradigm: guess a circuit C' and check that the assignment encoded by C' indeed satisfies ϕ_C. The check that the assignment satisfies the formula can be done naturally in co-non-deterministic polynomial time: Universally guess a clause of ϕ_C and check using three calls to the circuit C' (each call recovering one bit of the succinct witness) that the clause is indeed satisfied by the assignment encoded by C'. The key point here is that this is a co-non-deterministic computation with only $n + O(\log(n))$ guess bits, since that many guess bits suffice to identify a clause of ϕ_C.

At this point, we use our algorithmic assumption and replace the co-non-deterministic computation by a deterministic one. Using the non-trivial algorithm for Circuit-SAT, we can implement the co-non-deterministic computation in time $2^n/\omega(1)$, since the co-non-deterministic computation is equivalent to solving a Circuit-SAT instance of size $\mathsf{poly}(n)$ with parameter $n + O(\log(n))$. By putting together the guess of the circuit C' with this computation, we get a non-deterministic algorithm which decides correctly whether $x \in L$ in time $2^n/\omega(1)$ as desired, yielding a contradiction to the non-deterministic hierarchy theorem.

Hopefully, this description clarifies how this argument is a much more refined version of the arguments giving the simpler implications. The Karp-Lipton-Meyer collapse appears here implicitly in our use of local checkability, and we use a much tighter version of the non-deterministic time hierarchy than is required for the simpler implications. The explicit use of the easy witness method is a new ingredient, though it appeared indirectly in our earlier argument since it underlies the Karp-Lipton-Meyer style collapse for NEXP [20].

Though Theorem 9 is interesting, it hasn't yielded any lower bounds yet as we do not know any non-trivial algorithms for Circuit-SAT. In the follow-up paper [38] which showed $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$, Williams significantly generalized Theorem 9 to apply to any circuit class satisfying some natural conditions.

Theorem 10. *[38] Let* C *be any circuit class which is closed under composition, contains* AC^0 *and is contained in the class of general Boolean circuits. If* C*-SAT has a non-trivial algorithm, then* NEXP *does not have polynomial-size circuits from* C*.*

Examples of classes C to which Theorem 10 applies include AC^0, ACC^0 and NC^1. Thus it gives a generic approach towards proving circuit lower bounds of interest.

Why doesn't the proof technique of Theorem 9 suffice to establish Theorem 10? The reason is that the reduction from $x \in L$ to a circuit C doesn't yield circuits that are structured enough. It is unclear whether the variant of Succinct-SAT where the circuits encoding the exponential-length formula are constant-depth circuits is still NEXP-complete. Williams gets around this by using the assumptions that C-SAT has a non-trivial algorithm and that NEXP has polynomial-size circuits from C a second time in a clever way.

More specifically, assume for the purpose of contradiction that NEXP has polynomial-size circuits from C, and that C-SAT has a non-trivial algorithm. Since C is a sub-class of Boolean circuits, we have that $\mathsf{NEXP} \subseteq \mathsf{SIZE}(poly)$. As before, we consider an arbitrary language $L \in \mathsf{NTIME}(2^n)$ and reduce a given instance x of L to a circuit C encoding an exponential-length CNF such that the CNF is satisfiable iff $x \in L$. The circuit C is not in general an ACC^0 circuit, and this is where the new idea comes in: we guess an *equivalent* polynomial-size ACC^0 circuit D and check during the co-non-deterministic computation that D is in fact equivalent to C by using local checkability together with the non-trivial algorithm for ACC^0-SAT. We also guess a polynomial-size ACC^0 circuit D' representing an "easy witness". The point is that since by assumption $\mathsf{NEXP} \subseteq \mathsf{ACC}^0$, we also have that $\mathsf{P} \subseteq \mathsf{ACC}^0$ and this implies that the circuits C and C' in the old proof have equivalent ACC^0 circuits D and D'. In the case of D, we actually need to check that it is equivalent to C, but as mentioned, this can be done using the algorithmic assumption. The rest of the argument is the same as before - once we have D and D' which are ACC^0 circuits, the co-non-deterministic computation checking if the easy witness satisfies the formula encoded by D can be simulated deterministically in time $2^n/\omega(1)$ using the assumption of a non-trivial algorithm for ACC^0-SAT. Note that D and D' are guessed together, and the check of whether D is equivalent to C is performed before the check of whether the assignment encoded by D' satisfies the 3CNF encoded by D. What we get in the end is a non-deterministic algorithm for deciding x which runs in time $2^n/\omega(1)$, yielding a contradiction to the non-deterministic time hierarchy as before.

While Theorem 10, it could have been the case that for some fundamental reason, this approach to new lower bounds was not viable. Williams' greatest contribution was to give a "proof of concept" by using his approach to show that $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$. The biggest circuit class for which super-polynomial size lower bounds were known for NEXP previously was $\mathsf{AC}^0[p]$ - the class of constant-depth circuits with modular counting gates where the modulus is a prime. In fact, the lower bounds against $\mathsf{AC}^0[p]$ are for explicit Boolean functions in P [32, 36], however the full power of NEXP seems necessary to achieve Williams' lower bound.

Williams' algorithm for ACC^0-SAT is innovative even from the algorithmic viewpoint, as it uses algorithmic ideas which hadn't been explored before in the context of algorithms for SAT. The first algorithm he came up with was a rather involved one using a result of Coppersmith about matrix multiplication. Following on a suggestion of Bjorklund, he later came up with a much simpler algorithm which uses dynamic programming, and this is the one I discuss. The algorithm relies on a well-known structural property of polynomial-size ACC^0 circuits [40, 9, 2] - the fact that they can be simulated by quasi-polynomial-size depth-2 SYM+ circuits. A SYM+ circuit is a circuit where the bottom layer is composed only of ANDs of small fan-in and the top gate is a symmetric gate. An additional property that is required is that these depth-2 SYM+ circuits can be constructed efficiently from the original ACC^0 circuits, and the top symmetric gate can be efficiently evaluated.

The algorithm is not non-trivial in the sense we defined before, but using the proof of Theorem 10, it does imply that $\mathsf{NEXP} \subseteq \mathsf{ACC}^0$ since it runs in time $2^{n-\omega(\log(n))}$ on circuits of size $\mathsf{poly}(n)$.

Theorem 11. *[38] There is an algorithm for p-ACC^0-SAT running in time $O(2^{n-n^{\Omega(1)}})$ when $m = \mathsf{poly}(n)$.*

I now sketch the proof. Let C be an ACC^0 circuit of size $m \leqslant n^c$ with n inputs, where c is a constant. Let $l < n$ be a parameter which will be fixed later. First, convert C to an equivalent circuit C' of size $m2^l$ on $t = n - l$ variables by enumerating all possible assignments on the first l variables and taking a big OR of the resulting 2^l copies of C. Note that C' is still an ACC^0 circuit. Let $s = m2^l$. Next, convert C' to an equivalent depth-2 circuit C'' of size $s' = s^{\log^k(s)}$, where k is a constant. This can be done in time $O(s^{\log^{O(1)}(s)})$ using a result of Allender and Gore [2].

The key lemma is that a SYM+ circuit of size s' on t variables can be evaluated on all possible truth assignments to the variables in time $O((s' + 2^t)\mathsf{poly}(t))$. Note that this is superior to brute-force search in that the circuit size and the 2^t term are related additively rather than multiplicatively. This gives a significant advantage when the circuit size s' is large, as it is in our case.

Given the key lemma, we are done by choosing $l = n^\epsilon$ for ϵ sufficiently small. This is because, by the lemma, the SYM+ circuit can be evaluated on all possible truth assignments in time $O((2^{n^{\epsilon+k\epsilon+o(1)}} + 2^{n-n^\epsilon})\mathsf{poly}(n))$, which is $O(2^{n-n^\epsilon})$ when $\epsilon = 1/(k+2)$.

To prove the key lemma, we use dynamic programming. Essentially, we need to keep track of which AND gates evaluate to 1, in order to evaluate the symmetric function. We initialize a look-up table which states for every subset S of the input variables, the number $f(S)$ of AND gates which have precisely this subset as input. This initialization can be done in time $O((s' + 2^t)\mathsf{poly}(t))$. We then compute the *zeta transform* g of f using a standard dynamic programming algorithm, where for any subset T, $g(T)$ is the sum over all subsets $S \subseteq T$ of $f(T)$. For each T, $g(T)$ is the number of AND gates evaluating to 1 on the input which is 1 for precisely those input bits in T. This gives all the information required to evaluate the symmetric gate on that input. Thus we simultaneously obtain the answers of the circuit for all candidate assignments in time $O((s' + 2^t)\mathsf{poly}(t))$, proving the key lemma.

The Williams results raise the intriguing question of whether there are inherent barriers to proving lower bounds in this fashion. Progress on lower bounds using more traditional techniques has been halted by several barriers, including the relativization barrier [7], the natural proofs barrier [33] and the algebrization barrier [4]. None of these barriers seem to apply directly to the approach via algorithms. This is not necessarily cause for hope, but it is cause not to be pessimistic!

Of course, the viability of the approach depends on the existence of non-trivial algorithms (or algorithms at least good enough to be able to apply Theorem 10 for C-SAT, where C is a broader class of circuits than ACC^0. The jury is still out on this, but there's certainly a strong motivation now to develop the structural theory of the exact complexity of SAT variants, with the goal of understanding in which situations non-trivial algorithms are likely to exist.

5 Improved SAT Algorithms using Lower Bound Techniques

The results of Williams discussed in the previous section take advantage of a formal connection from algorithms to lower bounds. It is natural to ask whether there is a connection in the reverse direction - can lower bound techniques be used to design and analyze SAT algorithms?

In Section 4, I described structural properties of CNFs which were useful both in designing algorithms and proving lower bounds. The results in this section will have a slightly different flavour. Standard lower bound techniques will be used as inspiration to design SAT algorithms improving on brute-force search. No formal connection will be established, but using lower bounds as inspiration will have significant payoffs nevertheless.

While k-SAT and CNF-SAT have been widely studied, and improvements over brute-force search are known, until recently nothing non-trivial was known for Formula-SAT, where there is no restriction on the depth of the input formula. A year and a half ago, Santhanam [34] gave a simple deterministic algorithm which achieved savings $\Omega(n^3/m^2)$ for Boolean formulae over the de Morgan basis. Note that the savings is $\Omega(n)$ for linear-size formulae. Santhanam also gave a different algorithm which achieved savings $\Omega(n^2/(m\log(n)))$ on formulae over an arbitrary basis.

Theorem 12. *[34] Formula-SAT has savings $\Omega(n^3/m^2)$.*

The same savings applies to the problem of *counting* the number of satisfying assignments of a Boolean formula, using the same analysis.

The proof technique of Theorem 12 also yields a new lower bound consequence.

Corollary 13. *[34] Any linear-size sequence of formulae fails to compute Parity correctly on at least a $1/2 - 1/2^{\Omega(n)}$ fraction of inputs of length n, for all but finitely many n.*

The algorithm underlying the proof of Theorem 12 is very simple indeed. It is a DLL algorithm where the variable to be set is chosen as the most frequently occurring variable in the current formula, and the value to which it is set is chosen arbitrarily. This is a purely deterministic algorithm, however the analysis is probabilistic and uses the popular *random restriction* lower bound method as inspiration.

The random restriction method has been used to prove lower bounds in various settings, including for constant-depth circuits and Boolean formulae [3, 15, 16, 5, 17]. The basic idea is as follows. Suppose we are trying to prove a lower bound against a class C of circuits. We look at what happens when a circuit from the class is “hit” with a random restriction, meaning that some of the variables are set in a specific way. For the present, we deal with *pure* random restrictions. A pure random restriction with parameter p is a probability distribution on partial assignments to inputs which sets each variable independently to 1 with probability $(1-p)/2$, to 0 with probability $(1-p)/2$ and leaves it unset with probability p. We try to argue that when a pure random restriction is applied to the inputs of a circuit from C, the circuit "simplifies" drastically. For constant-depth circuits, this is done using the Switching Lemma [16], which says that the induced function is constant with high probability, where the meaning of "high" depends on the choice of p. For Boolean formulae over the de Morgan basis, this is done by analyze the shrinkage exponent, which is the largest constant γ so that a formula of size L shrinks to a formula of size $O(p^\gamma L)$ under a restriction with parameter p. Subbotovskaya [5] proved that the shrinkage exponent is at least 1.5, and there was a sequence of papers obtaining improvements until Hastad proved that the shrinkage exponent is exactly 2 [17]. Indeed, the current best formula size lower bound of $n^{3-O(1)}$ for an explicit function is based on Hastad's result.

How do random restrictions connect to DLL algorithms? There is a superficial similarity in that processes involve variables being set incrementally, but in fact the connection goes deeper. In both processes, the notion of "simplification" is important. A DLL algorithm stops when the formula simplifies to "true" and backtracks when it simplifies to "false". The hope is that not too much backtracking is required before finding a satisfying assignment, if one exists. In the case of random restrictions, simplification of the formula is key to the technique being usable to prove lower bounds. The more drastic the simplification, the more limited the circuit class is, in some sense, and hence the better the lower bounds that can be shown. Quick simplification is also useful for DLL algorithms, as it means less backtracking and hence better savings over brute-force search.

This intuition can be made precise in the analysis of the DLL algorithm described above for FormulaSAT. We analyze a slightly different kind of random restriction - an *adaptive* restriction. In a pure restriction, the choice of which variables to set is made uniformly at random, and so too which values to set variables to. In an adaptive restriction, while the choice of values remains uniform, the choice of which variables to set is done adaptively depending on which variables are already set and how this setting has simplified the formula. It makes sense to study adaptive restrictions where the variables are set in the same order as they are set in the algorithm for FormulaSAT, as this gives a natural correspondence between properties of the restriction and efficiency of the algorithm. Subbotovskaya's analysis of pure random restrictions can be refined to show a *concentration bound* for simplification of formulae under such adaptive restrictions, and this concentration bound can then be used to bound the running time of the DLL algorithm. Details can be found in the paper [34].

As with the results in Section 4, the analytical technique exposes a structural property of small formulae - they have *decision trees* that are not too large. This property can be exploited to prove Corollary 13, as it is easy to see that Parity requires decision trees of size 2^n. Indeed, any leaf of a decision tree that is not at depth n is uncorrelated with Parity, which is why this argument gives a strong correlation lower bound.

The random restriction method and the DLL algorithmic paradigm have both been the subject of much interest, so it is natural to wonder whether the connection between them can be exploited further. Santhanam conjectured that an analogous argument to his could yield an improved algorithm for AC^0-SAT, as well as new correlation bounds against AC^0 circuits. There has been a spate of recent work on this. Beame, Impagliazzo and Srinivasan (manuscript) have considerably improved an old correlation bound of Ajtai [3], and designed the current best deterministic algorithm for AC^0-SAT. Independently, Impagliazzo, Matthews and Paturi [21] came up with a probabilistic DLL algorithm for AC^0-SAT achieving savings close to linear.

Theorem 14. *[21] AC^0-SAT has probabilistic savings $\Omega(n/(\log(m/n))^{d-1})$.*

The analysis of the Impagliazzo-Matthews-Paturi algorithm extends and refines the Hastad switching lemma, and gives a new structural characterization of AC^0 functions in terms of partitions of the Hamming cube into subcubes where the function is constant. An optimal correlation bound for Parity against constant-depth circuits follows from this characterization, in a similar way to how Corollary 13 follows from Theorem 12.

Corollary 15. *[21] AC^0 circuits of size s fail to compute Parity correctly on at least a $1/2 - 1/2^{\Omega(n/(\log(m/n))^{d-1})}$ fraction of inputs, for n large enough.*

A similar correlation bound was obtained independently by Hastad (manuscript).

The above results exploit a connection between DLL algorithms and random restrictions. Are there other lower bound techniques that can be harnessed algorithmically? This is an intriguing question about which little is known. Santhanam's algorithm for formulae over an arbitrary basis can be interpreted as utilizing a connection between the algorithmic paradigm of *memoization* and the Neciporuk lower bound technique in complexity theory, but I do not know of any other results along this direction.

6 Speculation

The recent papers on SAT algorithms and lower bounds have opened up what promises to be a very fruitful area of research. There are many research directions that look interesting, and in this section I will give a personal selection.

Perhaps the most exciting questions arise from the work of Williams. His lower bound against ACC^0 circuits is for a Boolean function in NEXP. The lower bounds we know against weaker classes are all for functions in P. This is a major discrepancy - can we prove a similar lower bound for a much more explicit function? It seems that techniques somewhat different from Williams' will be required. Perhaps the limitations of the circuit class ACC^0 which are exposed by his algorithm for ACC^0-SAT could be exploited in a more direct fashion, giving a more explicit bound.

Another very natural question is to derive lower bounds against larger classes of circuits. This motivates the exploration of new algorithmic paradigms for SAT, such as dynamic programming and graph sparsification.

In terms of the reverse connection from lower bounds to algorithms, it would be interesting to identify if there is any "algorithmic content" in other common lower bound techniques such as the polynomial method and the Khrapchenko method. New analyses for DLL algorithms have been found by constructivizing the proofs that random restrictions simplify formulae, and perhaps other lower bound proofs could be constructivized in a similar way. In an optimistic scenario, this would lead to new algorithmic methods that could be used elsewhere.

In the Boolean complexity world, the connections between algorithms and lower bounds have only been studied so far in the context of the Satisfiability problem. There are various other NP-hard problems, such as Clique, Colouring, Subset Sum etc. for which improved algorithms beating brute-force search are an active topic of study. Could any of the lower bound connections help in analyzing these problems? An immediate obstacle to doing this is that none of these problems are inherently meta-algorithmic, unlike SAT. But maybe the use of alternative notions of complexity, such as graph complexity, could provide some insight here.

Connections analogous to those in the Boolean complexity setting could exist in the arithmetic complexity setting as well. Specifically, it is quite conceivable that algorithms for the Polynomial Identity Testing problem marginally beating brute force search could lead to new arithmetic complexity lower bounds, and this possibility ought to be explored further.

To reiterate, the complicity between lower bounds and algorithms could provide a way around the obstacles to which complexity theorist, and to a lesser extent algorithmists, are so accustomed. But the maps we can draw at this stage are of necessity rough, unformed. All we can do is to believe that the deep mysteries mask a deeper sense.

References

[1] S. Arora and B. Barak. *Complexity Theory: A Modern Approach.* Cambridge University Press, Cambridge, 2009.

[2] Eric Allender and Vivek Gore. A uniform circuit lower bound for the permanent. *SIAM Journal on Computing*, 23(5):1026–1049, 1994.

[3] Miklos Ajtai. Σ_1^1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24:1–48, 1983.

[4] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC '08)*, 2008. To appear.

[5] B.A.Subbotovskaya. Realizations of linear functions by formulas using and, or, not. *Soviet Mathematics Doklady*, (2):110–112, 1961.

[6] Harry Buhrman, Lance Fortnow, and Rahul Santhanam. Unconditional lower bounds against advice. In *Proceedings of 36th International Colloquium on Automata, Languages and Programming*, pages 195–209, 2009.

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

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

[9] Richard Beigel and Jun Tarui. On ACC. *Computational Complexity*, 4:350–366, 1994.

[10] Stephen Cook. A hierarchy for nondeterministic time complexity. In *Conference Record, Fourth Annual ACM Symposium on Theory of Computing*, pages 187–192, Denver, Colorado, 1–3 May 1972.

[11] Evgeny Dantsin and Edward Hirsch. Worst-case upper bounds. In H.van Maaren A.Biere, M.Heule and T.Walsh, editors, *Handbook of Satisfiability.* 2008.

[12] Lance Fortnow, Richard Lipton, Dieter van Melkebeek, and Anastasios Viglas. Time-space lower bounds for satisfiability. *Journal of the ACM*, 52(6):833–865, 2005.

[13] L. Fortnow. Time-space tradeoffs for satisfiability. *Journal of Computer and System Sciences*, 60(2):337–353, April 2000.

[14] Lance Fortnow and Rahul Santhanam. Robust simulations and significant separations. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming*, pages 569–580, 2011.

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

[16] Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 6–20, 1986.

[17] Johan Hastad. The shrinkage exponent of de morgan formulas is 2. *SIAM Journal on Computing*, 27(1):48–64, 1998.

[18] Juris Hartmanis and Richard Stearns. On the computational complexity of algorithms. *Trans. Amer. Math. Soc. (AMS)*, 117:285–306, 1965.

[19] Frederick Hennie and Richard Stearns. Two-tape simulation of multitape Turing machines. *Journal of the ACM*, 13(4):533–546, October 1966.

[20] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences*, 65(4):672–694, 2002.

[21] Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for AC0. In *Proceedings of Symposium on Discrete Algorithms*, page To appear, 2012.

[22] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 62(4):512–530, 2001.

[23] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, pages 220–229, 1997.

[24] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *Proceedings of the 35th Annual ACM Symposium on the Theory of Computing*, pages 355–364, 2003.

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

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

[27] Ketan Mulmuley. On p vs np and geometric complexity theory: dedicated to sri ramakrishna. *Journal of the Association of Computing Machinery*, 58(2), 2011.

[28] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

[29] Ramamohan Paturi, Pavel Pudlak, Mike Saks, and Francis Zane. An improved exponential-time algorithm for k-sat. In *Proceedings of 39th International Symposium on Foundations of Computer Sciece (FOCS)*, pages 628–637, 1998.

[30] Ramamohan Paturi, Pavel Pudlak, and Francis Zane. Satisfiability coding lemma. In *Proceedings of 38th International Symposium on Foundations of Computer Science (FOCS)*, pages 566–574, 1997.

[31] Prasad Raghavendra. Optimal algorithms and inapproximability results for every csp? In *ACM Symposium on Theory of Computing (STOC)*, pages 245–254, 2008.

[32] Alexander 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.

[33] Alexander Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997.

[34] Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *Proceedings of 51st Annual IEEE Symposium on Foundations of Computer Science*, pages 183–192, 2010.

[35] Joel Seiferas, Michael Fischer, and Albert Meyer. Separating nondeterministic time complexity classes. *Journal of the ACM*, 25(1):146–167, January 1978.

[36] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the 19th Annual Symposium on Theory of Computing*, pages 77–82, 1987.

[37] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing*, pages 231–240, 2010.

[38] Ryan Williams. Non-uniform ACC circuit lower bounds. In *Proceedings of 26th Annual IEEE Conference on Computational Complexity*, pages 115–125, 2011.

[39] Andrew Yao. Theory and application of trapdoor functions. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

[40] Andrew Yao. On ACC and threshold circuits. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 619–627, 1990.

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