Complexity Lower Bounds from Algorithm Design

(Invited Paper)

R. Ryan Williams
Computer Science & Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139
Email: rrw@mit.edu

***Abstract*—Since the beginning of the theory of computation, researchers have been fascinated by the prospect of proving impossibility results on computing. When and how can we argue that a task cannot be efficiently solved, no matter what algorithm we try to use?**

In this short article, I will briefly introduce some of the ideas behind a research program in computational complexity that I and others have studied, for the last decade. (The accompanying talk will contain more details.) The program begins with the observations that:

(a) Computer scientists know a great deal about how to design efficient algorithms.

(b) However, we do not know how to prove many weak-looking complexity lower bounds.

It turns out that certain knowledge we have from (a) can be leveraged to prove complexity lower bounds in a systematic way, making progress on (b). For example, progress on faster circuit satisfiability algorithms (even those that barely improve upon exhaustive search) automatically imply circuit complexity lower bounds for interesting functions.[1]

I. Introduction

The area of computational complexity lower bounds is chock full of bad news. Not only is "bad news" the *goal* of complexity lower bounds (we want to prove that interesting tasks cannot be solved efficiently) but the desired bad news has its own bad news: there are substantial collections of barrier results (such as relativization [1], natural proofs [2], algebrization [3], and locality [4]) demonstrating broadly how various methods in complexity theory are not simultaneously subtle enough and powerful enough to prove theorems along the lines of $\mathsf{P} \neq \mathsf{NP}$ (and significantly weaker results). In short, we cannot prove lower bounds, and we can prove that we can't prove lower bounds without significantly new ideas.[2]

In recent years, some progress in complexity lower bounds has been made by taking a particular *algorithm design* viewpoint to the lower bound problem. Let $\mathcal{A}$ be a "weak" class of algorithms; we want to prove there are interesting tasks not solvable by any algorithm in $\mathcal{A}$. The idea is this: if we can design procedures that can take an arbitrary algorithm A from $\mathcal{A}$ as input, and say something interesting about the behavior of A, then this procedure can be applied to prove limitations on the computational ability of algorithms from $\mathcal{A}$.

To get an idea of how this may be possible, let us look closely at what happens in worst-case algorithm design. Let D be some domain of inputs and outputs (in computational complexity these days, it is typical to simply assume that D is a set of bit strings: either $D = \{0,1\}^n$ for a fixed n, or $D = \{0,1\}^\star$). When we design a worst-case algorithm A for computing a task $T : D \to D$, we are asserting that a sentence of the following type is true:

$$(\forall x \in D)[A(x) = T(x)]. \tag{1}$$

If we want to prove that *no* algorithm A from a set of algorithms $\mathcal{A}$ solves task T, we want to prove a sentence of the following type:

$$(\forall A \in \mathcal{A})(\exists x_A \in D)[A(x_A) \neq T(x_A)]. \tag{2}$$

That is, for all algorithms A of type $\mathcal{A}$, there is a "bad input" x_A on which A fails to compute T.

For the moment, let us suppose the existential quantifier of sentence (2) can be somehow eliminated and/or ignored (basically, we assume that bad inputs x_A are not a problem to construct). Then, there is a superficial similarity between the two above sentences:

- (1) universally quantifies over all x in the domain D, and checks that A computes T on x.
- (2) universally quantifies over all algorithms A from $\mathcal{A}$, and checks that A does not compute T.

For a closer similarity, suppose the x's in our domain D *are* algorithms from $\mathcal{A}$. Then, these two sentences would have the same quantifier domains as well. In particular, imagine our task T is *meta-computational*, meaning that its domain D consists of *descriptions of algorithms* from $\mathcal{A}$, and the task to be solved analyzes the function computed by the given algorithm from $\mathcal{A}$. For a trivial example, suppose we fix a function f to lower-bound, and consider the task T_f which accepts the description of an algorithm A from $\mathcal{A}$ if and only if A does *not* compute f. Then, if we managed to prove that

$$(\forall A \in \mathcal{A})[T_f(A) \text{ accepts}] \tag{3}$$

we would prove that no algorithm in $\mathcal{A}$ can compute the function f. Note that if (3) is true, then there is a trivial algorithm for computing T_f: simply accept every input! However,

[1]Supported by NSF grants CCF-1741615 and CCF-1909429.

[2]There is also a research program (called Geometric Complexity Theory) which endeavors to show lower bounds for arithmetic computation, pioneered by Mulmuley and Sohoni [5], [6]. It has had some successes (for example, [7]) but it has also recently been hampered by its own barrier results [8]–[10].

978-1-6654-4895-6/21/$31.00

"meta-computational" task T_f is not very useful, in that we have merely rephrased the lower bound problem: the trivial algorithm accepting every input "solves" T_f if and only if f can't be computed by any algorithm in $\mathcal{A}$. We want to study other meta-computational tasks, interesting in their own right, where non-trivial algorithms will provide insight into lower bounds against $\mathcal{A}$.

II. Circuit Lower Bounds From Algorithms

Analyzing non-trivial properties of functions computed by general Turing machines is generally undecidable, by Rice's Theorem [11]. So we have to be careful about what kinds of meta-computational tasks we will study, in order to hope for designing a "non-trivial" algorithm. Instead of Turing machines which can take arbitrarily long inputs, we consider computational models which can only take a fixed finite number of inputs. With this switch, many interesting analysis tasks become decidable (generally falling in the range of NP, coNP, and other complexity classes).

A canonical example is the Boolean circuit satisfiability problem. Determining whether a given Turing machine accepts at least one input string is well-known to be undecidable (in fact, it is many-one complete for the recursively enumerable languages). But determining whether a given Boolean circuit accepts at least one input is NP-complete. For a general collection of circuits $\mathcal{C}$ (think CNFs, or 3CNFs, or formulas, or fan-in two circuits), we consider the $\mathcal{C}$-SAT problem.

> **$\mathcal{C}$-SAT:** *Given a Boolean circuit C of type $\mathcal{C}$, determine if there is an input x such that $C(x) = 1$.*

We also consider a weaker *promise* form of the $\mathcal{C}$-SAT problem.

> **$\mathcal{C}$-GAP-SAT:** *Given a circuit C of type $\mathcal{C}$ for which it is* **promised** *that either $\Pr_x[C(x) = 1] \geq 1/2$ or $\Pr_x[C(x) = 1] = 0$, determine which of the two is the case.*

We say that an algorithm "solves" $\mathcal{C}$-GAP-SAT if it always concludes the correct case for those C that satisfy the promise (it could have arbitrary behavior on C that do not satisfy it).

The $\mathcal{C}$-GAP-SAT problem is, as far as we know, **significantly** easier than $\mathcal{C}$-SAT. While $\mathcal{C}$-SAT is NP-complete for essentially all interesting $\mathcal{C}$ (including 3CNFs), $\mathcal{C}$-GAP-SAT can be solved with high probability using randomness, by simply sampling random inputs x and trying them. Indeed, assuming lower bounds that we expect to be true, $\mathcal{C}$-GAP-SAT can be solved in deterministic polynomial time! (This follows from work of Impagliazzo and Wigderson [12].)

Clearly $\mathcal{C}$-GAP-SAT can be solved by exhaustive search over all inputs. A host of theorems show that, if we can design algorithms that *barely* improve over exhaustive search, then interesting complexity lower bounds against families of $\mathcal{C}$ circuits follow.

Such implications are interesting, as there are many fascinating open questions about circuit complexity which look as if they should be easy to resolve, yet they have remain unanswered for decades. To give one example, it is open whether every problem in NP can be computed with an infinite family of linear-size circuits. More precisely, it is possible that for every problem L in NP (including those with n^{100}-length witnesses verifiable in n^{100} time) there is an infinite family of Boolean circuits $\{C_n\}$ such that for all n, C_n agrees with L on all n-bit inputs ($C_n(x) = 1 \iff x \in L$ for all $x \in \{0,1\}^n$) and the number of gates in C_n is at most $100n$. This looks like a ludicrous possibility: how could linear-size circuits be so powerful? However, we do not know yet how to prove such *non-uniform* lower bound results ("non-uniform" referring to the fact that there can be a completely different circuit for each input length n). For some intuition about how powerful non-uniform models can be, note that for *every* unary (a.k.a. tally) language L (including undecidable languages!) there is a circuit family $\{C_n\}$ computing L in the above sense.

The following theorem is representative of the current state of the art.

Theorem 2.1 (Informal): For all "typical" circuit classes $\mathcal{C}$,[3] If there is an $\varepsilon > 0$ such that $\mathcal{C}$-GAP-SAT on circuits of n inputs and 2^{n^ε} size can be decided in $O(2^{n-n^\varepsilon})$ time, then:

- [13] There are functions in nondeterministic $n^{\text{poly}(\log n)}$ time which cannot be computed with poly(n)-size $\mathcal{C}$-circuit families.
- [14] There is a $\delta > 0$ and there are functions computable in $2^{O(n)}$ time with a SAT oracle that cannot be computed with 2^{n^δ}-size $\mathcal{C}$-circuit families, not even on infinitely many input lengths.

Let us emphasize the weakness of the algorithmic hypothesis of Theorem 2.1. It is widely believed that $\mathcal{C}$-GAP-SAT should be in deterministic poly(S) time, where S is the circuit size. We are only asking for an algorithm that runs in $O(2^{n-n^\varepsilon})$ time on 2^{n^ε} size circuits, where $\varepsilon > 0$ can be as tiny as one likes. Moreover, Theorem 2.1 is a special case of more general theorems: if the running times of the $\mathcal{C}$-GAP-SAT algorithm can be reduced, then the resulting lower bounds against $\mathcal{C}$ circuits can be improved. One can even obtain nontrivial results from an algorithm that *barely* beats the 2^n cost of exhaustive search. Here is one example result:

Theorem 2.2 (cf. [15]): If GAP-SAT can be solved in $O(2^n/n^{10})$ time on poly(n)-size Boolean circuits of fan-in two, then there are functions in nondeterministic exponential time that do not have polynomial-size circuits.

While the algorithmic hypothesis looks extremely plausible given that a polynomial-time algorithm is believed to exist, the resulting lower bound consequence of Theorem 2.2 is a longstanding open problem (see, for example, [16]). As above, Theorem 2.2 is a special case of a more general set of connections between $\mathcal{C}$-GAP-SAT algorithms and $\mathcal{C}$ circuit lower bounds [17], [18].

[3] Here we leave the notion of "typical" undefined, but it is a very minimalist set of conditions.

We can get stronger lower bounds from algorithms for the $\#SAT$ problem, which count the number of satisfying assignments to a given circuit.

> **$\#\mathcal{C}$-SAT:** *Given a Boolean circuit C of type $\mathcal{C}$, output the number of x such that $C(x) = 1$.*

(In fact, for the results below we do not need an algorithm solving #SAT in its full generality: we only have to approximate the fraction of satisfying assignments to a circuit within an additive constant, something which can be done in randomized polynomial time, and is also believed to be solvable in deterministic polynomial time.) Stronger lower bound consequences can be derived from such algorithms, such as:

Theorem 2.3 (Informal): For a wide variety of circuit classes $\mathcal{C}$, If there is an $\varepsilon > 0$ such that $\#\mathcal{C}$-SAT on circuits of n inputs and 2^{n^ε} size can be decided in $O(2^{n-n^\varepsilon})$ time, then:

- [19], [20] There are decision problems computable in nondeterministic $n^{\text{poly}(\log n)}$ time which cannot be computed in the *average case* with $\text{poly}(n)$-size $\mathcal{C}$-circuit families.
- [14] There is a $\delta > 0$ and there are decision problems computable in $2^{O(n)}$ time with a SAT oracle that cannot be computed in the *average case* with 2^{n^δ}-size $\mathcal{C}$-circuit families, not even on infinitely many input lengths.

The average-case lower bounds obtainable from $\#SAT$ algorithms in Theorem 2.3 are rather strong. Note that for any Boolean function $f : \{0,1\}^n \to \{0,1\}$ there is always a circuit C (of $O(1)$ size) such that $\Pr_x[f(x) = C(x)] \geq 1/2$: simply have C output the majority value of $f(x)$ over all inputs x. The lower bounds implications of Theorem 2.3 say that for certain decision problems f and large enough input lengths n, there are no $\mathcal{C}$ circuits C achieving $\Pr_{x\in\{0,1\}^n}[f(x) = C(x)] \geq 1/2 + \varepsilon(n)$ for tiny $\varepsilon(n) \ll 1/\text{poly}(n)$. In other words, a $\mathcal{C}$ circuit cannot do much better at guessing the value of $f(x)$ than a trivial circuit.

The best part about such theorems is that their hypotheses actually hold for some interesting circuit classes $\mathcal{C}$! Presently, the *only* way we know how to prove strong lower bounds against some infamously annoying circuit classes such ACC^0, is to start from an appropriate $\#SAT$ algorithm for ACC^0 circuits, and apply theorems such as the above.

III. Conclusion

This article is only intended to stir the curiosity of the reader. The talk at LICS will provide more details. If you cannot wait until then, the survey article [21] touches upon some of the major points, but is out of date with respect to the state of the art.

References

[1] T. Baker, J. Gill, and R. Solovay, "Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question," *SIAM J. Comput.*, vol. 4, no. 4, pp. 431–442, 1975.

[2] A. A. Razborov and S. Rudich, "Natural proofs," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 24–35, 1997.

[3] S. Aaronson and A. Wigderson, "Algebrization: A new barrier in complexity theory," *ACM Trans. Comput. Theory*, vol. 1, no. 1, Feb. 2009.

[4] L. Chen, S. Hirahara, I. C. Oliveira, J. Pich, N. Rajgopal, and R. Santhanam, "Beyond natural proofs: Hardness magnification and locality," in *11th Innovations in Theoretical Computer Science Conference*, ser. LIPIcs, vol. 151. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 70:1–70:48.

[5] K. Mulmuley and M. A. Sohoni, "Geometric complexity theory I: an approach to the P vs. NP and related problems," *SIAM J. Comput.*, vol. 31, no. 2, pp. 496–526, 2001.

[6] K. Mulmuley, "On P vs. NP and geometric complexity theory: Dedicated to sri ramakrishna," *J. ACM*, vol. 58, no. 2, pp. 5:1–5:26, 2011.

[7] P. Bürgisser and C. Ikenmeyer, "Explicit lower bounds via geometric complexity theory," in *Proceedings of the ACM Symposium on Theory of Computing*. ACM, 2013, pp. 141–150.

[8] P. Bürgisser, C. Ikenmeyer, and G. Panova, "No occurrence obstructions in geometric complexity theory," in *IEEE 57th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 2016, pp. 386–395.

[9] C. Ikenmeyer, K. D. Mulmuley, and M. Walter, "On vanishing of kronecker coefficients," *Comput. Complex.*, vol. 26, no. 4, pp. 949–992, 2017.

[10] A. Garg, C. Ikenmeyer, V. Makam, R. M. de Oliveira, M. Walter, and A. Wigderson, "Search problems in algebraic complexity, gct, and hardness of generators for invariant rings," in *35th Computational Complexity Conference*, ser. LIPIcs, vol. 169. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 12:1–12:17.

[11] H. G. Rice, "Classes of recursively enumerable sets and their decision problems," *Transactions of the American Mathematical Society*, vol. 74, no. 2, pp. 358–366, 1953.

[12] R. Impagliazzo and A. Wigderson, "*P = BPP* if *E* requires exponential circuits: Derandomizing the XOR lemma," in *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing*. ACM, 1997, pp. 220–229.

[13] C. D. Murray and R. R. Williams, "Circuit lower bounds for nondeterministic quasi-polytime from a new easy witness lemma," *SIAM J. Comput.*, vol. 49, no. 5, 2020.

[14] L. Chen, X. Lyu, and R. R. Williams, "Almost-everywhere circuit lower bounds from non-trivial derandomization," in *61st IEEE Annual Symposium on Foundations of Computer Science*. IEEE, 2020, pp. 1–12.

[15] R. Williams, "Improving exhaustive search implies superpolynomial lower bounds," *SIAM J. Comput.*, vol. 42, no. 3, pp. 1218–1244, 2013.

[16] R. Impagliazzo, V. Kabanets, and A. Wigderson, "In search of an easy witness: exponential time vs. probabilistic polynomial time," *J. Comput. Syst. Sci.*, vol. 65, no. 4, pp. 672–694, 2002.

[17] R. Santhanam and R. Williams, "On medium-uniformity and circuit lower bounds," in *Proceedings of the 28th Conference on Computational Complexity*, 2013, pp. 15–23.

[18] E. Ben-Sasson and E. Viola, "Short PCPs with projection queries," in *Automata, Languages, and Programming - 41st International Colloquium, Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 8572. Springer, 2014, pp. 163–173.

[19] L. Chen, "Non-deterministic quasi-polynomial time is average-case hard for ACC circuits," in *60th IEEE Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 2019, pp. 1281–1304.

[20] L. Chen and H. Ren, "Strong average-case lower bounds from non-trivial derandomization," in *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing*. ACM, 2020, pp. 1327–1334.

[21] R. R. Williams, "Some ways of thinking algorithmically about impossibility," *ACM SIGLOG News*, vol. 4, no. 3, pp. 28–40, 2017.