Survey: An Intuitive Introduction to Natural Proofs and Data Complexity

Shubhang Kulkarni, Ryan Davis

May 7, 2018

Contents

1 Introduction

"The general problem of mathematically proving computational lower bounds is a mystery."

- Ryan Williams, Thinking Algorithmically About Impossibility

1.1 Some Context

We use the term *lower bound* to describe the limitations on the computational intractability of a problem. The stability and security of all modern e-commerce depends on the correctness of certain lower bounds. Yet, for even the most important lower bound claims, we do not know where or how to even begin the proof for correctness. While there is strong empirical evidence that argues in favor of these claims, such as P-NP which is at the center of all computational complexity theory, we must face the fact that these are, at the moment, just conjectures. They may simply be false. Mathematics may simply not be matured enough to deal with such conjectures yet, and advancements in the future could prove catastrophic for systems built on possibly false assumptions. Thus, it is certainly plausible to assume that the formal study of lower bounds is highly important.

1.2 Algorithm Design vs Complexity Theory

As one gets acquainted with the objectives of the two fields, one starts developing an intuitive notion that algorithm designers and complexity theorists have opposing goals. The algorithm designer has to reason about when a given problem is "easy" to solve. the easiness constraint may be anything practical - solvable in polynomial time/space, polynomial sized circuit, etc. However, complexity theorists have to prove the opposite i.e. when is a given problem "not easy" to solve. One can be slightly more formal, and express the two goals as follows - For a given problem P,

Algorithm Designers : $\exists$ algorithm A which *efficiently* solves P.

Complexity Theorists : $\forall$ algorithms A, A cannot *efficiently* solve P.

Thus, the algorithm designers can prove their claims by simply showing the existence of a solution, whereas the complexity theorist must reason about all possible algorithms to the problem, even ones they'll never encounter in their lifetimes!

1.3 Barriers

So we know that lower bounds are hard to prove, but why are we completely stuck? It turns out that we have some understanding of this problem. Formally speaking, lower bound proofs must overcome certain *complexity barriers*. These

are, in some sense, "meta-theorems" about theorems/tools which one may use to prove lower bounds.

1. Relativization[1]
2. Natural Proofs
3. Algebrization

Now, complexity theorists have also shown that its possible to overcome each of these barriers, although very few have overcome all three barriers simultaneously. One of the most recent and prominent such results[2] was proved by Ryan Williams [5]

Each of the barriers has a long line of dedicated literature associated with it. The focus of this survey will be to provide the reader an intuitive explanation to the second barrier i.e. Natural Proofs, for which Razborov and Rudich received the Godel prize in '07. We prefer keeping the discussion as intuitive as possible, by providing elaborations on the motivations behind key ideas, rather than the actual formal proofs, which are quite pedagogical.

2 Intuition from Lipton

"*The obstacle is, roughly, that a large class of approaches to circuit lower bounds must prove more*"

- R. Lipton

Our ultimate goal as complexity theorists is to be able to prove lower bounds on boolean functions i.e. prove that a given function f is *complex*, for some definition of a function being complex. It seems rather instinctive that this feat may be accomplished by using special properties of the boolean function in question. (Intuitively) The obstacle, while not apparent, and not at all obvious, is that your lower bound on f must also prove that many other seemingly unrelated functions have a large complexity. The example Lipton gives is suppose you were asked to prove that a certain number is prime, but in your proof you must also show that many other unrelated numbers are also prime. Of course, this is just an analogy in order to convey the monstrosity of the barrier, not an example of the barrier itself. To his own example, Lipton adds

"*That sounds nuts.*"

Which we feel is completely fitting, for a proof that a number is prime should only be localized to that number. You prove it is prime, nothing else.

[1] Note that we've already encountered an instance of this barrier in class while studying Oracle Turing Machines where we saw $\exists A, B$ oracles such that $P^A = NP^A$ but $P^B \neq NP^B$, as proved by Baker, Gill, Solovay [4]

[2] Termed as "Progress on the Frontier" by Lipton

3 Complexity Measure Lower Bounds

Here, we slightly formalize the idea of the previous section.

Let $B_n = \{0,1\}^n$. Let $\mathcal{F}_n = \{f | f : B_n \to B_1\}$. Now, define **complexity measure** μ as any mapping having the following properties:

- $\mu(f) > 0$. Every function is assigned a positive complexity
- $\mu(x) = 1$. Where x is a literal. Thus the complexity measure of every input literal is 1
- $\mu(f \vee g) \leq \mu(f) + \mu(g)$
- $\mu(f \wedge g) \leq \mu(f) + \mu(g)$

Then, we have the following surprising (yet simple) result:

Lemma 1. *Suppose μ is a formal complexity measure and there exists a function $f \in \mathcal{F}_n$ such that $\mu(f) > s$. Then, for at least $1/4$ of all g in $\mathcal{F}_n, \mu(g) > s/4$.*

Proof. [3]

Let g be any function in $\mathcal{F}_n$. Define $f = h \oplus g$ where $h = f \oplus g$. Then,

$$f = (f \oplus g) \oplus g = h \oplus g = h \wedge g \vee \bar{h} \wedge \bar{g}$$

Thus,

$$\mu(f) \leq \mu(g) + \mu(\bar{g}) + \mu(h) + \mu(\bar{h})$$

By way of contradiction assume that

$$\frac{|\{g : \mu(g) < s/4\}|}{|F_n|} > \frac{3}{4}$$

If we pick the above function g uniformly at random, then $\bar{g}, h, \bar{h}$ are also random functions (although not independent). By the union bound we have

$$Prob[\text{All of } h, \bar{h}, g, \bar{g} \text{ have } \mu \leq s/4] > 0.$$ [4]

This implies by the definition of complexity measure that $\mu(f) < s$, which contradicts the premise. □

4 Intuition from Gowers

This section goes more into some of the intuition behind why the natural proofs argument holds, as well as a key idea behind Razborov and Rudich's proof. The reader may find it useful to refer back to this section after reading the next section on Natural Proofs.

[3] We decided to include the proof of this lemma here, as it is quite simple, and adds flavor to the discussion.

[4] The full proof of the expansion is included in the appendix

4.1 Some Simple Intuition

Without even defining anything related to natural proofs yet, let us consider the following. A very general heuristic in terms of subset densities, tells us that the probability that a random function $f \in X$ belongs to the subset $A \cap B$ where A is a simple subset (for some definition of simple e.g. easy to compute), and B is a random subset of X, is

$$P(f \in A \cap B) \approx P(f \in A) \cdot P(f \in B)$$

In other words, the density of the intersection is approximately the product of individual densities - A and B are approximately uncorrelated. Here the $\approx$ should be interpreted as "close to, with high probability". With this in mind, let P_n be the set of functions computable in less than n^k steps. If we were allowed to assume that P_n is a random(like) subset of B_n, then by the above argument,

$$P(f \in P_n \cap B_n) = P(f \in P_n) \approx P(f \in P_n) \cdot P(f \in B_n)$$

Thus $P(f \in B_n) \approx 1$! This means that a random function from the set of all functions should be simple (with high probability!).

4.2 The Rubiks Cube Problem

Let's shift our attention away from random functions and probabilities, and focus on a very familiar problem - unscrambling a rubiks cube. With this section, we would like to motivate the basic idea behind the proof of [7], which is, stated simply:

Key Idea: Random efficiently computable functions are hard to distinguish from random computable functions.

It is a well known result that any standard rubiks cube can be solved in 20 moves[5] [6]. Now, consider the following interaction protocol between two people Alice and Bob:

- Both Alice and Bob are given the following two sets:

$$R = \{r | r \text{ is a valid rubiks cube permutation}\}$$

$$R_{15} = \{r | r \text{ is atmost 15 moves away from some fixed } r_0 \in R\}$$

- Alice flips a private coin to randomly choose one of the two sets, and samples a rubiks cube from the chosen set uniformly at random. This cube is sent to Bob

[5] We haven't really defined what a "move" in this regard means, but assume that the reader is satisfied when a move is defined as a single 90 deg "slice-rotation" (any direction, any slice)

- Bob correctly determines (with zero error) which set the received cube was from i.e. whether it was from R_{15}.

Clearly $R_{15} \subset R$. To the best of our knowledge, the only way that Bob may accomplish his task is by brute force i.e., this problem is "hard[6]" as far as we know, and the set R_{15} looks like a random subset of R.

5 Natural Proofs

5.1 What's so "natural"?

The natural proof argument shows how a very "natural" methodology to prove the $P \neq NP$ lower bound will fail, hence the name. The arguments generalize as a barrier for other lower bound proofs as well, but specifically for $P \neq NP$, they show, subject to a (strongly backed) hardness assumption, certain methodologies to prove $NP \not\subseteq P_{/poly}$ will fail.

5.2 Formal Definition & Properties

Let C_n be a combinatorial property of functions on n-bit inputs. Then $C_n \subseteq F_n$ i.e. C_n can be thought of as the subset of all functions satisfying the property. We say that f_n satisfies $C_n \iff f_n \in C_n$. Equivalently, sometimes the literature denotes this as:

$$C_n(f_n) = \begin{cases} 1 & \text{... if } f_n \text{ "satisfies" } C_n \\ 0 & \text{... Otherwise} \end{cases}$$

Now, the property C_n is **Natural** if it satisfies the following two conditions

1. Constructivity
   There is a polynomial time algorithm to determine whether $f_n \in C_n$

2. Largeness
   A random $g_n \in F_n$ has a non-negligible chance of satisfying C_n. Formally, $|C_n| \geq 2^{-O(n)} \cdot |F_n|$

Actually, a C_n is natural if any of its subsets satisfy the above two properties, but for the sake of simplicity in notation, we focus our conversation to the case of the entire property satisfying the conditions.

5.3 Terminology

A combinatorial property C_n is **useful against** $P_{/poly}$ if the circuit sizes of all functions satisfying C_n are super-polynomial.

[6] If you disagree with this statement, there's nothing we can really do, as it is certainly plausible that there is an efficient solution which we just haven't encountered yet. However, the glaring empirical evidence in favour of hardness suggests that the problem is interesting

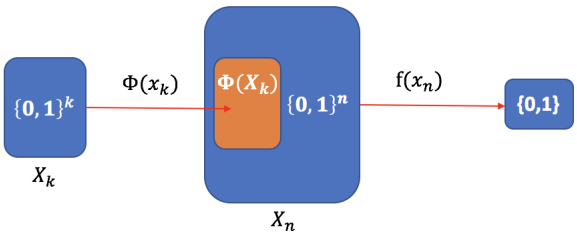

Figure 1: Psuedo-random generator

A proof that some function does not have a polynomial size circuit is **natural against** $P_{/poly}$ if the proof contains the definition of natural combinatorial properties useful against P_{poly}

5.4 The Naive Approach to $P \neq NP$

We now provide an abstract template for a way someone without the knowledge of natural proofs may go about proving $P \neq NP$. The main idea in this template is to somehow prove that $NP \not\subseteq P_{/poly}$, thereby implying that $P \neq NP$

1. Define mathematically, a notion of "discrepancy" or "scatter" of boolean function values.
   i.e. define a C_n s.t. C_n is true for functions of "high discrepancy"

2. Show (inductively) that polynomial size circuits can only compute low discrepancy functions
   i.e. C_n is useful against $P_{/poly}$ as $f_n \in C_n$ cannot be computed by poly-circuit

3. Show that (some function in NP) SAT has high discrepancy.
   i.e. $SAT \in C_n$. Thus $SAT \not\subseteq P_{/poly}$

It is a well known result that $P \subset p_{/poly}$. Thus $P \neq NP$

5.5 Psuedo-random Generators

This section just gives a brief overview of what psuedo-random generators are. The reader who already has this knowledge is encouraged to skip to next section, as to such reader we present nothing novel in this section. Let X_n and X_k be the set of n-bit and k-bit binary strings respectively. We denote x_k and x_n as binary strings chosen uniformly at random from their respective sets. See Figure 1.
Let $f : X_n \to \{0,1\}$

Let $\Phi : X_k \to X_n$
Since x_k and x_n are random strings, we denote:

$$P(f) = 1 : \text{ Probability that} f(x_n) = 1$$

$$P(f_\Phi) = 1 : \text{ Probability that} f(\Phi(x_k)) = 1$$

Then, we say that Φ is a [7]M-hard psuedo-random generator if $P(f) \approx P(f_\Phi)$. More formally, we have

$$|P(f) - P(f_\Phi|) \leq M^{-1}$$

5.6 The Breakthrough Result

The methodology described in Section 5.3 definitely looks promising, at least at an intuitive level. For years after the relativization barrier was discovered, researchers optimistically switched to circuit complexity to try and prove lower bounds. However, in November of 1996, Razborov and Rudich first published their revolutionary work on natural proofs, which pretty much washed away any attempts to prove prominent lower bounds using the above method. Their arguments are rather pedagogical, and so we just quote them on their results here. Readers are encouraged to refer to the original paper [7]

"Our main theorem ... ... gives evidence that no proof strategy along these lines [section 5.4] can ever succeed"

– Razborov, Rudich '96

They were able to prove that any Large and Constructive C_n useful against $P_{/poly}$ provides a statistical test that can be used to break any polytime psuedo-random generator, which violates a widely held belief that psuedo-random generators of hardness 2^ϵ $(\epsilon > 0)$ exist.

5.7 Implications

Recall, from section 5.2, the definition of natural proofs. We had defined such proofs to have two characteristics - Largeness and Constructivity. Thus any property used by a non-natural lower bound proof must violate at least one of the characteristics. We briefly mention what each would imply.

1. Largeness violation
   This would imply that the property used in the proof is shared by a very small number of functions. In other words, the probability that a random function possesses the property is negligible.

[7]In this case, M is actually the time taken to compute f. This may seem slightly weird, but its an "interesting" value upto a polynomial factor.

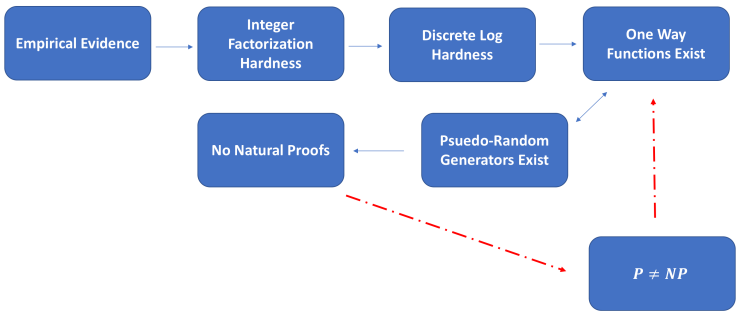

Figure 2: The Entire Picture: Blue arrows represent implications. Red arrows are (vaguely) arrows of influence.

2. Constructivity violation

   This would mean that there is no polynomial time algorithm that can decide membership in the property. This may not seem surprising until one realizes that for constructivity, the algorithm had to be polynomial in the size of the function i.e. the truth table, which is exponential in the size of the input! This would mean that the property is a highly complicated subset of all functions, one which we cannot hope to practically compute anytime soon.

6 Reminiscing and Looking Ahead

So far the discussion has been rather discouraging. Taking a step back and looking at the entire picture, we find that due to Natural Proofs, it seems that $P \neq NP$ being true, seems bad for its on provability! However, there has been recent work that tries to circumvent this barrier. In the next few sections, we present a recently introduced complexity framework, which tries to use ideas related to formal program verification, in order to link upper-bounds in the framework with lower bounds in circuit complexity.

7 A Workaround to the Barrier using Games

7.1 The Circuit-Input Game

In class we discussed zero-sum games with respect to some payoff matrix. For the game of "Rock, Paper, Scissors" and two players: Alice and Bob, we had

the following payoff matrix for Alice:

Bob \ Alice	*Rock*	*Paper*	*Scissors*
Rock	0	-1	$+1$
Paper	$+1$	0	-1
Scissors	-1	$+1$	0

If Alice plays *Rock* and Bob plays *Scissors* then Alice gets a payoff of $+1$. Respectively, Bob would get a payoff of -1 (i.e. their payoffs have *zero-sum*).

To generalize this, we can think of a zero-sum game with respect to a payoff matrix M. There will be a row player and a column player. The number of rows and columns in M will, respectively, be exactly the number of actions the row player and column player have available. For now, assume the row player has r actions available and the column player has c actions available (i.e. M is an $r \times c$ matrix). If the row player makes move i and the column player makes move j, then the payoff will be $M(i, j)$. Typically (and in our context) the goal for the row player will be to minimize $M(i, j)$ and the goal for the column player will be to maximize $M(i, j)$. You can see this intuition in the "Rock, Paper, Scissors" example. These definitions primarily come from Lipton and Young's [9].

Each "action" available to a player is known as a *pure strategy*. For example, Alice may choose *Rock* as her pure strategy and choose this move every time her and Bob play the game. Alternatively a player that has a set of pure strategies $\mathcal{I}$ available to them could choose a distribution p over $\mathcal{I}$. This distribution p is referred to as a *mixed strategy*. For instance, Alice may choose a distribution $p(\langle \textit{Rock}, \textit{Paper}, \textit{Scissors} \rangle) = \langle 0.4, 0.3, 0.3 \rangle$. You can see she slightly favors *Rock* over the other pure strategies. Then when it's game time, Alice will randomly select an action from the distribution p.

Von Neumman [10] showed that there exist optimal mixed strategies for any zero-sum game. Furthermore, he showed that revealing one's first move in a mixed strategy is not a disadvantage due to the following theorem:

Theorem 2. *Let $\mathcal{I}$ be the set of r pure strategies available to the row player and let p be some random distribution over $\mathcal{I}$. Let $\mathcal{C}$ be the set of c pure strategies available to the column player and let q be some random distribution over $\mathcal{C}$. We then have the following equivalence:*

$$\min_p \max_j \sum_i p(i) M(i, j) = \max_q \min_i \sum_j q(j) M(i, j)$$

We'll go through an example of finding the optimal strategy for the game I like to call Convoluted Primes. The row player chooses from the integer set $[0 \dots 7] \subset \mathbb{N}$. The column player chooses from the integer set $[0 \dots 3] \subset \mathbb{N}$. Then define the payoff matrix to be the following:

$$M(i, j) = \begin{cases} 1 & \text{if } (i + j + \textit{cardinality}(i \cdot j)) \text{ is prime} \\ 0 & \text{otherwise} \end{cases}$$

Where $cardinality(x)$ is equivalent to the number of 1 bits in the binary representation of x (e.g. $cardinality(7) = 3$). The matrix M can be seen below:

$M =$

$i \setminus j$	0	1	2	3
0	0	0	1	1
1	0	1	0	0
2	1	0	1	1
3	1	0	1	0
4	0	0	1	0
5	1	0	0	0
6	0	0	0	1
7	1	1	0	1

Von Neumman's original **maximin** [10] [8] technique gives some guidance on how to compute the optimal mixed strategies of the game. I will omit the details and go ahead an present the optimal mixed strategy for the row player:

$$p(\langle 0,1,2,3,4,5,6,7\rangle) = \langle 0, \tfrac{1}{4}, 0, 0, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, 0\rangle$$

The optimal mixed strategy for the column player:

$$q(\langle 0,1,2,3\rangle) = \langle \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}\rangle$$

These can be shown to be optimal by satisfying Theorem 2. The keen observer may also realize that these strategies result in a Nash Equilibrium [8]. Furthermore, the expected payoff when both players play optimally is the *value* of the game, denoted by $\mathcal{V}(M)$. In our game, this value is $\frac{1}{4}$. Naturally, the fact that the value of the game favors the row player slightly should make sense, because there are slightly more 0's in the matrix than 1's.

This is great news, an optimal strategy exists, and it doesn't even matter if we play first! There is a catch to this, the *size* of optimal mixed strategy could be linear in the number of pure strategies. This may not seem so bad, but let's think about how these games could model computation problems. We turn to a cryptographic game between a bank and a thief. The bank wants to ensure, with some confidence, that even if a thief can see someone's encrypted bank information, that the thief can not retrieve the original information. The thief, on the other hand wants to show that they can reveal the true information. The bank has used some encryption function $f : \{0,1\}^n \to \{0,1\}^m$ to encrypt user data. The bank wants to give a thief some encrypted m-bit message and see if the thief can return the original n-bit message. If the thief is correct, payoff to the bank is 0, if the thief is incorrect, payoff to the bank is 1. However, the bank can now choose from 2^m possible encrypted strings and the thief can choose from 2^n possible decrypted strings. This gives a payoff matrix of size $2^m \times 2^n$. You should now see the issue with a mixed strategy whose size is linear in the size of this game, it's exponential! If you're a smart thief who

wants to work quickly, you absolutely don't want to deal with an exponential sized strategy.

In fact, most "games" that model computational problems will have exponential size. This makes the optimal mixed strategy effectively useless if we have to represent it in exponential space. We now turn our attention to "small" *approximately* optimal mixed strategies discussed in [9]. Small strategies will be of logarithmic size with respect to the matrix. For our bank and thief example, this means the strategies for each player would be linear in n and m.

Definition 1. A mixed strategy is k-uniform if it chooses *uniformly* from a multiset of k pure strategies.

Lipton and Young prove that for a value k proportional to the *logarithm* of the number of pure strategies available to the *opponent*, we can achieve a near-optimal k-uniform strategy. Assume the minimum payoff in M is $M_{\min}$ and the maximum payoff in M is $M_{\max}$. Recall that M is an $r \times c$ matrix. Lipton and Young [9] prove:

Theorem 3. *For any $\epsilon > 0$ and $k \geq \ln(c)/2\epsilon^2$,*

$$\min_{p \in \mathcal{P}_k} \max_j \sum_i p(i)M(i,j) \leq \mathcal{V}(M) + \epsilon(M_{\max} - M_{\min})$$

Where $\mathcal{P}_k$ denotes the possible k-uniform mixed strategies for the row player. Note that the symmetric result holds for the column player. We will step through the proof [LY94] while also using our "Convoluted Primes" example to show the proof intuitively.

Proof. Assume, without loss of generality, that $M_{\max} = 1$ and $M_{\min} = 0$, which in our example is true. First, we fix a value $\epsilon > 0$, this will basically be our error. For our example we will choose $\epsilon = 0.6$, which is a rather high error, but makes our example more useful. Next we fix a value $k > \ln(c)/2\epsilon^2$. Given our choice of $\epsilon = 0.6$ and $c = 4$, we must satisfy $k > 1.92\ldots$, so we will choose $k = 2$. Next we construct our strategy S by drawing k times independently at random from the row player's optimal mixed strategy p. When doing this, I got the following:

$$S = \{4, 5\}$$

For any fixed pure strategy j of the column player, the probability that

$$\sum_{i \in S} \frac{1}{|S|} M(i,j) \geq \mathcal{V}(M) + \epsilon(1)$$

is bounded by $e^{-2k\epsilon^2} \approx 0.237$. By choice of k, we have that $e^{-2k\epsilon^2} < 1/c$. This is also true for our example $0.237 < 0.25$. Thus, the expected number of the column player's c strategies that satisfy the above inequality is less than 1. Since c is an integer, that number is 0 for some S of size k (Note it's also true for our example given the S we chose). This means we have a k-uniform strategy S that is within ϵ of optimal and is of logarithmic size! These results will lead directly to the next set of results on data complexity. □

7.2 Data Complexity

Measures of data complexity are essentially measures of program verification. If we both know the "goal" that we're trying to achieve and you give me a proposed solution, how many input/output tests do we need to verify your solution is correct? Trivially, we could test all input/output pairs, however, we are more interested in scenarios where the test suite is *small*.

We will again use a zero-sum game similar to those of our last section. The row player will have a set of inputs to choose from and the column player will have a set of circuits (i.e. programs) to choose from. If the chosen circuit computes a specific function correctly on the chosen input, the circuit player wins. If the chosen circuit does not compute a specific function correctly, the input player wins. This is the general game that we will call the *circuit-input game*.

Definition 2. The *circuit-input game* with respect to a payoff matrix M is as follows. Let $\mathcal{C}$ be the set of all circuits of size s. Note that $|\mathcal{C}| = 2^{O(s \log s)}$ because we can represent any size s circuit in $O(s \log s)$ bits. Let $\mathcal{I}$ be the set of all n-bit strings. Note that $|\mathcal{I}| = 2^n$. Let M be a matrix with 2^n rows (for $\mathcal{I}$) and $2^{O(s \log s)}$ columns (for $\mathcal{C}$). For any chosen circuit $C \in \mathcal{C}$ and any input $x \in \mathcal{I}$ let

$$M(C, x) = \begin{cases} 0 & \text{if } C(x) = f(x) \\ 1 & \text{otherwise} \end{cases}$$

where f is some function on bit-strings, $f : \{0,1\}^* \to \{0,1\}$.

The function f can be thought of as the decider of some language. If $f(x)$ returns 1 if and only $x \in L$ for some language L, our goal is essentially to test if the circuits in $\mathcal{C}$ can decide some *slice* of L. We say slice here because a circuit C has a fixed input size (number of input nodes). Therefore, if C has n bits of input, we want to check if C correctly decides all n-bit strings for inclusion in L. With this, we can define *data complexity* [3] .

Definition 3. The *data complexity* of testing size-s circuits for a function f to be the minimum cardinality set S of labeled input/output examples $(x, f(x))$ (where $|x|$ can range from 1 to s) that suffice to distinguish all size-s circuits which do not compute a slice of f from those which do.

Note that if we were doing black box testing, then this test suite would trivially be $O(2^s)$ because we would need to test all input output pairs to test circuits with certainty. However, we do have some information about the circuit, it's *size*. This can be thought of as a kind of gray box testing where we have the circuit size as side information. Also note that the data complexity is with respect to the *circuit size* s, **not** the input size n. This allows us to think of the circuits as the input to our testing problem, represented by $O(s \log s)$ bits.

Using our circuit-input game above, along with some notions of data complexity, we can restate the above theorems by Lipton and Young to get the following theorem by Williams and Chapman [3]:

Theorem 4. *Let $\epsilon > 0$, let $k > \dfrac{\ln|\mathcal{I}|}{2\epsilon^2}$, and let $\ell > \dfrac{\ln|\mathcal{C}|}{2\epsilon^2}$.*

1. *There exists a k-uniform distribution p on the set of circuits $\mathcal{C}$ such that for every input $x \in \mathcal{I}$, the expectation $\boldsymbol{E}_{C\sim p}[M(C,x)] < \mathcal{V}(M) + \epsilon$.*
2. *There exists a ℓ-uniform distribution q on the set of inputs $\mathcal{I}$ such that for every circuit $C \in \mathcal{C}$, the expectation $\boldsymbol{E}_{x\sim q}[M(C,x)] > \mathcal{V}(M) - \epsilon$.*

From this theorem we can get the following consequence related to data complexity [3]:

Theorem 5. *Let $\mathcal{C}_n$ be a set of 2^t circuits where each circuit has n inputs, let $\mathcal{I}_n \subseteq \{0,1\}^n$ (a set of n-bit inputs), and let $f : \{0,1\}^n \to \{0,1\}$. Let ϵ be some small error in $(0,1]$, and let p and q be values in $[0,1]$ such that $p + q \leq 1 - \epsilon$. Then one of the following must be true:*

1. *There exists an $O(n/\epsilon^2)$-size multiset of circuits $X_n \subseteq \mathcal{C}_n$ such that for every input $y \in \mathcal{I}_n$, $C(y) = f(y)$ for more than a p-fraction of the circuits $C \in X_n$.*
2. *There exists an $O(t/\epsilon^2)$-size multiset of inputs $Y_n \subseteq \mathcal{I}_n$ such that for every circuit $C \in \mathcal{C}_n$, $C(y) \neq f(y)$ for more than a q-fraction of the inputs $y \in Y_n$.*

The values of p and q can be thought of as a trade off between the success of the circuit player and the success of the input player. If we increase p, clearly the circuit player's strategy X_n computes f for a larger fraction of the circuits in X_n. It is a natural question to ask if $p + q = 1$ is a valid equality for the theorem to hold. Without going into the proof, it turns out that this is not possible, you need some non-zero epsilon as error.

7.3 Circuit Lower Bounds via Data Complexity

Using the previous theorems along with the circuit-input game, we can get some pretty interesting results for circuit lower bounds. The main idea is that circuit lower bounds are equivalent to data complexity upper bounds. The theorem is as follows [3]:

Theorem 6. *Let $f : \{0,1\}^* \to \{0,1\}$, and let $S(n) \geq 2n$ for all n.*

1. *If $f \in SIZE(S(n))$ then the data complexity of testing size-s circuits for f is* at least $2^{\Omega(S^{-1}(s))}$ *almost everywhere.*
2. *If $f \notin SIZE(n \cdot S(n))$ the the data complexity of testing size-s circuits for f is* at most $O(2^{S^{-1}(s)} + S^{-1}(s) \cdot s^2 \log s)$ *infinitely often.*

This is to say, (1) if the circuit complexity of f is relatively small then the data complexity of f is relatively large and (2) if the circuit complexity of f is large, then the data complexity small. The very high level intuition about this,

is that the size of circuit required to compute f reveals information to us when we are testing these circuits.

To get a little more specific with our intuition, imagine if we replaced "data complexity" in the above theorem with "time complexity". For functions f computable in exponential time, when the circuit complexity is large, the time complexity of testing those circuits is low.

Suppose that $S(n)$ is the circuit complexity lower bound for computing a function f on n bits of input. Now let's say we're given a circuit C of size s with n inputs. Clearly if $s < S(n)$ we can immediately reject because the circuit is smaller than the circuit lower bound. Otherwise, we may need to test all $2^n < 2^{S^{-1}(s)}$ inputs for our circuit and check if the output matches f on every instance. To clarify the function $S^{-1}(s)$, because I struggled to internalize what it meant at first, we can think of $S(n)$ as a mapping from the *number of inputs* to the *size of the circuit*. Thus, we can think of $S^{-1}(s)$ as a mapping from the *size of the circuit* to the *number of inputs*. This means, when we test all $2^n < 2^{S^{-1}(s)}$ inputs, it is upper bounded by $2^{S^{-1}(s)}$, a function of circuit size s. This is what we are looking for, because we're wanting to measure the time complexity in terms of *input size*, and the input to our testing problem is the circuit C of size s. For f computable within $2^{O(n)}$, this algorithm takes $2^{O(S^{-1}(s))}$ time. Therefore, larger $S(n)$ (circuit lower bounds) implies smaller $S^{-1}(s)$, which implies smaller running times $2^{O(S^{-1}(s))}$ with respect to s. Thus, our intuition, larger circuit lower bounds imply lower running time for testing. We will go through the proof of the above theorem in two parts.

Proof. (Part 1) Suppose for all n, there is a circuit C of size $S(n)$ that computes f_n on all inputs of length n. We then claim that every test set $T_{s'}$ for f on circuits of size $S'(n) = S(n) + n$ satisfies $|T_{s'}| \geq 2^n$. This means, if we add n more gates to our circuit than absolutely necessary, our data complexity must be at least 2^n, for the following reason. For any circuit C of size $S(n)$ that computes f_n, there exists a circuit C_x of size $S'(n) = S(n) + n$ which agrees with C on all inputs except $x \in \{0,1\}^n$. That is to say, with n additional gates we can hard-code into the circuit C_x, a tree of gates that outputs 1 if and only if the input is x. We then take the XOR of that tree with circuit C. This will be a circuit C_x of size $S(n) + n$ which agrees with C on all inputs except x. Thus for any circuit C computing f_n with size at most $S'(n)$, we must include all x of length n in $T_{s'}$ to distinguish C from all other C_x. That is, all x of length $\Omega(S^{-1}(s))$ must be in $T_{s'}$. Therefore, if $f \in \text{SIZE}(S(n))$, the data complexity must be at least $2^{\Omega(S^{-1}(s))}$.

(Part 2) It might be useful to recall Theorem 5 discussed earlier for this part. Suppose the circuit complexity of f is greater than $n \cdot S(n)$ on n inputs. Then we claim that there cannot be a collection $\mathcal{D}$ of $O(n/\epsilon^2)$ size-$S(n)$ circuits such that for all $x \in \{0,1\}^n$, $C(x) = f(x)$ for more than a $1/2$ fraction of circuits C in $\mathcal{D}$. Otherwise, we could take the MAJORITY of these circuits in $\mathcal{D}$ and get a circuit $C_{\mathcal{D}}$ that computes f. Specifically the size of $C_{\mathcal{D}}$ would be at most $n \cdot S(n)$, which contradicts our assumption of the circuit lower bound on f.

Therefore, item 1 of Theorem 5 does not hold with $p = 1/2$, hence item 2 must hold with $q \leq 1/2 - \epsilon$.

If we set ϵ appropriately in item 2 of Theorem 5 we can get that for all input lengths m ranging from n to $n \cdot S(n)$, let $\mathcal{I}_m = \{0,1\}^m$, let $\mathcal{C}_{S(n)}$ be all circuits of size $S(n)$, then:

There exists an $O(S(n) \log S(n))$-size multiset of inputs $X_{m,s} \subseteq \mathcal{I}_m$ such that for every circuit $C \in \mathcal{C}_{S(n)}$, $C(x) \neq f(x)$ for more than a 1/10-fraction of the inputs $x \in X_{m,s}$.

Note that the fraction 1/10 comes from our choice of ϵ and q. The cardinality of $O(S(n) \log S(n))$ comes from the fact that a circuit of size $S(n)$ can be represented with $O(S(n) \log S(n))$ bits (e.g. using an adjacency list). Thus there are $2^{O(S(n) \log S(n))}$ circuits of size $O(S(n) \log S(n))$ and since the whole point of item 2 of Theorem 2.2 was to get a logarithmic size strategy (of inputs) for the input player we get a strategy (multiset of inputs) of size $\log(2^{O(S(n) \log S(n))}) = O(S(n) \log S(n))$. Thus, we must add all strings in all $X_{m,s}$ to our test suite X_s, to refute any circuit with more than n inputs and size $S(n)$. This results in X_s of size $O(n \cdot S(n)^2 \log S(n))$.

Now this only covered input lengths m ranging from n to $n \cdot S(n)$. We must cover the case of input lengths $m < n$. For this case, we simply must include all m bit inputs up to length $n-1$ in the set X_s. This will increase the cardinality by 2^n. Thus our total size of X_s is $O(2^n + n \cdot S(n)^2 \log S(n))$. The catch is that we need data complexity in terms of circuit size, while it is currently in terms of input size. Thus we set the circuit size $s = S(n)$ and therefore the input size is $n = S^{-1}(s)$. We get the final data complexity of testing size s circuits for f is $O(2^{S^{-1}(s)} + S^{-1}(s) \cdot s^2 \log s)$. □

For the above theorems, the small cardinality test suites X_s have the following structure: for input sizes which are too long to support size-s circuits for f, we have small sets of counterexamples for the circuit-input game, but as the input sizes decrease, we reach a threshold where it's possible to compute f with size s circuits, thus we must start including all possible inputs and their labels.

Without going into details, from this we get a few interesting corollaries that allow us to try and separate P from NP via $NP \not\subset P_{/poly}$:

Corollary 6.1. *A function f is in $P_{/poly}$ if and only if for some $\epsilon > 0$, the data complexity of testing circuits for f is greater than 2^{s^ϵ} for almost every s.*

Corollary 6.2. *$NP \not\subset P_{/poly}$ if and only if for every $\epsilon > 0$ and for infinitely many s, the data complexity of testing size-s circuits for* SAT *is at most $O(2^{s^\epsilon}$.*

This presents some useful concepts for data complexity and how it could be possibly used as a new model for finding circuit lower bounds. This allows us to circumvent natural proofs to some extent via data complexity.

7.4 Future Work

There are still a couple questions regarding data complexity that would be interesting for future work. The following are presented in the paper as a reasonable follow-up:

1. Can new circuit lower bounds be proved, based on the guidance of data complexity upper bounds? This is mostly theoretical work up to this point, it would be interesting see the use of data complexity in the wild per se.

2. Can the equivalence of Theorem 6 be tightened further? There is currently a gap between functions in SIZE($S(n)$) and SIZE($n \cdot S(n)$). Is this factor n really necessary? Perhaps we can push the gap one way or the other.

3. How does the complexity (classes) of f relate to the data complexity of testing for f? If f is known to be in some complexity class(es), what can we say about the complexity classes of testing for f?

8 References

1. Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009.

2. Ryan Williams. *Thinking Algorithmically About Impossibility*, Inivited Talk.

3. Brynmor Chapman and Ryan Williams. The circuit-input game, natural proofs, and testing circuits with data. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 263–270, 2015.

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

5. C. D. Murray and R. R. Williams. *Circuit Lower Bounds for Nondeterministic Quasi-Polytime: An Easy Witness Lemma for NP and NQP*. To appear in the 50th ACM Symposium on Theory of Computing (STOC 2018)

6. Tomas Rokicki, Herbert Kociemba, Morley Davidson, and John Dethridge. God's number is 20, 2010. http://cube20.org.

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

8. D. Avis, G. Rosenberg, R. Savani, and B. von Stengel (2010), Enumeration of Nash Equilibria for Two-Player Games. Economic Theory 42, 9-37. Online solver available at http://banach.lse.ac.uk.

9. R. Lipton, N. Young, Simple strategies for large zero-sum games with applications to complexity theory. *STOC 94*. 734-740, 1994.

10. John von Neumann. Zur Theorie der Gesellschaftspiel. *Mathematische Annalen*, 100(295-320), 1928.

11. Gowers, Tim. "How not to prove that P is not equal to NP." *Gower's Weblog*, 03 Oct. 2013, https://gowers.wordpress.com/2013/10/03/how-not-to-prove-that-p-is-not-equal-to-np/

12. Gowers, Tim. "Razborov and Rudich's natural proofs argument." *Gower's Weblog*, 07 Oct. 2013, https://gowers.wordpress.com/2013/10/07/razborov-and-rudichs-natural-proofs-argument/

13. Lipton, Richard. "Who's Afraid of Natural Proofs?." *Gödel's Lost Letter and P=NP*, 25 March, 2009, https://rjlipton.wordpress.com/2009/03/25/whos-afraid-of-natural-proofs/

9 Appendix

9.1 Lemma 1 Proof (continued)

$$P(g \cup h \cup \bar{g} \cup \bar{h}) \leq P(g) + P(h) + P(\bar{g} + P(\bar{h})$$

$$P((g \cap h \cap \bar{g} \cap \bar{h})^c) \leq (1 - P(\bar{g})) + (1 - P(\bar{h})) + (1 - P(g)) + (1 - P(h))$$

$$1 - P((g \cap h \cap \bar{g} \cap \bar{h})) \leq 4 - P(g) + P(h) + P(\bar{g} + P(\bar{h})$$

$$P(g \cap h \cap \bar{g} \cap \bar{h}) \geq P(g) + P(h) + P(\bar{g}) + P(\bar{h}) - 3$$

$$P(g \cap h \cap \bar{g} \cap \bar{h}) > 0$$

□