On the Incompressibility of Truth With Application to Circuit Complexity

Luke Tonon

November 3, 2025

1 Introduction

In this paper, we revisit the fundamentals of Circuit Complexity and the nature of efficient computation from a new perspective. We present a framework for understanding Circuit Complexity through the lens of information theory with analogies to results in Kolmogorov Complexity, viewing circuits as descriptions of truth tables, encoded in logical gates and wires, rather than purely computational devices. From this framework, we reprove some existing strong Circuit Complexity bounds, explain what the optimal circuits for most boolean functions look like structurally, give insight into new circuit bounds, and explain the aforementioned results in a unifying intuition.

1.1 Brief Classical View of Circuits

Since their introduction in the 1930s by Claude Shannon, Boolean circuits have been understood primarily as computational devices, machines that transform inputs into outputs through logical operations. This perspective, rooted in the development of electronic computers, views circuits as implementing algorithms: collections of logical gates that execute a sequence of logical operations to compute a function.

Definition 1.1.1. *A Boolean circuit C over a functionally complete basis B (typically $B = \{AND, OR, NOT\}$) is a directed acyclic graph where:*

- *Nodes are either input variables $x_1, ..., x_n$ or gates labeled with operations from B.*
- *Edges represent wires carrying Boolean values.*
- *One or more nodes are designated as outputs.*
- *Each gate computes its operation applied to its input wires.*

Definition 1.1.2. *The size of a Boolean circuit, C, denoted s or $|C|$, is the number of gates. The depth, d, is the length of the longest path from an input to an output.*

Definition 1.1.3. *A circuit, C, computes a Boolean function $f : \{0,1\}^n \to \{0,1\}$ if for every input $x \in \{0,1\}^n$, evaluating C on x (propagating values through gates) produces $f(x)$ at the output.*

In this view, a circuit is active: it performs computation by processing inputs through gates as electricity flows through a graph, much like a computer executing instructions. The size measures computational resources (number of operations), and depth measures parallel time. This view has been immensely productive, but it may also obscure a more fundamental understanding of what circuit complexity measures!

1.2 New View of Circuits as Descriptions

We begin with a simple observation: a truth table is not a mathematical abstraction, but a concrete informational object. Specifically, a string of 2^n bits.

Definition 1.2.1. *For a Boolean function $f : \{0,1\}^n \to \{0,1\}$, the truth table T_f is the 2^n-bit string where the i-th bit (under lexicographic ordering of inputs) equals $f(x_i)$.*

For example, the 3-variable function that outputs 1 only on input 000 has truth table $T_f = 10000000$ (reading inputs 000, 001, 010, ..., 111). While we typically think of truth tables as representations of boolean functions, from an information-theoretic view, they are simply data: 2^n bits arranged in a sequence. The "meaning" (a bit corresponding to input pattern i) is a convention we impose.

1.2.1 From Computation to Describing

Consider the traditional view: a circuit C computes function f by taking inputs, processing them through gates, and producing outputs. We consider an alternative view: a circuit C describes or encodes the truth table T_f.
To see this, observe that a circuit and its truth table contain the same information, merely represented differently:

Proposition 1.2.1. *A circuit, C, with s gates over basis B and a truth table T on n variables are informationally equivalent:*

- *$C \to T$: Evaluate C on all 2^n inputs to recover T (possible in time: $O(g \cdot 2^n)$).*
- *$T \to C$: Construct a circuit computing T (always possible with circuit size being exponential in the worst-case).*

Both directions are effective: given either representation, we can reconstruct the other with no loss of information.

1.2.2 Circuits as Compression

Further insight emerges when we examine the size of these representations. Consider a truth table on n variables where exactly one row outputs 1, such as the first row, and all others output 0. As a bit string, this would grow massively for large n: $T = 0000...0001$. (2^n bits, with a single 1).
As written, this requires 2^n bits to specify, but there's clearly a much more compact description. Namely, the circuit $C = AND(x_1, x_2, x_3, ..., x_n)$. We simply output 1 if the input matches the pattern and output 0 otherwise. This circuit has a remarkably small number of gates, yet compresses a much larger amount of information.

Remark 1.2.1. *The above example reframes circuits: they are compression mechanisms where the "compression algorithm" is logical structure itself. Instead of standard compression that exploit statistical patterns in data, circuits exploit logical patterns in truth values.*

1.2.3 Connection to Kolmogorov Complexity

We now see this perspective connects circuit complexity to a fundamental concept in information theory.
The Kolmogorov complexity $K(s)$ of a string s is the length of the shortest program (in some fixed universal language) that outputs s.

- Incompressible strings have $K(s) \approx |s|$: the best "program" is essentially "print s."
- Compressible strings have $K(s) \ll |s|$ as they have compressible descriptions.

As an interesting analogy:

- Kolmogorov complexity: Shortest program to print a string.
- Circuit complexity: Smallest circuit to compute a truth table.

Both measure description length in different computational models:

- Programs: sequential, Turing-complete
- Circuits: parallel, Boolean logic

Remark 1.2.2. *Reframing Shannon's original circuit lower bound: Most truth tables are incompressible data, so the circuit must essentially "print the table," similar to enumerating all cases where output is 1. Just as random strings have $K(s) \approx |s|$, random functions have circuit complexity $\approx 2^n/n$ (the best "logical print statement").*

1.3 A Unifying Intuition

Putting the details together suggests a fundamental principle that also serves as intuition of what makes solvers efficient. For the rest of the paper, we will take this principle to its deep logical consequences.

Principle 1.3.1. *Efficient computation is efficient description. An algorithm that computes a function quickly is performing a compressed encoding of that function's truth table through logical operations.*

When we ask "can this function be computed efficiently?" we are really asking "can its truth be compressed efficiently using logic?" As another example, consider a simplified view of Turing machine performing a computation. It reads an input string on the tape, and then depending on the machine's logical rules, may replace parts of the strings on the tape, and may put more characters on the tape as it performs its computation, ultimately printing a final correct output string (a true string relative to the description of the problem it solves). Additionally, we can interpret the classes of circuit models studied in Circuit Complexity as various restrictions on logic as a compression device.

2 Applying the framework to existing bounds

Having established circuits as logical descriptions of truth tables, we now revisit classical results through this new lens. Rather than prove new theorems yet, we first demonstrate that known results emerge naturally, and with potentially greater clarity from the information-theoretic perspective.
This serves two purposes: First, it validates the framework is mathematically sound, capable of recovering established facts as special cases. Second, it reveals why these results hold, not merely that they hold. The explanations we obtain are potentially more intuitive than the original proofs, suggesting that the descriptive view is not merely equivalent to the computational view, but are potentially more fundamental. We begin with the cornerstone result of circuit complexity: Shannon's 1949 lower bound.

2.1 Shannon's Lower Bound through Information Theory

Shannon proved that almost all Boolean functions require large circuits. The original approach is to count the number of distinct circuits of size s with combinatorics and compare to the number of Boolean functions 2^{2^n}. The approach then shows that, since the number of circuits of size s for large n is less than 2^{2^n}, most functions require larger circuits (specifically $\Omega(2^n/n)$ gates). We now examine how the framework recreates this bound in purely information-theoretic view.

Observation 2.1.1. *Approach this bound through description length.*

- *Recall a truth table T_f for n variables is a 2^n-bit string.*

- *A circuit C with s gates can be encoded in $O(s \log s)$ bits.*

  Proof. Each gate is specified by: (1) its type from basis B, requiring $\log |B|$ bits, (2) its two input wires, each selected from at most s previous gates/inputs, requiring $2 \log(s + n)$ bits. Total per gate: $O(\log s)$ bits. For s gates: $O(s \log s)$ bits. □

A natural question arises: If circuits of size s can describe truth tables of size 2^n, what compression ratio do they achieve?

Proposition 2.1.1. *A circuit with s gates describes a 2^n-bit truth table using $O(s \log s)$ bits. The compression ratio is:*

$$p(s, n) = \frac{\textit{[Description size]}}{\textit{[Data size]}} = \frac{s \log s}{2^n}$$

For the optimal circuit size $s = 2^n/n$ predicted by Shannon:

$$p(\frac{2^n}{n}, n) = \frac{\frac{2^n}{n} \log \frac{2^n}{n}}{2^n}$$

$$p(\frac{2^n}{n}, n) = 1 - \frac{\log n}{n}$$

Taking the limit as n approaches infinity:

$$\lim_{n\to\infty} p(\frac{2^n}{n}, n) = \lim_{n\to\infty} 1 - \frac{\log n}{n} = 1$$

As n grows, we observe that circuits of size $2^n/n$ achieve compression ratio approaching 1, essentially no compression at all. The circuit description requires nearly as many bits as the truth table itself, just as information theory intuition would predict.

Theorem 2.1. ***(Repeat of Shannon's bound)*** *Most Boolean functions on n variables require circuits of size $\Omega(2^n/n)$.*

Proof. Suppose most functions had circuits of size $s = o(2^n/n)$.
Then these circuits could be described in $O(s \log s) = o(2^n)$ bits.
However, there are 2^{2^n} distinct Boolean functions, and each function's truth table requires 2^n bits of information to specify.
By the Pigeonhole Principle, circuits describable in $o(2^n)$ bits can encode at most $2^{o(2^n)}$ distinct functions.
Therefore, most functions (the remaining $2^{2^n} - 2^{o(2^n)} \approx 2^{2^n}$ of them for large n) cannot be described by such circuits. Hence, most functions require circuits of size $\Omega(2^n/n)$. □

Remark 2.1.1. *Shannon's original proof showed that most functions need large circuits by counting. The information-oriented proof shows structurally* ***why*** *this must be so:* ***information theory forbids significantly compressing most data.***

Most truth tables are essentially "random": they have barely any structure for logic to compress. Random data is mostly incompressible. Therefore, the circuit "description" must be nearly as large as the data itself. The factor of n in the denominator ($2^n/n$ rather than 2^n) represents the minimal overhead of circuit structure: the "syntax cost" of writing gates and wires instead of raw bits. It's the best you can do even with optimal organization. This is not a non-constructive combinatorial coincidence. It is an **information-theoretic necessity**.

2.2 What Do Optimal Circuits Look Like?

Having established that most functions need $\Omega(2^n/n)$ gates, we can now ask: what is the structure of these optimal circuits? Answering this question seems mysterious from a purely combinatorial view. However, the information-theoretic view provides an intuitive prediction: **if a circuit cannot compress the truth table, it must essentially enumerate it, essentially become the table itself**.