On the Incompressibility of Truth With Application to Circuit Complexity

Luke Tonon

December 23, 2025

1 Introduction

In this paper, we revisit the fundamentals of Circuit Complexity and the nature of efficient computation from a fresh 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 re-prove some existing Circuit Complexity bounds, explain what the optimal circuits for most boolean functions look like structurally, give an explicit boolean function family that requires exponential circuits, and explain the aforementioned results in a unifying intuition that re-frames time entirely.

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 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: $O(s \cdot 2^n)$).*
- *$T \to C$: 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 [3] $K(s)$ 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

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). A Turing machine implicitly describes the solution as it performs its computation. 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 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: validates that the framework is mathematically sound, capable of recovering established facts as special cases, and 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 is potentially more fundamental. We begin with the cornerstone result of circuit complexity: Shannon's 1949 lower bound [7].

2.1 Shannon's Lower Bound through Information Theory

Shannon proved that almost all Boolean functions require large circuits. The original approach is often shown as counting 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 a 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 via entropy. Let U_n be the uniform distribution on all Boolean functions $f : \{0,1\}^n \to \{0,1\}$. Then

$$H(U_n) \;=\; \log_2(2^{2^n}) \;=\; 2^n,$$

since there are 2^{2^n} such functions. Assume for contradiction that for some $s = s(n)$, a $1 - \varepsilon$ fraction of all functions on n bits admit a circuit of size at most s, where $\varepsilon \in (0, 1/2)$ is fixed and s is significantly smaller than $2^n/n$ (to be made precise below). Let $\mathcal{F}_{\le s}$ be the set of such functions.
By hypothesis, each $f \in \mathcal{F}_{\le s}$ has a circuit C_f of size at most s, and hence a binary description of length at most $\ell(s)$. Extend this to a (partial) description scheme for *all* functions by declaring that functions $f \notin \mathcal{F}_{\le s}$ are described by some fallback representation of length at most $L := 2^n$ bits (for instance, the raw truth table).
Thus for a random $f \sim U_n$ the description length $L(f)$ satisfies

$$L(f) \;\le\; \begin{cases} \ell(s) & \text{if } f \in \mathcal{F}_{\le s}, \\ L & \text{otherwise.} \end{cases}$$

By the assumption that $\mathcal{F}_{\le s}$ has measure at least $1 - \varepsilon$ under U_n, we obtain

$$\mathbb{E}[L(f)] \;\le\; (1 - \varepsilon)\, \ell(s) \;+\; \varepsilon\, L.$$

On the other hand, by the basic entropy bound for lossless codes (or by Shannon's source coding theorem [6], using a prefix-free refinement of our encoding) any lossless description scheme must satisfy

$$\mathbb{E}[L(f)] \;\geq\; H(U_n) \;=\; 2^n.$$

Combining the two inequalities gives

$$2^n \;\leq\; (1-\varepsilon)\,\ell(s) \;+\; \varepsilon\,2^n.$$

Rearranging,

$$(1-\varepsilon)\,2^n \;\leq\; (1-\varepsilon)\,\ell(s) \quad\Rightarrow\quad \ell(s) \;\geq\; c\,2^n$$

for some constant $c > 0$ depending only on ε. Since $\ell(s) = O(s \log s)$, this forces

$$s \log s \;\geq\; c\,2^n \quad\Rightarrow\quad s \;\geq\; \Omega\Big(\frac{2^n}{\log(2^n)}\Big) \;=\; \Omega\Big(\frac{2^n}{n}\Big).$$

Thus, it is impossible for a positive fraction of Boolean functions on n inputs to be computable by circuits of size $o(2^n/n)$. Equivalently, almost all functions require size at least $\Omega(2^n/n)$, as claimed. □

Remark 2.1.1. *Shannon's original proof is often shown 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 that shows logic cannot cheat entropy**.

2.2 What Do Optimal Circuits Look Like?

Having established that most boolean 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 become the table itself**.

Observation 2.2.1. *For "random" (incompressible) Boolean functions requiring $\Omega(2^n/n)$ gates, the minimal circuit should have the structure of an "optimized lookup table". Essentially, a decision tree that systematically checks input patterns and outputs the corresponding truth value.*

This prediction is not new: it was already implicitly known through Lupanov's representation [4]. However, the framework reveals why this structure is optimal: it's similar to the most efficient way to "print" a truth table using logical gates.

Construction 2.2.1. *For any Boolean function $f : \{0,1\}^n \to \{0,1\}$, define the canonical lookup table circuit as follows:*

- *For each input pattern x where $f(x) = 1$*
  - *Create a "pattern detector": an n-input AND gate with appropriate negations.*
  - *This gate outputs 1 if and only if the input equals x.*
- *Combine all detectors:*
  - *OR together all pattern detector outputs.*
  - *This produces 1 if input matches any pattern where $f(x) = 1$.*

Construction 2.2.2. *As an example, consider the 3-variable function f with $f(101) = f(110) = 1$ and $f(x) = 0$ otherwise:*

$$Detector_1 = AND(x_1, NOT(x_2), x_3)$$

$$Detector_2 = AND(x_1, x_2, NOT(x_3))$$

$$Output = OR(Detector_1, Detector_2)$$

This is close to a direct "printing" of part of the truth table in circuit form, but it clearly scales poorly, so we should try to do better.

Proposition 2.2.1. ***(Lupanov Representation)*** *Any Boolean function f on n variables can be computed by a circuit of size at most $2^n/n + o(2^n/n)$.*
The construction in the canonical form naively uses $O(n \cdot 2^n)$ gates (one detector per true row, each with n gates). Lupanov's optimization organizes this like a decision tree: check variables sequentially, reusing intermediate comparisons across branches. For brevity and sake of not reinventing, we will not go into great detail of the construction of the representation, but cover enough to connect to the intuition.

Fix a block size $k = \lceil \log n \rceil$. Split the input as $x = (u, v)$ where $u \in \{0,1\}^k$ (the "address" block) and $v \in \{0,1\}^{n-k}$ (the "payload" block). For each $u \in \{0,1\}^k$, let $f_u(v) = f(u, v)$ denote the u-cofactor.
Step 1: Shared equality bank for the address. *Build a bank of* 2^k equality detectors

$$E_u(x) \;:=\; [\, x_{1..k} = u \,],$$

each computed as an $O(k)$-gate tree from the k address bits (using $\neg$ and $\wedge$ to check each bit, then an $\wedge$-tree to combine). These E_u's are **reused** *globally.*

The total cost of this bank is $O(2^k \cdot k) = O(n \log n)$ since $k \approx \log n$.
Step 2: Block-Shannon expansion. *We write*

$$f(x) \;=\; \bigvee_{u \in \{0,1\}^k} \big(E_u(x) \;\wedge\; f_u(v)\big).$$

This is a standard Shannon expansion but performed on a block *of $k = \Theta(\log n)$ variables at once; the savings come from sharing the E_u's across the entire circuit.*
Step 3: Realizing all cofactors f_u uniformly. *Each cofactor is a function on $n - k$ inputs. For the cost analysis, we do not synthesize each f_u separately. Instead, we* repeat the same two-level trick *on v: choose a second block of size $k' = \lceil \log_2(n-k) \rceil$, split $v = (w, z)$ with $|w| = k'$, and precompute a second equality bank $\{F_t(v) = [\, w = t \,]\}_{t \in \{0,1\}^{k'}}$ (shared across all u's). Then expand*

$$f_u(v) \;=\; \bigvee_{t \in \{0,1\}^{k'}} \big(F_t(v) \;\wedge\; \lambda_{u,t}(z)\big),$$

where each $\lambda_{u,t}$ is now a cofactor on $n-k-k'$ inputs. Iterate this block-expansion on the remaining inputs until no variables remain. At the final layer, the leaves are constants $\in \{0,1\}$: they are just the bits of the truth table of f arranged by blocks.
Cost accounting (the "short-circuited lookup table"). *At each layer ℓ we: (i) precompute a shared equality bank for a block of size $k_\ell = \Theta(\log n_\ell)$, where n_ℓ is the number of inputs remaining at that layer; this costs $O(2^{k_\ell} k_\ell) = O(n_\ell \log n_\ell)$ gates; (ii) feed these equalities into a shallow $\vee/\wedge$ spine that* selects *the appropriate subtable slice. The number of layers is*

$$L \;=\; \frac{n}{\Theta(\log n)}\,(1 + o(1)).$$

Across the entire construction we still "expose" all 2^n leaf bits of the truth table, but every layer fans them forward through a small shared *equality bank rather than through 2^n disjoint minterms. A standard summation over the layers (choosing $k_\ell \asymp \log n_\ell$) yields the total gate count*

$$(1 + o(1))\,\frac{2^n}{n}.$$

Intuitively: a naive lookup table wires $\Theta(2^n)$ minterms; the block-Shannon scheme batches *$\Theta(\log n)$ variables at a time, amortizing the address cost so that each of the 2^n table bits is delivered through only $\tilde{O}(1)$ additional gates on average, giving the $2^n/n$ factor.*

Remark 2.2.1. *Since the above representation is asymptotically optimal, intuitively, we can say that for most boolean functions, "optimized brute-force enumeration is optimal." Quite literally, the best circuit description for most truth tables is the table itself: a physical manifestation of brute-force.*

The prediction confirmed: Shannon's and Lupanov's results together say that, for the very hardest functions, optimal circuits are essentially optimized lookup tables: compressed representations that approach the theoretical limit of incompressibility. In particular, many circuits admit isomorphic rewritings, so even if we had an algorithm for computing minimum circuits for a given truth table, the circuit it returned could be viewed as an "encryption" of the function's true nature, hiding the fact that it is logically equivalent to a short-circuited LUT. A natural notion of "decryption" would be to test whether the circuit is isomorphic to an optimized lookup-table structure. In this sense, what looks like clever algebraic manipulation might be much simpler in description: it is just the best possible way to wire the truth table into gates. In this view, non-uniform efficient computability is not merely related to description length — it is defined by it: a function is efficiently computable in the circuit model precisely to the extent that its truth can be compressed into a small circuit. Time, in this picture, shows up only as a secondary constraint on how those compressed descriptions are discovered or used, not on what is in principle computable.

Remark 2.2.2. *Consider some analogies:*

- *Kolmogorov incompressibility (infinite-scale): Most strings have no short program. You can't, in general, prove or compute "this string has no short program" beyond a provability threshold.*
- *Circuit incompressibility (finite model): Most truth tables have no small circuit. They're best computed by an optimal short-circuited LUT. Asking for a general procedure that says "this specific table has no small circuit" (MCSP, or strong lower bounds) is like asking for a finite-scale analog of those incompressibility judgments.*
- *"If I had a solver for these finite-scale incompressibility instances, why would I expect it to be anything but brutally hard?"*

2.3 Revisiting the Natural Proofs Barrier

In 1993, Razborov and Rudich [5] analyzed successful circuit lower bound techniques from the 1980s and showed that if these techniques were used to prove separations like $P \neq NP$, then we could use the properties of those proofs to obtain faster distinguishers of random functions from pseudo-random functions.
Recall Natural Proofs:

- A natural lower bound is based on a property P of Boolean functions that is:
  - **Constructive**: membership in P is decidable in "easy" time (poly).
  - **Large**: a random function has P with noticeable probability.
  - **Useful**: every function in P avoids small circuits.

- Natural Proofs say: such P contradicts strong PRGs, so we shouldn't expect them if Crypto is real.

The "Incompressibility of Truth" framing says something more philosophically biting and implies a perhaps more substantial barrier.

- A constructive property P that captures many hard functions is automatically a kind of logical compression:
  - The algorithm deciding P + a short index i can serve as a succinct description of "the i-th function in the big P-set."
- If the hard functions are truly "incompressible truth," then having a short constructive handle on them is already suspicious: you've grouped many logically "random" objects into one low-complexity logical pattern.

Trying to find a constructive property that explains why a hard function is hard seems like a dead-end from the information-theoretic view if the property is too common. Suppose one crafts a constructive proof that proves a hard function is hard ("incompressible"). In that case, that proof is itself a concise logical description of the supposedly logically incompressible information inside the truth tables of the hard function. A would-be prover revealed a way to compress the incompressible: exactly what they didn't want to prove.

2.4 Revisiting Proof Complexity Struggles

We can also consider how the "Incompressibility of Truth" framing explains struggles in Proof Complexity with understanding strong proof systems. If a tautology truly is hard to prove, then it should be logically incompressible. Having an efficient general method to deterministically find/generate hard tautologies would intuitively contradict the tautology's hardness because the efficient method is a polynomial ("compressed") logical description generating and explaining objects that need super-polynomial ("incompressible") logical descriptions.

Construction 2.4.1 (Lifting the Paradox to Circuit Frege Proof Systems)**.** *Encode the contradiction "There exists a circuit (description), C, less than size s describing the truth table, T".*

$$DESC_{<s}(T) \;:=\; \exists C \left(|C| < s \;\wedge\; \forall x \in \{0,1\}^n \; C(x) = T[x] \right)$$

$$TT_{T,s} \;:=\; \mathit{Encode}\big(DESC_{<s}(T)\big)$$

$$s < \mathsf{DESC}(T) \implies TT_{T,s} \text{ is unsatisfiable (a propositional contradiction).}$$

Observe that if there were a "small" (polynomial) Circuit Frege refutation for one of these such contradictions where $DESC(T)$ is large as it corresponds to an informationally-large table, T, it appears the proof is itself an encoding of the small circuit (description) of the table that it simultaneously proves doesn't exist.

In essence, if there is a polynomial-bounded proof system that proves, for every T, that "T is incompressible," then that proof system looks to be compressing the incompressible. In Section 4, we provide functions that are maximally hard for circuits, and so they are likely good candidates for hard $MCSP$ tautologies/contradictions outlined above.

3 Investigating MCSP

Having seen how the information-theoretic framework illuminates classical results, we now examine the Minimum Circuit Size Problem (MCSP) that has long had a mysterious complexity. The decision version of this problem asks, given a truth table, T, and a size bound, s, whether the table can be computed by a circuit of at most s gates. MCSP is a meta-problem that has been studied extensively for its connections to cryptography, learning theory, and derandomization. Various conditional hardness results exist, but unconditional lower bounds have been elusive.

Definition 3.0.1. ***(Classical MCSP)*** *The Minimum Circuit Size Problem is the language* $\{(T, s) : T$ *is a truth table on* n *variables and there exists a circuit* C *with* $|C| \leq s$ *computing* $T\}$.

Lemma 3.1. *MCSP is in NP (the class of problems where solutions are verifiable in polynomial time in the size of its input [2]).*

Proof. The certificate to verify a YES instance is the circuit, C, where $|C| \leq s$, that exists to compute the truth table T.
Verification algorithm:

- First check that $|C| \leq s$ (count gates).
- For each input $x \in \{0,1\}^n$, evaluate $C(x)$ and check if $C(x) = T[x]$.
- Accept if all 2^n checks (all elements in $\{0,1\}^n$) pass.

Time complexity:

- Step 1: Checkable in $O(|C|)$ by counting each gate in the data structure holding the gates.
- Step 2: 2^n iterations, each evaluation takes $O(|C|)$ time.
- Total: $O(2^n \cdot s)$ iterations, each evaluation takes $O(|C|)$ time.

Since the input size is $|T| = 2^n$ bits, this is polynomial in the input size. Therefore, $MCSP \in NP$. □

Remark 3.0.1. *Notice what verification does: it recovers the truth table from the circuit description. This is precisely the "from* $C \to T$*" direction of proposition 1.2.1 (Information Equivalence). The certificate (small circuit) is a compressed description of the truth table. Verification is the decompression process:*

evaluate the circuit on all inputs to recover the full truth table, then check it matches. This is the same information recovery we used to establish circuits as descriptions.

3.1 The Information-Theoretic Self-Reference

In the classical view of MCSP, instances of the problem ask statements of the form: "Given a function's truth table, can it be computed efficiently?" However, this algorithmic view obfuscates what the problem is truly asking.
From the information-theoretic view, this is really asking: "Given a 2^n-bit data string T, can it be described using at most s logical operations?"

Observation 3.1.1. *In the information-theoretic view, MCSP is more accurately named the* ***Minimum Truth Table Description Problem****. It asks whether data (truth table) admits a compressed logical description (circuit less than a specific size).*

Remark 3.1.1. *This reframing reveals MCSP's meta-nature: it's a problem about logical* ***describability****. The circuit deciding MCSP is itself a description, one that must encode which truth tables are compressible and which are not.*

This is a description describing descriptions, and immediately creates a logical tension.

3.2 The Information-Theoretic Impossibility

To understand MCSP's hardness, we must be careful about which size bounds create difficulty.

Observation 3.2.1. ***(Spectrum of Compressibility)*** *Truth tables fall into a spectrum:*

- ***Highly compressible****: Circuit size $O(n^k)$ for small k. Examples include parity, 2-SAT, addition, and multiplication. These are "P-like" functions with exploitable logical structure and are likely to be problems we care about.*
- ***Slightly compressible****: Circuit sizes like $2^{n/k}$ for moderate k. Some structure, but exponential complexity. An intermediate regime that are likely more condensed lookup tables.*
- ***Essentially incompressible****: Circuit size $\geq 2^n/n$. Random-like functions that essentially require a lookup table representation.*

Despite MCSP's extensive study, it is still an open problem whether decision-MCSP lets you solve the search variant of MCSP in polynomial time.

Definition 3.2.1. *$MCSP_{SEARCH}$ is the search variant of MCSP where we want a solver to give us the minimum circuit, C, describing a table, T, instead of only telling us that such a circuit exists below some size.*
It is clear that SAT could help polynomially solve this problem as SAT can encode a satisfying assignment that encodes some valid circuit C describing a table T.

Conjecture 3.1. *$MCSP_{SEARCH}$ requires circuits of super-polynomial size (in the input size 2^n).*
Intuition:
If a circuit C of size $poly(2^n)$ solves $MCSP_{SEARCH}$, then C is effectively a short circuit description that encodes the shortest circuit description of every table (2^{2^n} of them) on n variables." This is like asking C to "compress the incompressible."

Proof idea. By Shannon's bound theorem 2.1, most truth tables ($\Omega(2^{2^n})$) are incompressible. Consider a circuit, C_n, that computes $MCSP_{SEARCH}$ for truth tables on n variables. Circuit C_n must "know" the shortest circuit description of every truth table on n variables, and this knowledge must be encoded in C_n's structure (its gates and wiring) as we can use C_n as a description to recover all the shortest table descriptions. Again, proposition 1.2.1 Information Equivalence is how we losslessly recover the information encoded by a circuit, and it's how we verify a YES instance of $MCSP$.
A circuit with $poly(2^n)$ gates can be described in: $O(poly(2^n) \cdot \log(poly(2^n)))$ bits. For any polynomial function, $poly(n)$, and sufficiently large n:
$O(poly(2^n) \cdot \log(poly(2^n)) \ll \Omega(\frac{2^{c \cdot 2^n}}{2^n} \cdot \log(\frac{2^{c \cdot 2^n}}{2^n}))$ where $\frac{1}{2} > c > 0$ and represents a sensitive "entropy fence" the circuit can't pass.
The bound here is based on the premise that C_n could use clever short-circuit lookup table strategies on information that we know from Lupanov's representation admits short-circuit lookup table structures. Therefore, a polynomial-sized C_n cannot encode enough information to correctly classify all minimum descriptions of truth tables on n variables, as it would be an impossible compression device for sufficiently large n, bypassing the inherent incompressible patterns in the many, many 2^{2^n} tables it describes. If C_n did compress them, it would contradict the fact that most of the tables it describes are incompressible in the scheme (circuit logic) it tries to use to compress them, offering a faster way of "solving" the tables in parallel than what should be possible based on Shannon's bound. □

With a relatively "small" $MCSP_{SEARCH}$ circuit, we could input any data we'd like as bits (a truth table, a book, DNA, etc) and it would efficiently output the most efficient way to store it in the circuit logic scheme. This makes $MCSP_{SEARCH}$'s efficiency not only worthwhile for problem solving, but also for essentially optimal lossless information storage. An efficient $MCSP_{SEARCH}$ circuit is an efficient universal data compressor, which sounds like a fantasy.

Remark 3.2.1. *The impossibility is fundamentally self-referential. To know the exact shortest description of all truth tables on n variables, the solver circuit*

must encode information about incompressibility, but that very information is already incompressible. As an analogy: ***Imagine I have a book that describes the most efficient way to summarize every other book, including ones of noise. How could that book do so without essentially admitting all the most efficient patterns shared in every other book?***

Conjecture 3.2. *$P \neq NP$ due to entropy.*

Proof idea. Assume, for the sake of contradiction, $P = NP$, then we could solve $MCSP_{SEARCH}$ with polynomial circuits. However, $MCSP_{SEARCH}$ requires super-polynomial circuits.
Therefore, our assumption was wrong, $P \neq NP$. □

Proposal 3.1. *To attempt to falsify the hypothesis above, we can construct multi-valued truth tables that capture the complexity of $MCSP_{SEARCH}$.*
Construct the table of the function, $f : \{0,1\}^{2^n} \to Enc(C_n)$, that defines the optimal circuit compressor, where for each row, $Enc(C_N)$ is the bit encoding of the optimal circuit description of the $\{0,1\}^{2^n}$ bits on the left-side of the row for a fixed basis, such as $\{AND, OR, NOT\}$.
This is feasible for small n and we can use modern circuit minimization algorithms to find the smallest circuit computing/describing f. If the minimal circuits obtained look more like an exhaustive description of every pattern in the tables rather than an algorithm, the hypothesis survives for small n.
We can tap into what the "Book of Books" looks like on a small scale.

4 An Explicit Boolean Function Family That Requires Exponential Circuits

We can now use the fact that circuits operate as lossless codes to prove an exponential lower bound on an explicit family of Boolean functions. To do this, we construct the Boolean function family, $\{f_{\Omega_n}\}$, where each f_{Ω_n} in the set has its output bits equal to the first 2^n bits of a Chaitin constant Ω_U. Recall that a truth table is simply an encoding of 2^n bits. A circuit for this problem decides, given a position of a bit (represented as a row in the truth table) in Ω_U, whether the bit is 0 or 1. Essentially, it solves a finite version of the halting problem. Since the exact bits of Ω_U depend on the universal prefix-free Turing machine U, the $\{f_{\Omega_n}\}$ family is infinite.

Definition 4.0.1. *A Chaitin constant Ω_U is the halting probability of a universal prefix-free Turing machine U [1], and it is algorithmically random, so the shortest program to output the first n bits of Ω_U must be of size at least $n - O(1)$.*

Theorem 4.1. *The boolean function family, $\{f_{\Omega_n}\}$, requires 2-fan-in circuits of $\Omega(2^n/n)$ gates.*

Proof. Assume, for the sake of contradiction, for each n, there exists a circuit C_n with $O(s)$ gates computing/describing f_{Ω_n}. Then, we can store each C_n with $O(s \log s)$ bits. Then, create a program, P_n, that contains the encoding of C_n and decodes it into C_n and decompresses C_n into the first 2^n bits of Ω_U, and prints them (essentially what the first part of a verifier program for $MCSP$ does, but hardcoded). This means P_n produces the first 2^n bits of Ω_U as output with $|P_n| = O(s \log s) + O(1)$.
For sufficiently large n, $O(s \log s) \ll \Omega(2^n)$ bits, so circuits encoded in $O(s \log s)$ bits for $\{f_{\Omega_n}\}$ would let us compress incompressible truth: a contradiction. Thus, the only way out is $s = \Omega(2^n/n)$. □

References

[1] Gregory J. Chaitin. "A Theory of Program Size Formally Identical to Information Theory". In: *Journal of the ACM* 22.3 (July 1975), pp. 329–340. DOI: 10.1145/321892.321894.

[2] Stephen A. Cook. "The P Versus NP Problem". In: *The Millennium Prize Problems*. Clay Mathematics Institute Millennium Prize Problem Description. URL: https://www.claymath.org/wp-content/uploads/2022/06/pvsnp.pdf.

[3] A. N. Kolmogorov. "Three approaches to the quantitative definition of information". In: *Problems of Information Transmission* 1.1 (1965). Originally published in Russian in *Problemy Peredachi Informatsii*, pp. 1–7.

[4] O. B. Lupanov. "A Method of Circuit Synthesis". In: *Problemy Kibernetiki (Problems of Cybernetics)*. Vol. 3. Moscow: Fizmatgiz, 1958.

[5] A. A. Razborov and S. Rudich. "Natural Proofs". In: *J. Comput. Syst. Sci.* 55.1 (1997), pp. 24–35.

[6] Claude E. Shannon. "A Mathematical Theory of Communication". In: *Bell System Technical Journal* 27 (1948), pp. 379–423, 623–656.

[7] Claude E. Shannon. "The Synthesis of Two-Terminal Switching Circuits". In: *Bell System Technical Journal* 28.1 (1949), pp. 59–98.